Universidade Federal do Rio Grande do Sul
Instituto de Informática
Pós-Graduação em Ciência da Computação

Migração de Agentes em Sistemas Multi-Agentes Abertos

por

Jomi Fred Hübner

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, outubro de 1995

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Hübner, Jomi Fred

Migração de Agentes em Sistemas Multi-Agentes Abertos / Jomi Fred Hübner. — Porto Alegre: CPGCC da UFRGS; 1995.
131 p.: il.

Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1995. Geyer, Cláudio Fernando Resin, orient.

1. Inteligência Artificial. 2. Inteligência Artificial Distribuída. 3. Sistemas multi-agentes. 4. Migração de agentes. 5. Identificação de papéis. 6. Aprendizado em sistemas multi-agentes. 7. Reconhecimento de processos. 8. Reconhecimento de planos. 9. Problema Produtor-Consumidor. I. Geyer, Cláudio Fernando Resin, orient. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Hëlgio Trindade
Pró-Reitor de Pesquisa e Pós-Graduação: Prof. Cláudio Scherer
Diretor do Instituto de Informática: Prof. Roberto Tom Price
Coordenador do CPGCC: Prof. José Palazzo Moreira de Oliveira
Bibliotecária chefe do Instituto de Informática: Zita Prates de Oliveira

“A ignorância é vizinha da maldade”
(provérbio árabe)

“Apliquei o coração a conhecer a sabedoria e a saber o que é loucura e o que é estultícia; e vim a saber que também isto é correr atrás do vento. Porque na muita sabedoria há muito enfado; ”
(Eclesiastes 1.17-8)

Para Ilze e Morgana

AGRADECIMENTOS

Certamente muitas pessoas contribuíram para que este trabalho pudesse ser realizado, algumas de maneira especial, às quais gostaria de agradecer especificamente.

Aos meus pais, Siegfried Bertold Reinold Hübner e Paula Hübner, que sempre incentivaram minha vida como estudante.

Aos meus orientadores, Prof. Geyer e Prof. Rocha Costa, de quem pude aprender muito, tanto academica como profissionalmente. Em especial, agradeço a disposição do Prof. Rocha Costa para discussões sobre assuntos sempre muito interessantes, alguns refletidos neste trabalho.

A todos os professores e funcionários do Instituto de Informática pelo espaço e ambiente onde pude passar bons momentos. À Profa. Rosa, pela oportunidade de participar no grupo de IA. Ao Rafa pelas boas conversas. À Aline, Milene e Neila, pela companhia.

À Universidade de Blumenau (FURB), nas pessoas de Aldírio Vicente e Prof. Celso Zipf, pelo apoio e incentivo para o curso de mestrado.

E, especialmente, a Ilze Zirbel, que tem estado ao meu lado desde muito tempo, e também por isso, sou o que sou.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

IA ..... Inteligência Artificial

IAD ..... Inteligência Artificial Distribuída

SMA ..... Sistemas Multi-Agentes

PIC ..... (Sociedade) Produtor-Intermediário-Consumirdor

DTE..... Diagrama de Transição de Estados

PI..... Processo de Interação

PRPI ..... Papel Relacionado ao Processo de Interação

RESUMO

A Inteligência Artificial Distribuída traz uma série de novas perspectivas para a computação quando considera sistemas heterogêneos, adaptativos, evolutivos, continuamente em funcionamento e abertos. Estes sistemas, chamados de sociedades, apresentam tais características por permitirem que seus componentes, chamados de agentes, migrem entre sociedades, isto é, agentes podem sair e entrar em sociedades. Sociedades abertas permitem a migração dos agentes e coloca dois tipos de problemas para o agente que está migrando: problemas de *linguagem e interação*, que concernem ao uso de expressões usadas e à maneira como as interações são organizadas na nova sociedade; e, problemas de *conhecimento e atuação*, que se referem à como um agente irá se comportar a fim de realizar justamente aquilo que a sociedade espera dele.

Este trabalho se atem aos problemas de conhecimento e atuação. Para que os agentes da sociedade possam cooperar e coordenar suas ações, é necessário que tenham conhecimento das capacidades, habilidades, desejos e planos dos outros agentes. Grande parte do conhecimento a respeito dos outros pode ser extraído dos papéis que estes podem assumir na sociedade. Assim sendo, o problema colocado para este trabalho é como os agentes da sociedade que receberam o agente imigrante e o próprio agente imigrante conhecerão/aprenderão os papéis uns dos outros. São desenvolvidos três mecanismos de identificação de papéis, bem como a comparação entre eles e sua adequação a tipos de migração. Os três mecanismos são os seguintes:

i) **Identificação de papéis por protocolo de apresentação**: é proposta uma linguagem de descrição de protocolos (LDP) e uma especificação de protocolo de apresentação nesta LDP. Os agentes que utilizam este mecanismo conseguem se identificar com rapidez, porém necessitam conhe-

cer várias informações “locais” da sociedade, o que pode ser muito restritivo para um agente migrante.

ii) **Identificação de papéis por observação e classificação**: esta solução procura classificar o agente observado em um papel de um conjunto pré-definido de papéis. Neste conjunto, os papéis são descritos por meio de processos de interação (PI). Para isto, desenvolveu-se a noção de PI. Foram desenvolvidas duas formas de proceder a classificação: construir uma especificação do agente a partir da observação das suas ações e verificar se esta pertence ao conjunto pré-definido de papéis; e, verificar se o comportamento do agente confere com as execuções possíveis para algum dos papéis pré-definidos. Este mecanismo é mais adequado para sociedades abertas e tem boa precisão no resultado apresentado, porém, a identificação do papel de um agente pode ser demorada.

iii) **Identificação de papéis por reconhecimento de intenções em planos**: este mecanismo baseia-se na existência de uma relação entre intenções e papéis. A partir das ações observadas para o agente, procura-se saber qual seu plano, sua intenção e, conseqüentemente, seu papel. Para isto foi implementado um procedimento de indução de planos. Este mecanismo também é adequado para sociedades abertas, no entanto, a identificação, embora satisfatória, nem sempre é completamente correta.

Estes três mecanismos foram testados em simulações numa implementação da sociedade Produtor - Consumidor, onde puderam ser comprovadas as características de cada um.

Palavras-chave: Inteligência Artificial, Inteligência Artificial Distribuída, sistemas multi-agentes, migração de agentes, identificação de papéis, aprendizado em sistemas multi-agentes, reconhecimento de processos, reconhecimento de planos, problema Produtor-Consumidor.

TITLE: "AGENT MIGRATION IN OPEN MULTI-AGENTS SYSTEMS"

ABSTRACT

Distributed Artificial Intelligence brings a number of new perspectives to Computing Science when heterogeneous, adaptative and evolutive systems, those under functioning and open, are taken into consideration. These systems, named societies, present these characteristics because they allow their components, named agents, to migrate within societies, that is, agents are allowed to enter and to leave societies. Agents' migration brings two kinds of problems to the migrating agent: **language and interaction** problems both related to the use of used expressions and to the way the interactions are organized in the new society; and, **knowledge and performance** problems refering to the way the agent will behave in order to accomplish exactly what society expects him to do.

This work is limited to knowledge and performance problems. In order to cooperate and coordinate their actions, the society's agents need to learn about the capabilities, abilities wishes and plans of other agents. A great part of knowledge of others can be extracted from the social roles these agents can play. Therefore, the problem posed in this work is how social agents who has received an immigrating agent and the immigrating agent himself will know and learn one another's roles. Three role identification mechanisms, and the comparison between them and their adaptation to migration types as well are developed.

The three mechanisms are the following:

i) **Role Identification by means of presentation protocol:**a language of protocol description (LPD) and a specification of presentation protocol in this LPD are proposed. The agents who use this mechanism can rapidly identify each other, however they need know a number of 'local' social information, which can be very restrictive to the migrating agent.

ii) **Role identification by means of observation and classification**: this solution tries to classify the observed agent as a role out of set of definite roles. In this set, the roles are described by means of interactional processes (IP). Therefore, the notion of IP was developed. Two ways to proceed the role classification were developed: to build the agent's especification departing from the observation of their actions and to check whether this especification belongs to a set of pre-defined roles; and to check whether the agent's behavior fits the possible executions to some predefined roles. This mechanism is more adequate to open societies and has good precision in the result presented, but, the agent's role identification can last longer.

iii) **Role Identification by means of intention and plans recognition**: this mechanism is based on the existence of a relationship between intentions and roles. By departing from the agent's observed actions, his plan, intention, consequently, his role is recognized. Therefore an induced plan procedure was implemented. This mechanism is also adequate to open societies, however, the identification, though satisfactory, is not always totally correct.

These three mechanisms were tested in simulated situations in a kind of Producer-Consumer Society implementation in which each one's characteristics could be verified.

Keywords: Artificial Intelligence, Distributed Artificial Intelligence, Multi-agents systems, agents migration, role identification, learning in Multi-agents systems, process recognition, plan recognition, Producer-Consumer Society.

1 INTRODUÇÃO

1.1 Inteligência Artificial Distribuída

Este trabalho está situado na área de Inteligência Artificial Distribuída (IAD), uma sub-área da Inteligência Artificial, que surgiu devido a uma necessidade em certos tipos de problemas onde a solução é inerentemente distribuída, seja geograficamente ou funcionalmente [GAS 88], e devido à disponibilidade de plataformas distribuídas em outras áreas da computação, como, por exemplo, redes de computadores e bancos de dados distribuídos.

A IAD acrescenta às abordagens da IA clássica, que tem como metáfora o comportamento humano individual, a metáfora do comportamento social onde os sistemas computacionais são vistos como sociedades de agentes inteligentes [GAS 88, DEM 90, SIC 92]. Esta metáfora coloca os sistemas mais próximos da realidade das sociedades humanas, onde a solução de problemas em muitos casos é resultado da cooperação de vários indivíduos. Destes sistemas, pode-se esperar que apresentem adaptabilidade, decremento no custo, aumento da eficiência e da velocidade, naturalidade de distribuição para os problemas, segurança através de redundâncias e a especialização do conhecimento em grupos semânticos [GAS 88].

A IAD é dividida em duas áreas, Solução Distribuída de Problemas (SDP) [GAS 88] e Sistemas Multi-Agente (SMA) [DEM 90]. A primeira estuda técnicas para resolver problemas específicos dividindo o trabalho entre muitos módulos que coope-

ram interagindo e trocando conhecimentos sobre o problema e a sua solução. Esta primeira abordagem é representada pelo seguinte fluxo (a partir do problema criam-se os agentes para solucioná-lo):

problema a ser resolvido
⇓
criação dos agentes
⇓
solução do problema

A segunda área (SMA) estuda o comportamento inteligente em uma sociedade de agentes *autônomos*, isto é, como coordenar seus conhecimentos, metas e planos para resolver problemas. Por agente autônomo entende-se aquele que tem sua própria existência, independente do problema a ser resolvido pela sociedade, e que age conforme seus próprios estados intencionais. Sob um aspecto externo, i. e., aquilo que se pode observar em um agente, Demazeau e Müller [DEM 93] definem um agente autônomo como uma entidade real ou virtual envolvida num ambiente, capaz de percebê-lo e representá-lo, capaz de agir dentro dele, capaz de comunicar-se com outros agentes e de exibir um comportamento autônomo.

Esta segunda abordagem é representada no seguinte fluxo (a partir dos agentes existentes procura-se cooperação para atingir a solução):

agentes existentes na sociedade
⇓
problema a ser resolvido
⇓
solução do problema

Esta abordagem apresenta as seguintes características:

- a decomposição de tarefas é feita pelos agentes;
- os agentes são autônomos;
- os agentes são hábeis em solucionar mais de um problema;
- é um sistema aberto, ou seja, agentes podem entrar e sair da sociedade quando quiserem, possibilitando que a sociedade se adapte a novas situações (problema que é tratado de forma inicial em [BER 92]); e

- o ambiente dos agentes pode ser alterado, ou seja, a organização interna do mundo dos agentes pode sofrer alterações.

A fim de que os vários agentes autônomos possam cooperar, e assim atingirem seus objetivos, é necessário que a sociedade possua organização e comunicação (cf. [WER 87]). A organização concerne à natureza e à função da sociedade, ela se propõe a facilitar a cooperação, um vez que esta não se daria de forma espontânea entre agentes autônomos. A comunicação é o principal instrumento dos agentes para conseguir coordenação de suas ações. Estas duas necessidades colocam aos projetos em ambientes de IAD os seguintes problemas (conforme [SIC 92]):

i) como formular, descrever, decompor e alocar problemas provendo resultados entre um grupo de agentes inteligentes;
ii) como habilitar agentes para comunicação e interação, e qual linguagem de comunicação e protocolo empregar;
iii) como verificar que agentes agem coerentemente em suas decisões e ações;
iv) como habilitar agentes individuais a representar e raciocinar sobre as ações, planos e conhecimentos dos outros agentes para comunicar-se com eles;
v) como reconhecer e reconciliar pontos de vista diferentes e conflitos entre grupos de agentes tentando coordenar suas ações;
vi) como projetar e construir sistemas práticos de IAD.

1.2 O problema proposto

Uma característica importante para sistemas de IAD é tornar a sociedade aberta à entrada dinâmica de agentes. Com esta característica, que em [COS 94] é chamada de *migração de agentes* (pois inclui tanto a entrada como a saída), estes sistemas tornam-se mais flexíveis, passando a solucionar problemas anteriormente não solucionáveis. Uma migração se dá ou porque um agente necessita de alguma função social que não existe na sua sociedade atual e existe na sociedade que pretende entrar

ou porque a sociedade precisa de um agente com determinada capacidade, e este somente está disponível em outra sociedade.

A migração do agente pode se dar em dois níveis; ele pode sair da sociedade onde se encontra e entrar na outra sociedade (como mostra a figura 1.1) ou pode somente entrar na outra sociedade sem sair da atual, passando a atuar em ambas (como mostrado na figura 1.2). Em ambos os casos, existe o problema da entrada do agente, todavia, somente no primeiro caso existe o problema da saída.

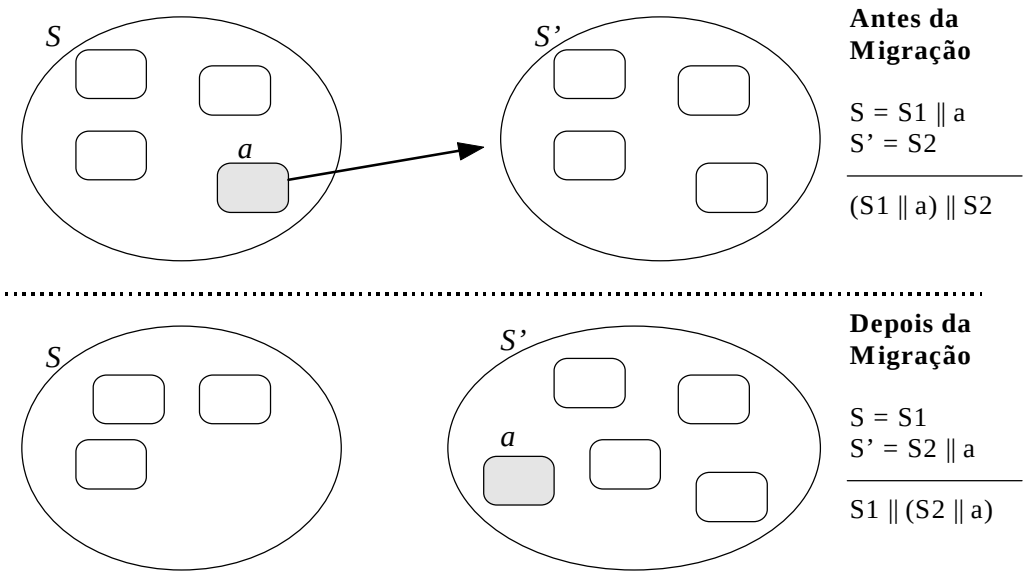

Figura 1.1 - Saída de um agente e sua entrada em outra sociedade

No processo de migração das sociedades abertas, são identificadas as seguintes etapas pelas quais passam os agentes (da entrada à saída) e suas respectivas necessidades:

i) **Colocação** do agente na nova sociedade: para isso deve existir, ou ser criado, espaço para o agente nesta nova sociedade.

ii) **Adaptação** do agente às condições de comunicação e cooperação na nova sociedade: além do agente, em alguns casos, a própria sociedade deve adaptar-se reformulando seus funcionamentos.

iii) **Atuação** na sociedade.

iv) **Saída** da sociedade: nesta etapa os outros agentes devem saber que o agente saiu e este deve ter cumprido seus compromissos com os demais.

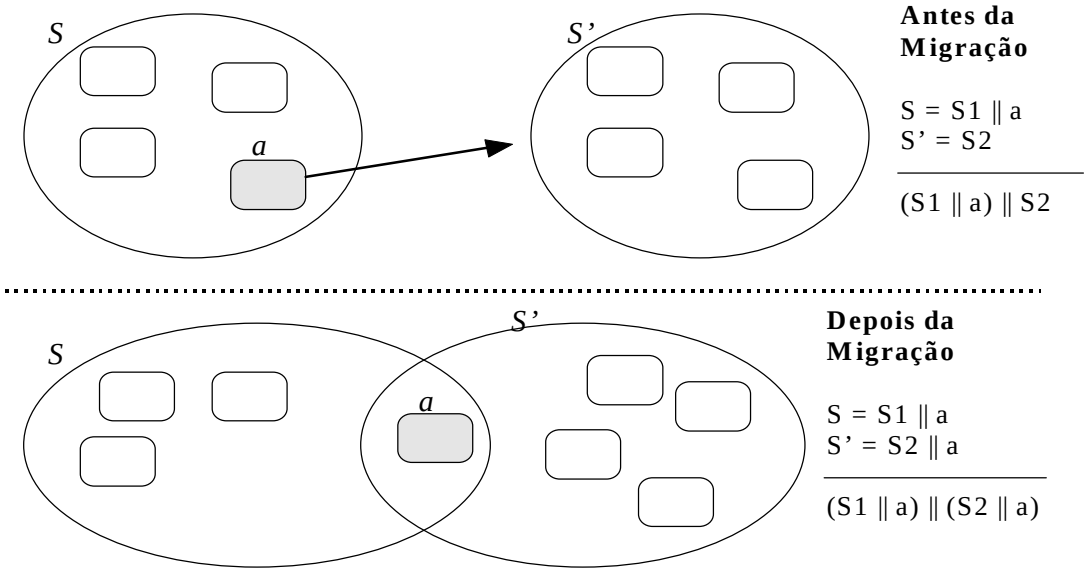

Figura 1.2 - Incorporação de um agente em outra sociedade

A capacidade de aceitar novos agentes e perder agentes também coloca novos problemas às sociedades. Tais problemas podem ser classificados em dois níveis, o nível social e o nível do agente. No nível social, o maior problema é manter a integridade funcional da sociedade depois da migração.

> *O problema principal que a sociedade deve resolver quando um agente entra ou sai dela é manter sua* ***integridade funcional****. Quer dizer, a sociedade deve assegurar-se que depois da migração do agente ela continue funcionando tão bem (ou melhor que) estava funcionando antes da migração. A sociedade da qual sai um agente deve ter capacidade para encontrar entre os agentes que permaneceram algum que possa tornar-se responsável pelas funções que o agente que saiu era responsável.*[1]

1 [COS94a, p. 537], tradução do autor.

Para o agente que está migrando, são identificados dois tipos de problemas adaptativos: de *linguagem e interação*, que concerne ao uso de expressões usadas e a maneira como as interações são organizadas na nova sociedade; e, problemas de *conhecimento e atuação*, que se referem a como um agente irá se comportar a fim de realizar justamente aquilo que a sociedade espera dele.

Este trabalho irá se ater aos problemas de conhecimento e atuação. Para que os agentes da sociedade possam cooperar e coordenar suas ações, é necessário que tenham conhecimento das capacidades, habilidades, desejos e planos dos outros agentes. Grande parte do conhecimento a respeito dos outros pode ser extraído dos papéis que estes podem assumir na sociedade. Assim sendo, o problema colocado é como os agentes da sociedade que recebeu o agente imigrante e o próprio agente imigrante conhecerão/aprenderão os papéis uns dos outros. Resolvendo este problema, os problemas de conhecimento e atuação na migração de agentes são também resolvidos.

1.3 Objetivo

O objetivo principal da dissertação é estudar os problemas gerados pela migração de agentes inteligentes e autonômos entre sociedades, em especial, os problemas acima classificados como de "conhecimento e atuação". A nível dos agentes, a fim de possibilitar cooperação na sociedade de entrada, foi buscado atribuir-lhes papéis que possam assumir. Três mecanismos de identificação de papéis foram estudados:

- por meio de um protocolo de apresentação;
- pelo reconhecimento do processo que o agente está executando; e
- pelo reconhecimento do plano do agente.

Estes mecanismos colocam a necessidade da elaboração de uma série de definições, por exemplo, a definição apropriada de agente com vistas aos mecanismos de identificação, a definição de um protocolo de apresentação, a definição de processos, a

definição de regras para inferência de processos e de planos a partir do comportamento do agente, etc. Além de descrever e estudar estes mecanismos, eles foram comparados avaliando as vantagens e desvantagens de cada um. A fim de validar, testar e comparar os mecanismos, foram realizadas experimentações numa sociedade, no caso, a sociedade de Produtores e Consumidores.

1.4 Trabalhos correlatos

Esta seção relata alguns trabalhos relacionados ao problema proposto. Os problemas envolvidos na migração de agentes estão colocados em [COS 94], que dá uma visão geral do escopo em que se encontra este trabalho. O artigo classifica estes problemas em dois tipos, problemas de linguagem e interação e problemas de conhecimento e atuação. Estes dois tipos de problemas estão relacionados aos aspectos de comunicação e de organização, respectivamente, sem os quais não se pode alcançar cooperação (cf. [WER 87]). A solução dos problemas de migração implica na solução dos problemas que ela causa na comunicação e na organização.

O problema de linguagem e interação nas sociedades abertas é tratado em [BOR 94], podendo ser considerado como uma etapa anterior à etapa abordada aqui. A questão principal está em que o agente imigrante não conhece as formas de comunicação (os protocolos) da sociedade onde está entrando. O objetivo do trabalho de [BOR 94] é criar mecanismos no agente para aprender a comunicar-se com os demais agentes da sociedade em que está ingressando. Para tanto, presupõe-se que os protocolos de comunicação estejam descritos na sociedade conforme certa linguagem de descrição de protocolos (LDP). O agente imigrante precisa, então, aprender esta LDP, os protocolos propriamente ditos, e como estes são utilizados. Para descrever a LDP é proposta uma meta-linguagem e um meta-protocolo para efetuar a comunicação nesta meta-linguagem.

O problema de conhecimento e atuação (organização em [WER 87]) colocado às sociedades abertas foi inicialmente tratado em [BER 92], que coloca a pergunta: quando da entrada de um agente, como permitir que os agentes se conheçam? No artigo, é proposto um protocolo de apresentação onde os agentes trocam informações sobre seus papéis e capacidades. A seção 2.3.1, que também propõe um protocolo de apresentação, descreve mais detalhadamente este artigo.

Problema semelhante ao apresentado neste trabalho também é apontado no contexto do projeto DARPA Knowledge Sharing Effort [FIN 92, MAY 95]. O problema principal neste projeto é proporcionar aos agentes formas de interação em um ambiente dinâmico; diante deste problema, são colocadas as seguintes questões:

- Em que linguagem os agentes irão formular suas requisições?
- Que protocolos serão utilizados para enviar e receber mensagens?
- Como um agente pode saber para quem enviar um de pedido de informação? Como encontrar este agente? Que protocolo utilizar para encontra-lo?

Para solucionar os problemas a nivel de linguagem, isto é, sobre o conteúdo das mensagens, é proposta um liguagem padrão de comunicação chamada KIF (Knowledge Interchange Format) que é basicamente uma extensão da lógica de primeira ordem. A solução para os problemas de comunicação esta baseada no KQML (Knowledge Query and Manipulation Languages), uma linguagem e protocolo de comunicação que pretende dar suporte a interoperabilidade entre agentes num ambiente distribuído. Como solução para o terceiro item acima, propõe um tipo de protocolo de apresentação. O principal problema destas soluções é o grande número de conhecimentos prévios que o agente deve possuir para interagir com os demais, dentre eles, pode-se citar:

- o agente tem que conhecer a linguagem utilizada (KIF) e estar sempre atualizado sobre novas versões desta;
- o agente deve conhecer as regras de comunicação (KQML); e

- o agente que envia uma mensagem tem que saber que termos da linguagem utilizar para garantir que o outro agente irá interpretar a expressão da mesma forma.

Além destes materiais que motivaram e iniciaram o assunto desta dissertação, outros foram utilizados e serão apresentados no decorrer do texto assim que for necessário.

1.5 Organização do texto

Este trabalho está organizado em dois capítulos principais. O capítulo 2 trata do problema proposto a nível do agente; para tanto, apresenta uma definição de agente baseada em processos e três mecanismos de identificação de papéis. Para o primeiro mecanismo, é especificado um protocolo de apresentação, para o segundo, são definidas regras para reconhecimento de processos a partir de comportamento do agente, e, para o terceiro, são definidas regras para reconhecimento do plano do agente e a intenção do agente com este plano. No final do capítulo, é feita uma comparação entre os três mecanismos considerando aspectos de eficácia, eficiência e limitações. O capítulo 3 relata experiências de implementação das propostas na sociedade Produtor - Intermediário - Consumidor, acrescentando ao capítulo 2 detalhes dos mecanismos bem como exemplos concretos. Nos anexos são colocados os códigos que implementaram a sociedade PIC e acompanhamentos de simulações realizadas para esta sociedade.

2 AGENTES E PAPÉIS EM SOCIEDADES

Este trabalho busca estudar a entrada e saída de agentes de uma sociedade aberta, mas para que isso possa ser feito, antes é preciso estabelecer algumas definições, sem as quais seria difícil e impreciso tratar do assunto. Dentre elas, a definição de agente é uma das mais importantes, uma vez que ele é um dos objetos de estudo. Outra definição importante, dependente da primeira e base para que o processo de entrada e saída mantenha a integridade da sociedade, é a noção de papel que um agente assume num certo momento e numa sociedade.

Para a definição de agente, serão vistos dois aspectos, um fixo e um dinâmico. O aspecto fixo do agente remete à sua estrutura e terá como base a noção de processo de Hoare [HOA 85] e de funcionamento de Rocha Costa [COS93b, COS94a]. O aspecto dinâmico é visto como uma parte do funcionamento que é refletida no comportamento do agente inserido e atuante numa sociedade. Essa parte do funcionamento, relacionada com a sociedade, é justamente o papel do agente. Identificar o papel de um agente é essencial para saber qual a sua participação na sociedade. Sabendo o papel, juntamente com uma especificação apropriada deste, pode-se verificar algumas propriedades do agente; fazer predições do seu comportamento; auxiliar na coordenação e cooperação da sociedade; verificar qual o impacto que o agente causa no funcionamento global da sociedade; etc.

2.1 Definição de agente

Um *agente* pode ser definido como uma entidade real ou virtual que emerge num ambiente onde pode agir intencionalmente, hábil a perceber e representar o ambiente, hábil a comunicar-se com outros agentes e autônomo [DEM 90, CAS 90, SIC 92]. Por agir intencionalmente, entende-se que o agente planeja suas ações em conformidade com suas crenças e desejos para satisfazer uma determinada intenção [BRA 84]. Por autonomia entende-se que o agente tem sua própria existência, que não é justificada pela existência de outros [DEM 90], e age conforme suas próprias crenças, conhecimentos e capacidades [CAS 90].

Conforme [COS93a], existem três formas de descrever um agente: pela sua estrutura, pelo seu funcionamento, e/ou pelo seu comportamento. Vê-se a seguinte relação entre estes três aspectos: a estrutura possibilita certos funcionamentos, dos quais o agente possui um; um funcionamento permite que o agente assuma um determinado conjunto de comportamentos, sendo o comportamento a parte externa (no sentido de observável) do funcionamento[2]. Obviamente, por composição, existe relação entre a estrutura e o comportamento. A definição de agente apresentada acima enfatiza os aspectos estruturais e comportamentais deste, deixando de abordar os aspectos funcionais que estão diretamente relacionados aos papéis. Como pretende-se utilizar uma definição de agente que corrobore a identificação de seu papel, opta-se por uma descrição de agente com base nos aspectos funcionais.

Para descrever o funcionamento do agente, se faz necessária a noção de processos, uma vez que este também descreve um conjunto de comportamentos. Descrever os comportamentos possíveis do agente, e não seus estados mentais, como é comum nos trabalhos onde existem modelos dos outros agentes com vista à cooperação, apresenta a vantagem de que não é necessário conhecer ou descobrir qual seu estado mental, uma vez que esta informação nem sempre é acessível ou mesmo confiável. Um

outro problema em descrever um agente pelos seus estados mentais (crenças, desejos, intenções, planos…) é que estes são muito voláteis, ou seja, se alteram com muita freqüência, o que não acontece com o funcionamento do agente e o conjunto de papéis que este determina, onde existe maior constância no decorrer do tempo.

2.1.1 Processos

Para descrever o processo que rege o comportamento do agente, será utilizado o formalismo de especificação de processos definido por Hoare [HOA 85] para descrever comportamentos de objetos. A base para a especificação de processo está nos eventos de que determinado objeto participa. Na verdade, um *evento*, como utilizado aqui, denota uma classe de eventos, ou seja, podem existir várias ocorrências de um evento no decorrer do tempo. Por exemplo, o evento "letra H" pode ter várias ocorrências num texto. O conjunto de eventos considerados relevantes para descrever o comportamento de um objeto, ou mais especificamente, de um agente, é chamado *alfabeto*. Um *processo* pode ser descrito em termos de um conjunto de eventos pertencentes ao alfabeto do objeto. A seguir, serão apresentadas algumas notações iniciais para especificação de processos.

2.1.1.1 Prefix

Tomando *x* como um evento e **P** como um processo. Então [3]

$$\mathbf{Q} = (x \rightarrow \mathbf{P})$$

descreve um processo **Q** que primeiro participa no evento *x* e então se comporta conforme o processo **P**. O alfabeto de **Q**, denotado $\alpha(\mathbf{Q})$, é igual ao de **P**, ou seja, *x* está no alfabeto de **P**.

---

2 Ver [COS93d, p. 27ss] para uma análise destes três aspectos.

3 Serão utilizadas as seguinte convenções:
letras minúsculas em *itálico*: eventos;

$\alpha(x \rightarrow \mathbf{P}) = \alpha\mathbf{P}$

Existe um tipo especial de processo que marca o fim de uma execução, denotado por **STOP**.

Exemplo 2.1: sendo um agente do tipo Consumidor, com o alfabeto {*pegar_peça*, *consumir*}, e que consome uma única peça, então tem-se o seguinte processo:

P = (*pegar_peça* → *consumir* → **STOP**).
$\alpha\mathbf{P}$ = {*pegar_peça*, *consumir*}

se o mesmo agente consumisse duas peças, o processo seria

P = (*pegar_peça* → *consumir* → *pegar_peça* → *consumir* → **STOP**).
$\alpha\mathbf{P}$ = {*pegar_peça*, *consumir*}

2.1.1.2 Recursão

Prefix somente pode descrever processos que param. Processos que realizam coisas repetidamente podem ser descritos de maneira mais elegante utilizando recursão.

Exemplo 2.2: tomando o processo

(*pegar_peça* → *consumir* → **C**), onde
C = (*pegar_peça* → *consumir* → **C**)

tem-se a descrição de um objeto que infinitamente pega peças e as consome, ou seja, o processo realizado por um consumidor real.

Exemplo 2.3: um consumidor que não tem acesso direto às peças, e tivesse que pedi-las a outro, poderia ser descrito da seguinte forma:

---

letras maiúsculas em **NEGRITO**: processos.

$\mathbf{C} = (pedir\_peça \rightarrow receber\_peça \rightarrow consumir \rightarrow \mathbf{C})$.
$\alpha\mathbf{C} = \{pedir\_peça, receber\_peça, consumir\}$

2.1.1.3 Escolha

Um objeto pode se comportar de formas diferentes, dependendo das circunstâncias em que se encontra. Esse comportamento alternativo é descrito pela notação

$$(x \rightarrow \mathbf{P} \mid y \rightarrow \mathbf{Q}),\ x \neq y$$

onde o objeto pode participar de um dos dois eventos (x ou y) e, depois disso, se comportar conforme **P**, se o primeiro evento foi x ou conforme **Q** se foi y. A "escolha" deve manter as seguintes propriedades:

$\alpha(x \rightarrow \mathbf{P} \mid y \rightarrow \mathbf{Q}) = \alpha\mathbf{P}$
$\{x, y\} \subseteq \alpha\mathbf{P}$
$\alpha\mathbf{P} = \alpha\mathbf{Q}$ [4]

Exemplo 2.4: no caso do exemplo 2.3, o consumidor poderia não aceitar a peça e devolvê-la.

$\mathbf{C} = (pedir\_peça \rightarrow receber\_peça \rightarrow (devolver \rightarrow \mathbf{C} \mid consumir \rightarrow \mathbf{C}))$.
$\alpha\mathbf{C} = \{pedir\_peça, receber\_peça, devolver, consumir\}$

Que também é representado na figura 2.1.

4 Isto porque os dois processos, para poderem funcionar juntos, devem possuir o mesmo alfabeto.

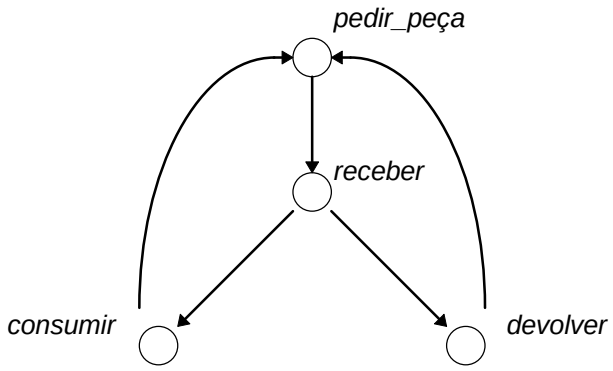

Figura 2.1 - Exemplo de um processo Consumir

2.1.1.4 Seqüência

Dois processos podem ser unidos, de modo que, quando um acaba, o outro inicia. Para isso, é utilizada a notação

P ; **Q.**

Normalmente um processo termina executando **STOP**, na caso da seqüência o primeiro processo não pode parar o funcionamente, mas deve passa-lo para o segundo processo. Assim, uma condição para utilização deste operador é que o último evento do processo **P** seja **SKIP**.

Como exemplo, o processo do exemplo 2.1 (que consome duas peças) poderia ser descrito pelo processo **P2**,

P = (*pegar_peça* → *consumir* → **SKIP**)
P2 = **P** ; **P** ; **STOP** .

2.1.2 Especificação de agentes

Um agente não pode ser descrito simplesmente por um processo, porque, conforme a definição vista acima, um agente tem um estado mental (aspectos internos e não comportamentais, logo, não abarcados por processos). Mesmo o aspecto funcio-

nal não pode ser restrito a uma descrição por um único processo, considerando que um agente que possa assumir simultaneamente mais de um papel na sociedade e, naturalmente, mais de um processo. Assim, o aspecto funcional de um agente fica descrito como um conjunto de processos. Outro motivo para adotar esta descrição de funcionamento é que o interesse está em descrever uma parte do funcionamento do agente que representa um papel, ou seja, descrever um processo do conjunto de processos (cf. figura 2.3).

Assim, construímos um agente conforme a definição de *dinâmica* apresentada por [COS94a], onde o agente é formado por um conjunto de processos — seu funcionamento — mais informações do agente (identificação, estrutura, etc) que o identificam como único. Assim, o aspecto fixo do agente fica definido como um par

agente = (Funcionamento, Informações)

Temos assim, a estrutura de um agente mostrada na figura 2.2.

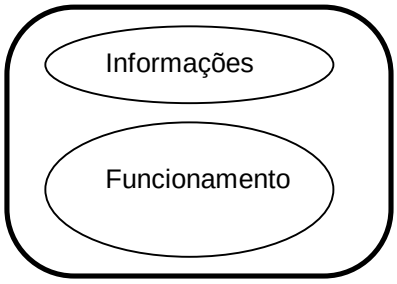

Figura 2.2 - Estrutura do agente

Na literatura em geral (cf., por exemplo, [SIC 94, GAS 88, WOO 94]), é dada especial atenção ao que, na definição de agente apresentada acima, chamou-se Informações, em especial, aos estados mentais e à estrutura/arquitetura. Por exemplo, o agente, em relação ao seu comportamento, é classificado basicamente num dos dois grupos seguintes [SIC 92]:

- *Reativos*, possuem representação implícita, não têm história e o controle não é deliberativo. Estão baseados em modelos de organizações biológicas

(como as formigas), onde cada elemento em si não possui inteligência, mas a coletividade sim.

- *Cognitivos*, possuem representação explícita, têm história, o controle é deliberativo e existe organização social.

Como pode ser visto, esta classificação é feita considerando os atributos dos estados mentais dos agentes. Também com base nos estados mentais (no caso, capacidades, objetivos e planos) são feitos modelos dos outros agentes com objetivo à cooperação.

Agentes cognitivos conseguem, entre outras coisas, planejar suas ações futuras criando planos, que tornam-se mais um elemento dos estados mentais do agente. Estes agentes podem ser vistos como aqueles onde os processos são criados por eles próprios, num processo[5] de raciocínio ou planejamento [POL 92]. O planejamento busca alcançar uma intenção levando em consideração as crenças e capacidades atuais do agente. Do ponto de vista externo, não há diferença entre um processo pré-definido na estrutura do agente e processos construídos.

2.2 Papéis de agentes

Para que os agentes de uma sociedade possam realizar suas tarefas, em especial, aquelas em que a participação de outros agentes é necessária[6], eles precisam saber quais os papéis que os demais agentes dessa sociedade podem assumir. Por exemplo, se um agente precisa de determinada informação, tem que pedi-la a um agente que pode respondê-la, ou seja, que tenha um papel de "informante". É perda de tempo pedir algo a um agente que não pode atender ao pedido.

---

5 Estes agentes têm, no mínimo, este processo de planejamento na sua estrutura.

6 Onde busca-se alcançar um objetivo social. Um objetivo social é aquele que não pode ser atingido por um único agente, mas somente por vários agentes cooperando, cf. [WER 87, p. 3].

Para conseguir tal cooperação, muitos autores (por exemplo, [SIC 94] [GAS 88]) dão ao agente a capacidade de descobrir os estados mentais do outro agente, ou seja, quais suas metas (para eventualmente conciliá-las com as suas), capacidades (para poder contratar algum serviço), planos (para poder atendê-lo melhor [ALL 80]), etc. Alguns desses estados mentais podem ser inferidos se temos o papel do agente. Por exemplo, do papel (visto como uma função do agente na sociedade) pode-se inferir qual o plano do agente. A partir do plano, pode-se saber quais são os objetivos (estado final do plano) bem como algumas das capacidades do agente (aquelas que utiliza na realização do plano). Além das funções já apresentadas, os papéis dos agentes também são utilizados por [WER 87] para construir a estrutura organizacional da sociedade, onde, para que haja cooperação, são necessárias duas coisas: comunicação e organização.

A figura 2.3 mostra como o autor vê o conceito de papéis. Um agente tem processos internos e participa de processos globais. *Processos internos* são aqueles onde não existe interação com outros processos de outros agentes. Os *processos globais* são da sociedade como um todo, não "pertencem" a nenhum agente específico. Processos globais são formados por eventos globais, um evento é caracterizado como global se envolve mais de um agente, logo, corresponde a uma interação. O conjunto dos eventos globais onde o agente participa define o seu comportamento. A participação do agente num processo global é feita por um processo que mistura eventos internos e eventos globais, este tipo de processo é chamado *processo de interação* (PI). Um PI define um comportamento parcial do agente, ou seja, o comportamento do agente em relação a um processo global.

O *papel de um agente* sempre é relacionado a outro agente ou grupo de agentes da sociedade [COS94a]. Esta relação fica representada através do processo de interação do agente. Por esse motivo, o aspecto dinâmico de um agente é constituído dos papéis que assume, já que diz respeito ao agente num determinado instante na so-

ciedade e com um determinado comportamento. Em outros momentos, o papel pode ser outro, embora os processos (aspecto fixo) sejam sempre os mesmos.

Definindo papel da forma apresentada acima, como um processo de interação do agente, podemos utilizar a mesma notação de processos para descrever os papéis de um agente.

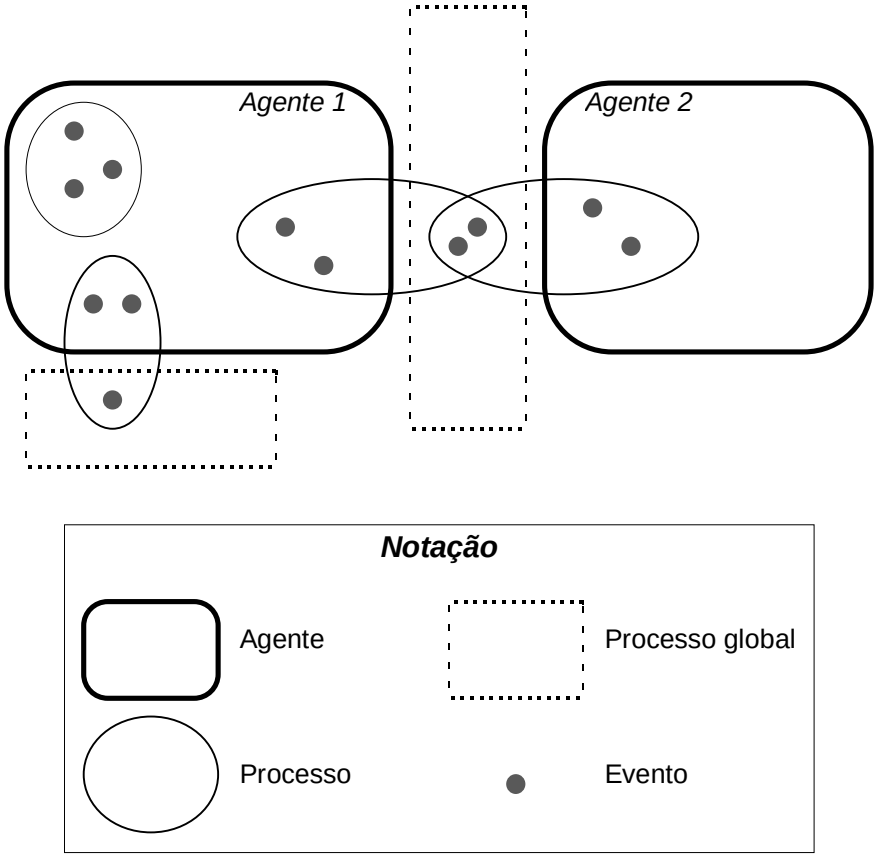

Figura 2.3 - Relação entre estrutura e papéis do agente

2.3 Identificação de papéis

Em sociedades não abertas, onde o número de agentes é fixo, cada agente pode saber previamente quais os papéis dos outros agentes. Obviamente, em socieda-

des abertas, isto não pode acontecer. Quando um novo agente entra nesta sociedade, é preciso que os demais agentes identifiquem o seu papel e que ele próprio identifique os papéis dos agentes já inseridos nesta sociedade.

Em [BER 92], o problema da entrada de novos agentes em uma sociedade é tratado, dando-se especial atenção à identificação dos papéis destes agentes com o objetivo de integrá-los à sociedade, buscando

- usar o mínimo de fluxo de comunicação;
- que os agentes adquiram dinamicamente conhecimento dos outros sem interromper o funcionamento do sistema;
- que os agentes possam aumentar seu conhecimento a respeito dos outros de tal modo que a interação seja mais frutífera e útil; e
- aproximar ao máximo a intenção da comunicação com o conteúdo dela, evitando os problemas de intencionalidade.

A identificação do papel de um outro agente pode ser feita de duas formas, (i) diretamente, através de uma requisição explícita aos agentes; ou (ii) indiretamente, inferindo o papel a partir do seu comportamento. Poderia-se ainda propor um terceiro enfoque que abarcasse ambos, por exemplo, verificando se o comportamento observado confere com o esperado (obtido de forma direta por apresentação); ou ainda, se houvesse dúvidas quanto ao papel inferido por observação, poder-se-ia entrar num protocolo de apresentação.

Berthet, Demazeau e Boissier [BER 92] utilizam a primeira forma de identificação. O agente interessado em conhecer um outro envia-lhe uma mensagem de apresentação onde constam os papéis (Ro) que pode assumir e as ações básicas (Ba) que pode executar. Depois desta primeira etapa de apresentação, as seguintes situações de conflito podem ocorrer (considera-se um agente P se apresentando a um agente N):

- RoP = RoN e BaP = BaN $\rightarrow$ os agentes aparecem um ao outro como *clones*; eles continuam a apresentação para verificar se são realmente clones;

- RoP = RoN e BaP ≠ BaN → os agentes aparecem em competição; logo, um dos agentes pode ser melhor que o outro já que pode assumir o mesmo papel, mas com ações básicas diferentes;
- RoP ≠ RoN e BaP = BaN → aparece uma incoerência entre as capacidades dos agentes; ou um dos agentes pode descobrir que pode assumir outros papéis que antes não sabia;
- RoP ≠ RoN e BaP ≠ BaN → os agentes não apresentam nada em comum, podem continuar a apresentação para se conhecerem melhor.

A continuação do protocolo de apresentação, no caso dos agentes desejarem se conhecer ainda melhor, é feita numa segunda etapa com troca de outras informações, que são: como as Ba do agente podem ser usadas, isto é, condições para os parâmetros das ações; e exemplos de como as Ba são instanciadas.

No presente estudo de caso, foram analisados os dois enfoques (direto e indireto) para descobrir o papel de um agente, procurando alcançar os mesmos objetivos de [BER 92], embora os protocolos de apresentação e a linguagem sejam diferentes. A forma de descoberta direta é desenvolvida de forma semelhante (indagação explícita por mecanismo de apresentação, seção 2.3.1), embora não tenham sido considerados os problemas de resolução de conflitos pós-apresentação tratados por [BER 92]. A forma de descoberta indireta foi enfocada de duas maneiras: primeiro por comparação e classificação do comportamento observado com descrições prévias de papéis (seção 2.3.2) e, segundo, por indução do plano do agente seguida da associação deste plano a um papel (seção 2.3.3).

Em termos de implementação, podem-se caracterizar duas situações na identificação do papel durante a entrada de um agente. O agente que está entrando identifica os papéis dos agentes da sociedade (i), provavelmente, num caso onde o agente procura atingir algum objetivo que não era possível na sua sociedade de origem. E os agentes da sociedade identificam o papel do agente que está entrando (ii), caso a sociedade precise de novas funções para atingir seus objetivos. O agente na situação (i) pode utilizar tanto a forma direta como a indireta para identificar os papéis. Já os agentes na situação (ii) somente podem utilizar os métodos de identificação diretos,

uma vez que não há comportamento a ser observado no novo agente. É claro que a sociedade poderia experimentar interações com este novo agente a fim de observar-lhe o comportamento, mas sem conhecimento prévio do agente é difícil saber o que experimentar.

A seguir são apresentados, em maiores detalhes, os métodos de identificação de papel propostos (cf. [COS 94]), sendo que exemplos do uso desses métodos podem ser vistos no capítulo 3.

2.3.1 Identificação de papéis por mecanismo de apresentação

No mecanismo por apresentação, o agente interessado em saber o papel de um outro lhe faz uma pergunta. A apresentação exige dos agentes envolvidos

i) conhecimento do mesmo protocolo de apresentação;
ii) que tenham descrições comuns para papéis; e
iii) que tenham conhecimento dos seus próprios papéis.

Em sociedades não abertas, estas exigências são facilmente satisfeitas, por esta razão, o mecanismo de apresentação é o mais utilizado neste tipo de sociedades. Em sociedades abertas, a segunda exigência é difícil de ser satisfeita, principalmente quando considerada a migração de agentes entre sociedades com histórias diferentes e, portanto, com boa chance de terem identificações diferentes para papéis semelhantes. Por exemplo, em uma sociedade o papel de consumidor pode ser identificado por “Consumidor” e em outra por “Receptor”, no caso, ambas identificam papéis por palavras, o que em si é também uma restrição. Para contornar esta exigência, primeiro, os agentes devem ter consciência de que podem ter descrições diferentes; segundo, devem adotar uma representação comum para os papéis, ou um deles deve aprender a forma de descrição do outro. Estas soluções não serão abordadas no presente trabalho por exigirem protocolos de negociação referentes aos aspectos de linguagem e interação (cf. [BOR 94]).

A terceira exigência pressupõe que o agente tenha um papel pré-determinado, independente da função que está tendo num dado momento (o que vai contra a definição de papel apresentada), ou que ele tenha capacidade de avaliar constantemente qual o papel que está assumindo.

2.3.1.1 Linguagem de descrição de protocolos

A fim de que os agentes possam saber quais regras regem a "conversa" de apresentação, a definição de um protocolo de apresentação torna-se necessária. Genericamente, um protocolo é identificado por um nome, um estado inicial e uma seqüência de transições de estados. Diante disso, o autor propõe uma linguagem de descrição de protocolos (LDP) com a seguinte sintaxe[7]:

```
<protocolo>  ::= protocol <id>;       {id do protocolo}
                 <dcl_agentes>
                 <estado_pro>         {estado  inicial}
                 <transições>.

<id>         ::= {definição tradicional de identificador}

<dcl_agentes>::= <dcl_agente> <dcl_agentes> |
                 <dcl_agente>

<dcl_agente> ::= <agente> : <tipo_ag> ;

<tipo_ag>    ::= <id>

<transições> ::= <transição> <transições> |
                 <transição>

<transição>  ::= <mensagem> → <estado_pro>

<estado_pro> ::= (<estado_agente>,<estado_agente>) |

                 (<estado_agente>,<estado_agente>) |
                 (<estado_agente>,<estado_agente>)

<mensagem>   ::= <tipo_mem>(<conteúdo>)

<tipo_mem>   ::= request | inform | reply

<conteúdo>   ::= {definido conforme um termo de Prolog}

<estado_agente> ::= <agente> <estados> |
```

7 O conteúdo entre {} são comentários da descrição, os *tokens* em negrito são terminais e os *tokens* entre <> são não terminais.

```
                    <agente>[8]
<agente>        ::= {definido conforme uma variável de Prolog}
<estados>       ::= . <id> <estados> |
                    . <id>
```

A *declaração dos agentes*, `<dcl_agentes>`, determina quais os papéis que cada agente irá assumir no decorrer do protocolo. Se o papel do agente não for essencial, o tipo pode ser "Agente".

Um *estado do protocolo*, `<estado_pro>`, é um par, onde cada parte representa o estado de um dos agentes envolvidos na comunicação. O estado do protocolo pode ter uma das seguintes instâncias: quando o primeiro agente do par toma a iniciativa na próxima transição (caso em que o primeiro argumento está sublinhado); quando o segundo agente toma a iniciativa (segundo argumento é sublinhado); ou quando não há próxima transição (nenhum dos argumentos é sublinhado), isto só acontece no estado final do protocolo.

Um *estado de agente*, `<estado_agente>`, é descrito por uma variável, representando o agente, e por uma seqüência de estados (separados por ponto), indicando os estados pelos quais o agente passou antes da próxima transição.

Transições ocorrem devido a mensagens que alteram o estado atual do protocolo. Cada mensagem tem um tipo e um conteúdo. Como em [GAS91a], os tipos de mensagem podem ser *request* (requer que o receptor da mensagem envie uma resposta à solicitação), *reply* (a mensagem de resposta a uma solicitação) e *Inform* (mensagem que informa algo a outro agente sem necessitar de resposta).[9]

Por exemplo, a transição

$$(\underline{X.i}, Y.q)\ \text{request}(\text{Algo}) \rightarrow (X.j, Y.r)$$

---

8 Para o caso de ser irrelevante indicar o estado do agente.

9 O tipo de mensagem *present* proposto por [GAS91b] foi substituído por um protocolo de apresentação nesta dissertação.

indica que o agente X, no estado *i*, enviou a mensagem “request(Algo)” ao agente Y, que estava no estado *q*, passando assim a estarem nos estados *j* e *r*, respectivamente.

2.3.1.2 O protocolo de apresentação

Definida a linguagem de descrição de protocolos, podemos definir o protocolo que controla a apresentação, como segue:

```
protocol presentation;
     X:Agent;
     Y:Agent;
        (X.init,Y.wait) request(role) →
        (X.wait,Y.send) reply (role(R)) →
        (X,Y).
```

Se um agente deseja saber o papel de outro, assume o comportamento X. Se lhe é solicitado o seu papel assume o comportamento Y.

2.3.2 Identificação de papéis por observação e classificação

Este mecanismo se dá da seguinte maneira: um agente observador observa uma seqüência de eventos globais em que o agente observado está participando, montando um *trace*, que é uma representação do comportamento do agente observado em relação a um outro agente. Identificado o *trace*, têm-se duas alternativas para chegar ao papel do agente.

i) A partir do *trace*, construir um processo que o descreve e comparar o processo obtido com processos previamente definidos que caracterizam papéis (por exemplo, o processo que define Consumidor no exemplo 2.3).

ii) A partir dos processos previamente definidos, verificar em qual deles o *trace* é possível.

Estas duas alternativas estão representadas na figura 2.4, onde, em (i) a relação entre o *trace* e o processo é de construção e em (ii) é de pertinência.

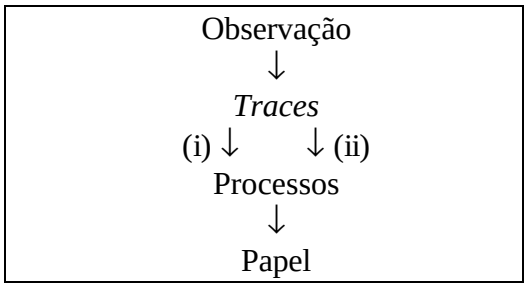

Figura 2.4 - Representação das duas formas de identificação do papel a partir do *trace*.

Estas soluções não garantem que os papéis identificados sejam os reais, pelos seguintes motivos:

- o agente observador pode não ter observado suficientemente o agente alvo, e assim, o agente observado ainda pode apresentar um comportamento não observado anteriormente: o *problema da parada*;
- o agente observador não tem o papel observado no conjunto de papéis pré-definidos (este motivo tem maior relevância quando se trata do caso (ii) apresentado acima); e
- o agente observador não tem capacidade de descrever o papel do agente observado (este motivo somente tem relevância quando se trata do caso (i) apresentado acima).

2.3.2.1 *Traces* de processos

"Um *trace* de um comportamento de um processo é uma seqüência finita de símbolos representando os eventos em que o processo tem participado até certo momento." [HOA 85, p. 41] Um *trace* é denotado por uma seqüência de símbolos, separados por vírgulas, colocados entre os sinais de maior e menor. Exemplos:

<*x*, *y*>	*trace* de dois eventos, *x* seguido de *y*.
<*x*>	*trace* contendo apenas um evento.
<>	*trace* sem eventos.

Exemplo 2.5: sendo o processo definido no exemplo 2.1, os seguintes *traces* seriam possíveis:

<>	quando ainda não fez nada
<*pegar_peça*>	quando pegou a peça
<*pegar_peça*, *consumir*>	quando está consumindo a peça

Exemplo 2.6: tomando o processo do exemplo 2.2, temos um número infinito de *traces* possíveis (visto que o processo é recursivo sem parada), por exemplo,

<>,
<*pegar_peça*>,
<*pegar_peça*, *consumir*>,
<*pegar_peça*, *consumir*, *pegar_peça*>,
<*pegar_peça*, *consumir*, *pegar_peça*, *consumir*>,
…

A seguir, são apresentadas as operações que podem ser realizadas sobre *traces* (as letras *s*, *t* e *u*, denotam *traces*).

Concatenação (^): junta dois *traces*, colocando um atrás do outro. Por exemplo: <*x*, *y*> ^ <*z*, *x*> = <*x*, *y*, *z*, *x*>.

Tamanho (#): número de eventos do *trace*, exemplos: #<> = 0, #<*x*, *y*, *z*, *x*> = 4.

Subscrição (π): retorna o i-ésimo evento do *trace s*, para $0 \leq i < \#s$, exemplos: π_0<*x*, *y*, *z*, *x*> = *x*; π_1<*x*, *y*, *z*, *x*> = *y*.

Cabeça ($_{\mathbf{0}}$): retorna o primeiro evento de um *trace*, por exemplo: <*x*, *y*, *z*>$_0$ = *x*.

Cauda ('): retorna o *trace* sem o primeiro evento, por exemplo: <*x*, *y*, *z*>' = <*y*, *z*>.

Além destes operadores, já definidos por [HOA 85], acrescenta-se o seguinte:

Subtrace(*t, i, j*): retorna uma parte do *trace t*, da posição *i* até a posição *j* ($0 \leq i \leq j < \#t$), exemplo: subtrace(<*x, y, z, x*>, 0, 2) = <*x, y, z*>. Este operador tem a seguinte definição:

subtrace(*t, i, i*) = $< \pi_i$ t >.
subtrace(*t, i, j*) = $< \pi_i$ t > ^ subtrace(*t, i*+1, *j*).

2.3.2.2 Construção de processos a partir de *traces*

Tendo-se um *trace* é possível inferir o processo que o originou. O problema é encontrar o melhor processo que descreve o *trace*, uma vez que um mesmo *trace* é permitido por vários processos. Por exemplo, o *trace*:

<*x, y, x, y, x, y*>

pode ter sido gerado por qualquer um três processos seguintes:

P1 = ($x \rightarrow y \rightarrow x \rightarrow y \rightarrow x \rightarrow y \rightarrow$ **STOP**).
P2 = ($x \rightarrow y \rightarrow x \rightarrow$ **P2**).
P3 = ($x \rightarrow y \rightarrow$ **P3**). { provavelmente a melhor opção }

Um problema semelhante a este é tratado em especificação de programas. No caso, procura-se chegar ao programa a partir de exemplos de sua execução. Em [KOS 94], o objetivo é modelar objetos, para isso, analisa o comportamento do objeto e cria a partir dele um diagrama de transição de estados (DTE). Por exemplo, considerando o processo

P = $a \rightarrow b \rightarrow a \rightarrow$ **P**,

que determina um conjunto de comportamentos (*traces*) igual ao da expressão regular $(aba)^*$, para o *trace* <*a,b,a,a,b,a,a,b,a*>, o algoritmo de [KOS 94][10] irá constrir o DTE da figura 2.5 pelos seguintes passos:

---

10 Ao contrario do algoritmo original proposto por [KOS 94], neste exemplo não são consideradas informações sobre condições para a transição de um envento para outro. Esta condição tem que ser colocada porque o *trace* que obtido não inclui informações sobre o estado interno do agente, ou seja, as condições para passar de um evento para outro.

i) com o primeiro evento *a* é criado um novo estado, de nome *a*, isto é, para cada evento é criado um estado no DTE[11];

ii) com o segundo evento *b* é criado um novo estado e uma transição do último estado (*a*) para o estado atual (*b*), como não estão sendo consideradas as condições que levam de um estado a outro, as transições são incondicionais;

iii) para o terceiro evento (*a*) é criada uma transição do estado atual (*b*) para o estado *a*; como já existe um estado para este evento, não é criado um novo estado *a*;

iv) com o quarto evento (*a*), novamente já existe um estado para este evento, cria-se uma transição do último estado (*a*) para o estado atual (*a*); os eventos restantes do *trace* não irão alterar o DTE.

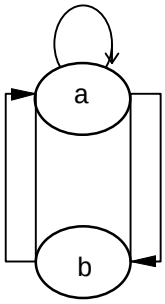

Figura 2.5 - DTE criado para o *trace <a,b,a,a,b,a,a,b,a>*

Deve-se observar que o algoritmo de [KOS 94] gerou um DTE que corresponde à expressão regular $a^*(ba)^*$, que não é igual à do processo que originou o *trace*. Este algoritmo difere basicamente em dois aspetos do algoritmo proposto neste trabalho. Primeiro, o algoritmo de [KOS 94] presupõe que o *trace* utilizado contém também informação sobre em que condições pode-se passar de um evento para outro; com estas informações provavelmente o problema do exemplo acima não ocorreria. Além disso, conhecendo as condições das transições pode-se criar facilmente a operação de "escolha" no DTE gerado. Segundo, o algoritmo de [KOS 94] prioriza a identificação de ciclos mais internos visto que quando surge um evento que já está no diagrama é

11 Se fossem consideradas as condições para mudança de evento, a utilização de um estado que já existe para um novo evento não deveria provocar indeterminismos.

feito um ciclo do estado atual para o estado do evento (desde que isso não crie indeterminismos), ou seja, não são necessárias duas sequências iguais para formar um ciclo.

A seguir é apresentado um mecanismo para identificação do processo que gerou determinado *trace*. São identificados três operadores de processos: prefix, recursão e escolha.

A. Identificação de prefix

É o caso mais simples, o processo é a própria seqüência de eventos do *trace*. Por exemplo, para o último *trace* do exemplo 2.5, o processo seria

$$\mathbf{P} = (pegar\_peça \rightarrow consumir \rightarrow \mathbf{STOP}).$$

Para ir do *trace* ao processo, é proposto um sistema de regras de reescrita, por meio de uma função de reescrita $\Rightarrow$: Traces $\rightarrow$ Processos. A transformação mais simples, que transforma a seqüência de eventos do *trace* num processo sem ciclos, é definida nas regras

r1. $\langle x \rangle \wedge s \Rightarrow (x \rightarrow \mathbf{P})$ tal que $s \Rightarrow \mathbf{P}$
r2. $\langle\rangle \Rightarrow \mathbf{STOP}$

onde a regra r1 diz que um *trace* $\langle x \rangle$ com cauda *s* é reescrito para um processo prefix com evento *x* e processo **P**, sendo **P** o processo gerado pelas regras de reescrita para a cauda *s*. A regra r2 diz que um *trace* vazio gera um processo **STOP**.

Continuando o exemplo acima, para o *trace* < *pegar_peça*, *consumir*> tem-se as seguintes transformações:

	<*pegar_peça*, *consumir*>	
$\Rightarrow$	(*pegar_peça* $\rightarrow$ <*consumir*>)	(por r1)
$\Rightarrow$	(*pegar_peça* $\rightarrow$ *consumir* $\rightarrow$ <>)	(por r1)
$\Rightarrow$	(*pegar_peça* $\rightarrow$ *consumir* $\rightarrow$ **STOP**)	(por r2)

B. Identificação de recursão

Processos muito simples (com pequeno número de eventos) podem ter *traces* muito extensos, é o caso de processos recursivos. Nestes casos, a técnica acima (identificação de prefix) não é eficiente. Considerando que os agentes sempre têm um comportamento cíclico, e portando a descrição de seus papéis incluem recursão, a identificação destes ciclos é essencial para a comparação do processo observado com o previamente descrito.

Como as regras r1 e r2 dão conta somente de processos sem ciclos, a regra seguinte reescreve processos sem ciclos em processos com ciclos.

r3. $s \Rightarrow \mathbf{I} ; \mathbf{C}$. Tal que
$\mathbf{C} = (\mathbf{P_1} \mid \mathbf{P_2} \mid \mathbf{P_3} \ldots) ; \mathbf{C}$.

Caso as seguintes condições sejam satisfeitas:

i) π_2 (idCiclos(s)) $\neq \{\}$ {deve existir pelo menos um ciclo}
ii) π_1 (idCiclos(s)) $\Rightarrow$ **I**[**STOP**/**SKIP**][12]
iii) $\forall\ t_n \in \pi_2$ (idCiclos(s)), $t_n \Rightarrow \mathbf{P_n}$[**STOP**/**SKIP**]

Onde **I** é um processo inicial, executado apenas uma vez e no início do *trace*, seguido pela execução de um ciclo **C**. Como um agente pode executar vários ciclos, o ciclo **C**, na verdade, é uma alternativa entre vários ciclos $\mathbf{P_n}$. A identificação de **I** e $\mathbf{P_n}$ é feita através da função idCiclos, que tem o tipo

idCiclos: Traces $\rightarrow$ Traces x P(Traces)

onde o argumento é o *trace s* do qual se pretende identificar ciclos e o resultado é um par formado pelo *trace* que inicia *s* (obtido por π_1)[13] e um conjunto de ciclos identificados em *s* (obtido por π_2). Tendo identificado estas partes de *s*, é utilizada a função $\Rightarrow$ para transformá-las em processos seqüenciais.

---

12 A notação [**STOP**/**SKIP**] substitui, no processo que a precede, **STOP** por **SKIP**.

13 A função π_n é a projeção do enésimo argumento de uma tupla.

Exemplo 2.7: o trace *s* = <*apresenta, pegar_peça, consumir, pegar_peça, consumir, pegar_peça, consumir, pegar_peça*>, tem um ciclo, <*pegar_peça, consumir* > e o início <*apresenta*>. Pela aplicação de ⇒, tem-se os passos seguintes:

Utilizando r3, da condição (iii), temos que
π_2 (idCiclos(*s*)) = {<*pegar_peça, consumir*> }
t_1 = <*pegar_peça, consumir*>

Por r1 e r2,
<*pegar_peça, consumir*> ⇒ (*pegar_peça* → *consumir* → **STOP**)

Com a substituição de r3(iii)
$\mathbf{P}_1$ = (*pegar_peça* → *consumir* → **STOP**)[**STOP**/**SKIP**]
= (*pegar_peça* → *consumir* → **SKIP**)

Aplicando r3(ii), temos
I = (*apresenta* → **SKIP**).

E, por fim, utilizando r3, temos o processo

I ; **C**.
C = P $_1$; **C**.

que descreve um Consumidor que se apresenta (processo **I**) e fica consumindo peças (no processo $\mathbf{P_1}$). A figura 2.6 também representa este Consumidor.

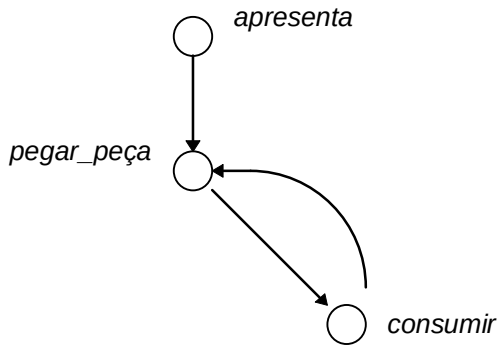

Figura 2.6 - Processo reconhecido a partir do trace <*apresenta, pegar_peça, consumir, pegar_peça, consumir, pegar_peça, consumir, pegar_peça* >

A implementação da função idCiclo é feita através do seguinte algoritmo:

idCiclos(*s*):

i) se $s = <>$
retorna $(<>, \{\})$

ii) se $\exists$ n (n variando de (#s/2-1) até 1 e
$\exists$ m (m variando de n+1 até #s-n)
tal que subTrace(*s*, 0, n) = subTrace(s, m, m+n)
então $c = c_a$ = subTrace(s, 0, n)
s_a = apaga(*s*, c)[14]
se $(|\pi_2(\text{idCiclos}(c_a))| = 1)$ {tem 1 sub-ciclo em c_a}
então $c = \pi_2(\text{idCiclos}(c_a))$
retorna $(<>, \{c\} \cup \pi_2(\text{idCiclos}(s_a)))$

iii) senão $c = \pi_2(\text{idCiclos}(s'))$
$i = s_0 \wedge \pi_1(\text{idCiclos}(s'))$
retorna (i, c)

A função idCiclos, como vista acima, possui três casos. No caso (i), o *trace* é vazio e, conseqüentemente, não existe início nem ciclos. No caso (ii), a função pega uma parte do início do *trace* e procura outra seqüência igual; se existe, tem-se um ciclo, que é justamente esta parte do *trace* que se repete. Como o ciclo identificado pode ter ciclos menores internamente, ele passa novamente pela função idCiclos. Uma vez que o ciclo foi identificado no início do *trace*, idCiclos retorna um *trace* vazio no primeiro argumento. A função continua procurando ciclos na parte restante do *trace*, sem o ciclo identificado. No caso (iii), não existe ciclo no início do *trace*, coloca-se o primeiro evento como parte inicial e procura-se ciclos na cauda do *trace*.

Exemplo 2.8: sendo o trace s = <*apresenta, pegar_peça, consumir, pegar_peça, consumir, pegar_peça, consumir*>, idCiclos(*s*) entra no caso (iii), parte "senão", porque não há outra sequência igual a que inicia o *trace*. Chama novamente idCiclos com a cauda de *s* (*s*' = <*pegar_peça, consumir, pegar_peça, consumir, pegar_peça, consumir*). Entra agora no caso (ii), parte "então", que identifica duas se-

14 A função apaga(*s*,*c*) retorna o *trace s* sem nenhuma ocorrência do *trace c*.

qüências iguais de <*pegar_peça, consumir*> (valor de n = 1 e m = 2), tem-se então os seguintes valores para as variáveis:

c = <*pegar_peça, consumir*>
i = <>
s_a = <>

Retornando à primeira chamada de idCiclos tem-se os valores finais das variáveis.

c = <*pegar_peça, consumir*>
i = <*apresenta*> ^ <> = <*apresenta*>

C. Identificação de escolha

A identificação de escolha em um processo somente é perceptível nos casos em que a escolha está dentro de um ciclo, caso contrário, o comportamento alternativo nunca será observado. A princípio, como apresentado na regra r3, cada ciclo identificado é uma alternativa de comportamento. Neste caso, para o exemplo 2.4 a função idCiclos identificaria dois ciclos,

< *pedir_peça, receber_peça, devolver*> e
< *pedir_peça, receber_peça, consumir*>

Dando origem aos processos

C = $\mathbf{P_1}$ | $\mathbf{P_2}$; **C**.
$\mathbf{P_1}$ = *pedir_peça* → *receber_peça* → *devolver* → **SKIP**.
$\mathbf{P_2}$ = *pedir_peça* → *receber_peça* → *consumir* → **SKIP**.

que possui indeterminismo entre os processos $\mathbf{P_1}$ e $\mathbf{P_2}$. O processo original do *trace* deveria ser definido como

C = $\mathbf{P_1}$; **C**.
$\mathbf{P_1}$ = *pedir_peça* → *receber_peça* →
(*devolver* → **SKIP** | *consumir* → **SKIP**)

A fim de solucionar este problema procura-se identificar a operação de "escolha" entre os ciclos identificados. Este procedimento dá-se da seguinte forma:

para cada processo identificado como ciclo procura-se nos outros processos uma parte inicial igual. Verificada a existência desta parte, cria-se um novo processo a partir de ambos iniciando com a parte comum e terminando com uma "escolha" entre as partes finais diferentes. O procedimento continua até que não sejam mais encontrados processos com o mesmo início. No exemplo apresentado acima, os processos $\mathbf{P_1}$ e $\mathbf{P_2}$ têm início comum (*pedir_peça* → *receber_peça*) e partes não coincidentes (*devolver* e *consumir*, respectivamente) que são operadas pelo operador "|", criando-se assim, o processo esperado.

2.3.2.3 Reconhecimento de papéis a partir do comportamento

Como processos não consideram relações entre agentes, o que é essencial conforme a definição de papéis, torna-se necessário acrescentar ao procedimento de reconhecimento de processos o aspecto relacional, ou seja, as interações do agente. Nos processos, as interações são consideradas como eventos globais, embora estes eventos não considerem quem é o outro agente na interação. Este problema é solucionado com a alteração na forma de obtenção do *trace*. Considerando o *trace* da observação de um agente como sendo relativo às suas interações com **um** outro, no caso do agente comunicar-se com mais de um agente no intervalo de observação, têm-se vários *traces* (uma para cada agente com quem mantém comunicação) e os seus vários processos reconhecidos. Estes processos reconhecidos são os processos de interação (PI) do agente e sua descrição inclui o papel do agente com quem o PI se relaciona. No caso do agente possuir mais de um PI, ou o agente observado está assumindo mais de um papel (tantos quantos forem os processos identificados) ou o papel é descrito por processos que são executados concorrentemente.[15]

---

15 No caso do agente que tem interações com mais de um agente, cai-se no que em [COS93a] é chamado de estrutura de conversa concorrente ou no caso de estrutura de conversa recursiva.

Da mesma forma que o agente tem um comportamento cíclico, também os atos de comunicação (parte dos processos do agente que é formada pelos eventos globais) apresentam-se ciclicamente formando ciclos de comunicação. Provavelmente, estes ciclos de comunicação estão definidos nos protocolos de comunicação utilizados na sociedade.

2.3.2.4 Identificação do papel

Como apresentado no início da seção 2.3.2, há duas formas de proceder com a identificação por classificação do papel, a partir do *trace*. Em ambas, o agente observador deve possuir um conjunto pré-estabelecido de papéis e seus processos associados, onde cada elemento deste conjunto é chamado de *modelo*. Este conjunto de modelos deve descrever os papéis em função de processos de interação. Caso o papel tenha mais de um PI, considera-se que estes PIs executam concorrentemente formando um processo único e maior que caracteriza o papel.

A primeira forma de identificação por classificação é procurar nos modelos o processo correspondente ao comportamento observado (construído conforme visto acima). Quando encontrado, pode-se dizer que o papel do agente observado é o papel associado ao modelo encontrado. Esta abordagem apresenta como vantagem o fato de se ter uma descrição (o processo) do agente observado mesmo que um processo pré-definido correspondente ao observado não tenha sido encontrado. O processo reconhecido pode ser utilizado de outras formas que não sejam a de associar um papel ao agente, como por exemplo, para prever seu comportamento.

A segunda forma é considerar os processos pré-estabelecidos como sendo uma gramática e verificar a qual destas gramáticas o *trace* pertence. Isso pode ser feito com as técnicas comuns de compilação. Como não existe o passo de construção do processo, esta abordagem apresenta a vantagem de facilmente reconhecer comportamentos complexos, uma vez que pode ser difícil construir o processo a partir da obser-

vação. Uma implementação eficiente seria utilizar inicialmente a segunda forma de identificação e, se não encontrasse o processo do *trace*, utilizar a primeira.

Porderia ser argumentado que este mecanismo é muito rígido, já que o agente deve se comportar exatamente como um dos processos pré-definidos. Esta rigidez pode ser atenuada de duas formas: primeiro, descrevendo para cada papel vários processos que o caracterizam e, segundo, definindo os processos de maneira suficientemente genérica, dando mais "liberdade" ao comportamento dos agentes.

2.3.3 Identificação de papéis por reconhecimento de intenções em planos

Quando um agente assume determinado papel certamente tem alguma intenção ao fazê-lo. O mesmo pode ser dito a respeito de planos; pois se um agente adota certo plano tem uma intenção com isso [BRA 84, COH 87]. Como já foi visto, planos e processos têm a mesma função, de controlar o comportamento do agente. Porém, existem algumas diferenças. A nível de representação, um plano é descrito por ações e um processo por eventos, e a nível de construção, um plano é construído pelo próprio agente enquanto um processo é pré-definido (conforme a definição de agente apresentada). A primeira diferença não é significativa porque ambos descrevem a mesma coisa, ou seja, um conjunto de comportamentos possíveis. Se o comportamento for visto como um diagrama de transição de estados, o plano descreve as transições (por meio de ações) e o processo os estados (eventos). Quanto à segunda diferença, do ponto de vista de um observador, não há diferença se o comportamento observado é pré-determinado ou se foi planejado pelo agente.

O que se propõe com esta técnica de identificação de papéis (resumido na figura 2.7), é identificar a intenção do agente a partir do plano que está atualmente executando. Sabendo a intenção, pode-se saber quais papéis se relacionam a esta intenção. Para saber qual plano um agente está atualmente executando existem a princípio duas

alternativas: o agente observado informar seu plano ou o agente observador reconhecê-lo. Esta segunda abordagem será tratada nesta seção, e para isso será utilizado o método de reconhecimento de planos conforme definido em [ALL 80].

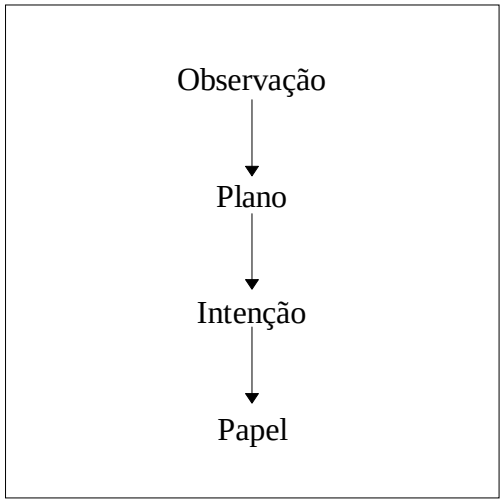

Figura 2.7 - Resumo do processo de identidicação de papéis por reconhecimento de intenções em planos

A indução de planos modelada em [ALL 80] pode ser descrita, de forma geral, como segue. Sendo A um agente que tem como meta adquirir alguma informação sobre um determinado mundo, A cria um plano (etapa de construção de planos) que envolve uma pergunta ao agente B. A executa seu plano perguntando a B sua questão. B recebe a pergunta e tenta induzir o plano de A (etapa de reconhecimento de planos). Neste plano podem existir obstáculos para A que B pode solucionar, assim, B aceita alguns destes obstáculos como suas próprias metas e cria um plano para alcançá-las. B responde a A enquanto executa este plano. O *mundo* é modelado como um conjunto de proposições que representam o que é conhecido sobre o aspecto estático. Este mundo é alterado por ações. *Ações* são descritas por suas pré-condições (proposições que devem ser verdadeiras para que a ação possa ser executada), seus efeitos (proposições que passam a ser verdadeiras depois da execução da ação) e pelo seu corpo. Dado um estado inicial do mundo, W, e uma meta G, um *plano* é uma seqüência

de ações que transformam W em G. A construção de um plano, para uma dada meta G, é encontrar uma ação que tenha como efeito G e avaliar as pré-condições desta ação, sendo que as condições não satisfeitas passam a ser novas metas.

Ainda conforme [ALL 80], a indução de planos pode ser feita por duas abordagens. Primeira, por (re)*construção do plano*, que consiste em iniciar por uma meta que seria esperada do agente e construir um plano objetivando esta meta, simulando o agente alvo. Se o plano construído incluir a ação observada, tem-se o provável plano do agente. É obvio que esta abordagem despende muito tempo, como alternativa tem-se uma segunda abordagem, por *indução do plano*, onde, o plano é reconstruído a partir de ações observadas.

Para a indução de planos são estabelecidos dois operadores (o1 e o2) e algumas regras de inferência (r1 - r4) relacionadas às ações (já que são as ações que podem ser observadas). Os operadores e regras são os seguintes (onde S é o agente que deseja induzir o plano e A o agente observado):

o1. B(P), operador *Acredita*. Exemplo: AB(P), A acredita que vale a proposição P.

o2. W(P), operador *Deseja*. Exemplo: AW(P), A deseja que o mundo alcance um estado em que vale a proposição P.

r1. SBAW(P) $\Rightarrow$ SBAW(ACT). Lê-se: se S acredita que A deseja P, então S também acredita que A deseja ACT, sendo P as precondições para a ação ACT;

r2. SBAW(B) $\Rightarrow$ SBAW(ACT). Lê-se: se S acredita que A deseja B, então S também acredita que A deseja ACT, se a ação B é parte do corpo da ação ACT;

r3. SBAW(ACT) $\Rightarrow$ SBAW(E) se E é efeito da ação ACT.

r4. SBAW($_n$W(ACT)) $\Rightarrow$ SBAW(ACT) sendo *n* um agente. Se A deseja que um outro agente *n* deseje ACT, então A também deseja ACT.

As regras de indução de plano, se lidas ao contrário, podem ser usadas na construção de planos. Por exemplo, a regra r1, chamada de $r1^{-1}$, pode ser usada na

construção de planos da seguinte maneira: se A quer executar ACT, antes deve satisfazer as pré-condições de ACT, ou seja, P.

Exemplo 2.9: considerando um agente observador que pretende identificar o papel de um outro agente, chamado agente observado, e que tenha as seguintes informações a respeito de algumas ações:

a1. ação: *pede_peça*
pré-condição: estado(*sem_peça*)
efeito: estado(*esperar_peça*)
corpo: envia_mensagem(solicita(peça), X)[16]

a2. ação: *recebe_peça*
pré-condição: estado(*esperar_peça*)
efeito: estado(*tem_peça*)
corpo: recebe_mensagem(responde(peça(P)), X)

a3. ação: *consumir*
pré-condição: estado(*tem_peça*)
efeito: estado(s*em_peça*)
corpo: …

e tenha observações do comportamento (neste exemplo, o que pode ser observado do agente são suas ações de comunicação), este agente pode inferir o plano do agente observado pelos passos seguintes (utilizando tanto indução como construção de planos).

i) É observado que o agente enviou a mensagem "solicita(peça)". Usando a regra r2, aplicada à ação observada, "solicita(peça)", temos que o agente deseja, e conseqüentemente, executa a ação a1, *pede_peça*.

ii) Utilizando $r1^{-1}$, se o agente executou a ação a1, então a pré-condição, estar no estado *sem_peça*, era verdade antes da execução.

iii) Usando r3, aplicada a ação *pede_peça*, temos que o agente deseja o efeito desta ação, ou seja, deseja e realmente passa para o estado de *espera_peça*.

16 A comunicação, neste exemplo, é feita por duas ações: envia_mensagem(<mensagem>,<destinatário>) e recebe_mensagem(<mensagem>,<emissor).

iv) É observado que o agente recebeu a mensagem “responde(peça(P))”. Novamente, usando `r2` aplicada à ação observada, temos que executou o corpo da ação `a2` (*recebe_peça*), que é “recebe_mensagem(responde(peça(P)),X)”. E como a pré-condição desta ação está satisfeita (isto é, estar no estado *espera_peça*) pelo passo (ii), o agente observado efetivamente executou `a2`.
Por `r3`, aplicado a ação `a2`, temos que o agente mudou para o estado *tem_peça*.

v) Por `r1`, se o agente deseja estar no estado *tem_peça*, pelo passo (iv), então executa a ação que tem este estado como pré-condição, ou seja, `a3`.

Se a observação parar neste momento, conclui-se que o agente observado tem um plano e que passou pelos estados colocados no diagrama abaixo.

Estados	**Plano**	**Passos**
sem_peça		(i)
⇓	solicita(peça)°	(ii)
esperar_peça	↓	(iii)
⇓	rebebe_peça°	(iv)
tem_peça	↓	(iv)
	***consumir*•**	(v)

Ou seja, o agente observado tinha um plano no qual a ação final, que é justamente o objetivo do plano, é estar consumindo uma peça. Se o papel de Consumidor é caracterizado como um papel que intenciona *consumir*, então o agente observado acima pode ser classificado como um Consumidor.

Neste mecanismo de identificação, ao contrário do mecanismo de comparação com processos pré-definidos, o agente observado não precisa se comportar de forma exatamente igual às pré-definidas para que se possa identificar-lhe o papel. Basta que se identifique uma intenção no agente, o plano que usa para satisfazê-la não é relevan-

° Estas ações foram observadas.

te. Apesar desta vantagem, este mecanismo está sujeito às limitações dos algoritmos de reconhecimento e construção de planos, dentre as quais pode-se citar: o agente observador identifica o plano do agente observado a partir de crenças que tem a respeito das ações que são executadas pelo agente observado, ou seja, tem um "ponto de vista" a respeito das pré-condições, efeitos e corpo das ações, e isto permite que um agente tenha uma visão diferente da "normal" na sociedade possibilitando a identificação errada de planos e papéis; o procedimento de reconhecimento/construção pode se tornar impraticável em casos onde o número de ações (bem como o número de pré-condições e efeitos) é muito grande; e, a identificação correta depende do momento em que se inicia e se pára a observação (problema da parada), pois se estes momentos não forem os certos, ou se identifica um papel errado (talvez como sub-papel, no caso de observar pouco tempo) ou não se consegue identificá-lo.

2.3.4 Comparação entre os mecanismos de identificação de papéis

A. Quanto à eficácia

A princípio, somente o mecanismo por apresentação produz o efeito desejado, ou seja, o papel identificado é realmente o papel que o agente alvo está assumindo. Porém, vale ressaltar que isso é conseguido presupondo que o agente alvo conhece seu papel, o que pode não ser verdade em alguns casos. Os outros mecanismos, devido ao problema da parada, não podem garantir que o papel observado seja realmente o papel do agente.

Comparando os mecanismos que utilizam observação, pode-se dizer que o mecanismo por classifição (de base funcionalista) tem maior chance de acertar o papel real do agente que o mecanismo por indução de intenções em planos (de base intencionalística). Isto se deve ao fato de que no mecanismo por classificação todas as ações

---

• Esta ação foi induzida.

observadas devem satisfazer o modelo do papel, logo, se um agente se comporta exatamente como o modelo de certo papel, ele está assumindo este papel (a menos do problema da parada, é claro). No caso da indução de intenções, de certa forma, o comportamento do agente não é essencial, bastando que o agente tenha determinada intenção que caracteriza um papel. Assim, é possível que se induza intenções parciais, isto é, uma intenção pode ser vista como dividida em sub-intenções, e se uma destas sub-intenções caracterizar outro papel o mecanismo irá falhar. Este mecanismo também não considera as relações do agente alvo com os demais, o que é um problema segundo a definição de papel apresentada.

Em resumo, quanto à eficácia, os mecanismos apresentam a ordem (do mais eficaz para o menos eficaz)

apresentação
⇓
classificação
⇓
intenções.

B. Quanto às restrições e necessidades impostas

O mecanismo por apresentação é o que impõe mais restrições/necessidades, pois os agentes devem conhecer o protocolo de apresentação, devem ter mesmas descrições para os papéis e devem ter conhecimento de seus próprios papéis. Dos comentários já apresentados para estas restrições na definição do mecanismo, conclui-se que estas são muito fortes, principalmente para sociedades abertas.

O mecanismo por classificação necessita uma descrição prévia de papéis em termos de processos e restringe o comportamento dos agentes alvo a estas descrições. O mecanismo por indução necessita de descrições das ações realizadas pelos agentes e da relação papel x intenção, e restringe o grupo de agentes com papel identificável

àqueles que possuem alguma das intenções do conjunto papel x intenção, ou seja, permite maior “liberdade” no comportamento dos agentes.

Quanto às restrições e necessidades, os mecanismos apresentam a ordem (do mecanismo menos carregado de restrições para mais carregado)

intenções

⇓

classificação

⇓

apresentação,

que é justamente a ordem inversa à ordem quanto a eficácia, alias, é justamente a maior carga de restrições que possibilita a maior efeicácia.

C. Quanto à eficiência

O mecanismo por apresentação é certamente o mais eficiente, pois basta que o agente alvo responda ao pedido de identificação. Os outros dois mecanismos, desconsiderando o tempo gasto pelos algoritmos de cada um, levam praticamente o mesmo tempo para identificar o papel de outro agente. Este tempo é determinado pelo número de ações observáveis que estão disponíveis aos mecanismos. Como os dois mecanismos necessitam ciclos no comportamento do agente, devem ficar observando, pelo menos, até que o agente complete o ciclo pela segunda vez. O mecanismo por indução leva uma pequena vantagem, pois pode induzir o fim do ciclo antes que ele realmente aconteça. Portanto, a ordem de eficiência é

apresentação

⇓

intenções

⇓

classificação.

Resumindo, em sociedades não abertas, o melhor mecanismo é o de apresentação (mais eficaz e eficiente). Em sociedades abertas, a escolha do mecanismo dependerá do que é mais importante: eficiência ou eficácia. Caso se opte por eficiência, o melhor mecanismo é por intenções. Caso se opte por eficácia, o melhor mecanismo é por classificação. Obviamente, nada impede que mais de um mecanismo seja utilizado, fazendo uso das vantagens de cada um. Embora isto implique na implementação de critérios de decisão caso os mecanismos dêem resultados diferentes.

3 ESTUDO DE CASO: A SOCIEDADE PRODUTOR-INTERMEDIÁRIO-CONSUMIDOR

3.1 Descrição da sociedade PIC

Algumas experimentações dos mecanismos propostos foram feitas tomando por base a sociedade Produtor-Intermediário-Consumidor (PIC). Em [COS93a] é feita uma primeira descrição desta sociedade em termos de processos funcionais. A sociedade de Produtores e Consumidores é composta principalmente por agentes que assumem justamente estes dois papéis. Resumidamente, Produtores produzem peças e Consumidores as consomem. Para que esta sociedade funcione de maneira eficiente, introduz-se o papel de Intermediário que recebe as peças dos Produtores, guarda-as, e quando uma delas for requisitada, envia-a para um Consumidor. Não obstante esta simplicidade aparente, a sociedade PIC apresenta algumas situações mais complicadas, como por exemplo: “o que acontece quando um Produtor envia uma peça ao Intermediário e não existe espaço para ela?”, ou “quando um Consumidor solicita uma peça ao Intermediário e este não pode enviá-la (por falta no estoque)?”. Além destes problemas, é notória a necessidade dos agentes se conhecerem nesta sociedade, como fica exemplificado no caso de Produtores que precisam conhecer os agentes que assumem papel de Intermediários para poder enviar-lhes suas peças.

A fim de descrever detalhadamente a sociedade PIC, os protocolos de comunicação utilizados são apresentados para descrever o comportamento padrão dos

agentes. Deve-se observar que o modo de funcionamento “normal” aqui especificado para os agentes a fim de experimentar os mecanismos propostos não é único.

3.2 Protocolos de Comunicação

3.2.1 Protocolo Consumidor-Intermediário

Os protocolos de comunicação serão descritos conforme a linguagem de descrição de protocolos apresentada na seção 2.3.1.1. A interação entre Consumidores e Intermediários é descrita no seguinte protocolo:

```
protocol consumidor-intermediário;
        C: Consumidor;
        I: Intermediário;
        (C.init,I.wait) request(peça) ->
        (C.wait,I.send) reply(peça(X)) ->
        (C.init,I.wait).
```

A figura 3.1 também representa este protocolo acrescentando alguns detalhes dos funcionamentos internos dos agentes envolvidos. Cada papel é representado por um retângulo pontilhado, dentro do qual constam eventos (ligados por linhas) nos quais o agente participa. Ainda dentro do retângulo, as partes do texto em itálico representam os estados do agente entre os eventos. Cada evento representa o envio de uma mensagem e é representado por setas saindo do papel origem e chegando no papel destino da mensagem.

Um Consumidor, inicialmente no estado `init`, pede (através da mensagem `request(peça)`) a algum Intermediário previamente conhecido por algum mecanismo de identificação de papel para enviar-lhe uma peça, e fica então esperando o envio da peça. Quando a peça chega (por meio da mensagem `reply(peça(X))`), ela é consumida e o agente volta ao estado inicial.

O agente Intermediário quando recebe um pedido de peça assume um dos seguintes comportamentos: se existir peça no estoque, retira a peça e envia-a ao Consu-

midor; se não existirem peças em estoque, coloca o Consumidor numa lista de espera e assim que receber uma peça será enviada ao primeiro agente desta lista.

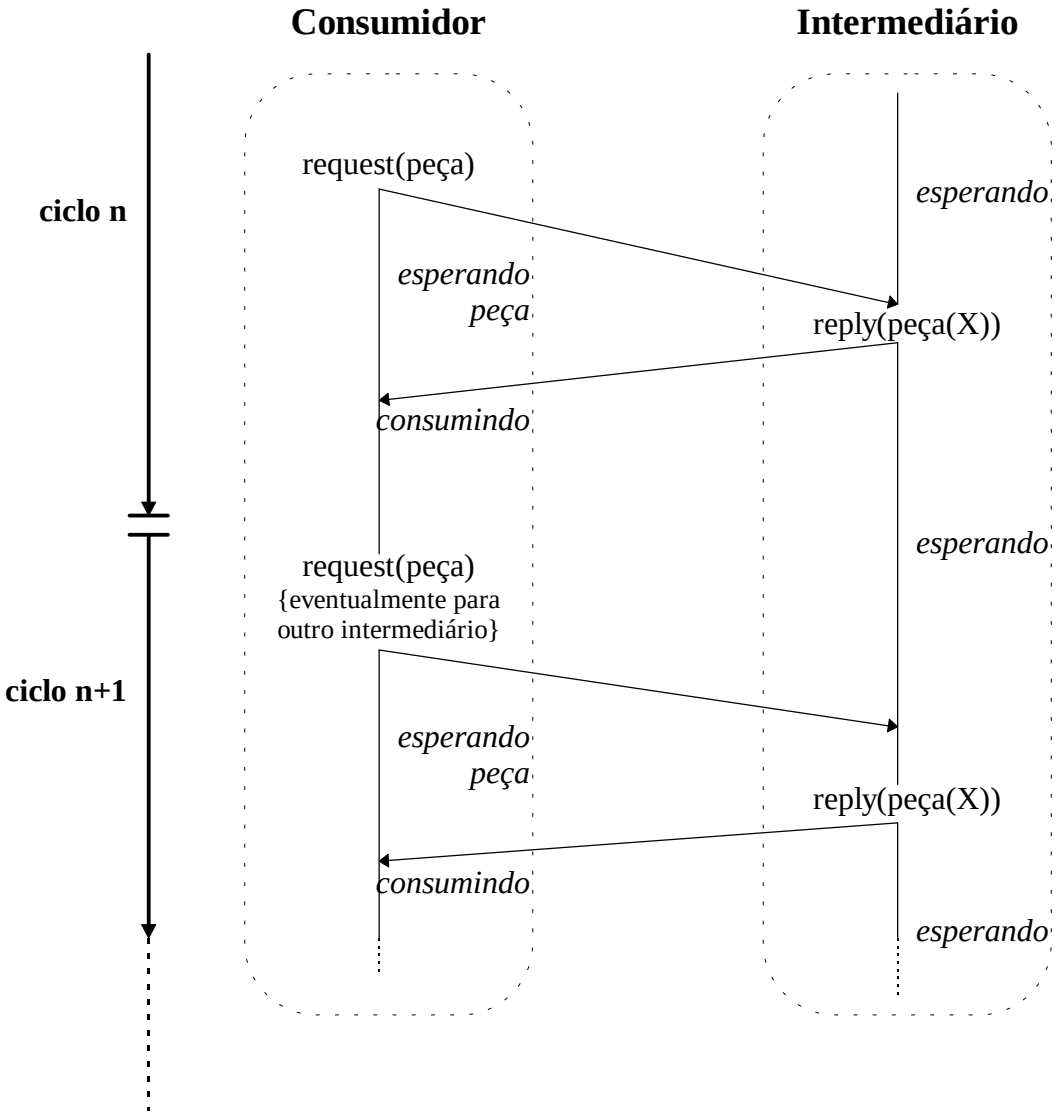

Figura 3.1 - Protocolo para interação Consumidor-Intermediário

3.2.2 Protocolo Produtor-Intermediário

O protocolo de interação Produtor-Intermediário pode se realizar de duas maneiras, conforme descrito nos protocolos abaixo. A primeira descrição também pode ser vista na figura 3.2.

```
protocol produtor-intermediário(1);
        P: Produtor;
        I: Intermediário;
        (P.init,I.wait) request(espaço) ->
                ((P.wait,I.send) reply(espaço nok) ->
                 (P.init,I.wait))
                or
                ((P.wait,I.send) reply(espaço ok) ->
                 (P.send,I.wait) inform(peça(X)) ->
                 (P.init,I.wait)).

protocol produtor-intermediário(2);
        P: Produtor;
        I: Intermediário;
        (P.init.send,I.wait) inform(peça(X)) ->
        (P.init,I.wait).
```

O comportamento do Produtor para o primeiro protocolo inicia com a produção de uma peça seguido de pedido a algum Intermediário por espaço para a colocação da mesma, utilizando para isso a mensagem “request(espaço)”. O Intermediário pode responder de duas formas: “reply(espaço ok)” ou “reply(espaço nok)”, significando que o espaço foi reservado ou que não há espaço, respectivamente. Caso haja espaço, o Produtor envia a peça por meio da mensagem “inform(peça(X))”, onde X é o número da peça produzida. Quando o Intermediário recebe esta mensagem, coloca a peça no estoque. No caso de não haver espaço para a peça, o Produtor pede espaço a outro Intermediário. Os Produtores que optam pelo segundo protocolo simplesmente enviam as peças aos Intermediários, sem antes pedir espaço.

O Intermediário, na interação com o Produtor, tem o seguinte comportamento: ao receber um pedido de espaço e tendo verificado a possibilidade deste no estoque (ou seja, estoque atual + reservas < capacidade máxima) efetua a reserva para o Produtor e responde confirmando. Caso não haja espaço, o Produtor é avisado. Quando o Intermediário recebe a mensagem “inform(peça(X))” de um Produtor, esta peça é colocada no estoque, o número de reservas é diminuído, e, se existe algum Consumidor na lista de espera, manda-lhe a peça.

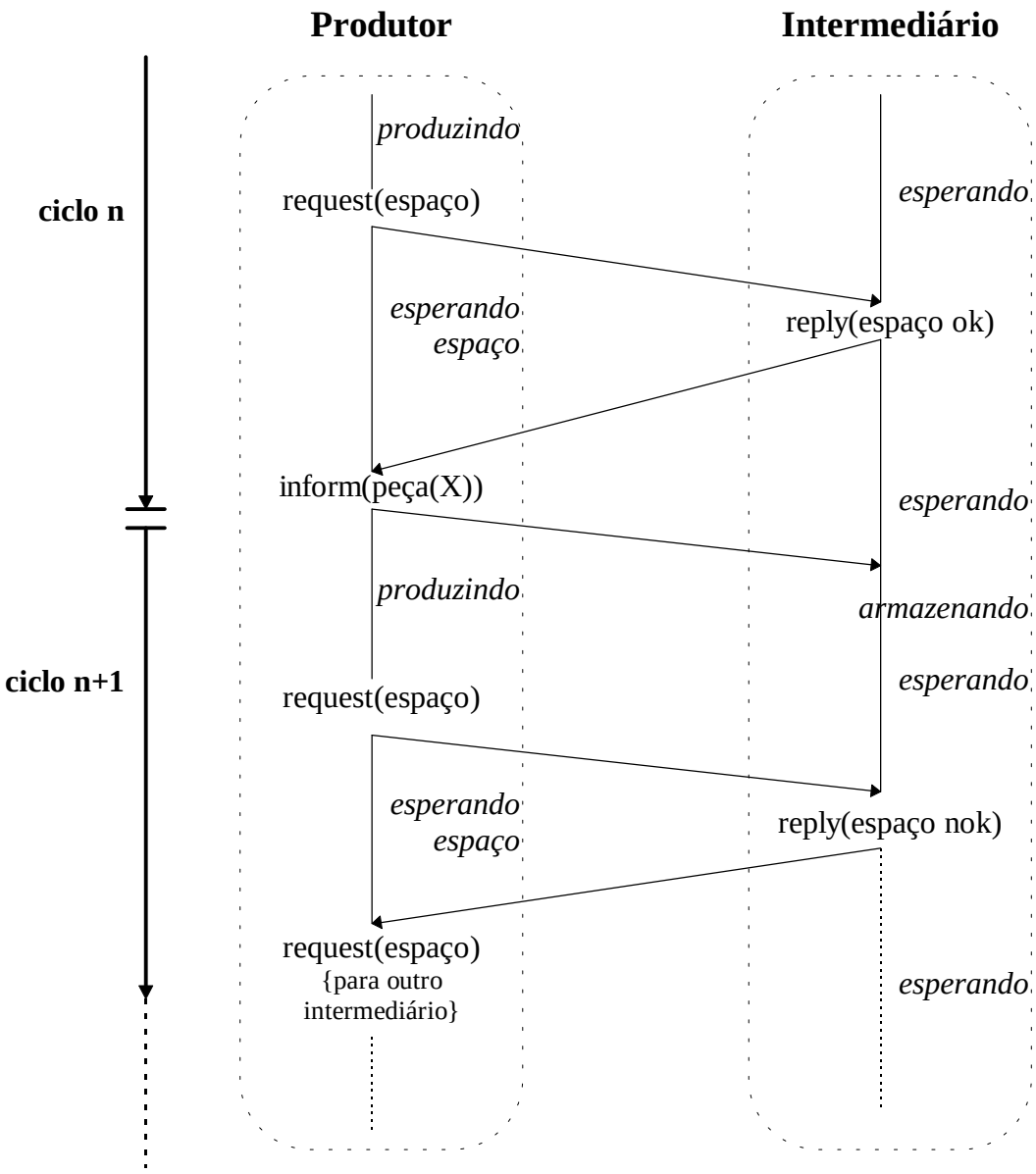

Figura 3.2 - Protocolo para a interação Produtor-Intermediário

3.2.3 Protocolo geral da sociedade PIC

Da união dos dois protocolos apresentados acima, pode-se derivar o protocolo geral da sociedade PIC que liga os Produtores aos Consumidores (cf. figura 3.3, onde o protocolo geral está representado por setas pontilhadas que não consideram o Intermediário).

```
protocol produtor-consumidor;
        C: Consumidor;
        P: Produtor;
        (C.init,P.wait) request(peça) ->
        (C.wait,P.send) inform(peça(X)) ->
        (C.init,P.wait).
```

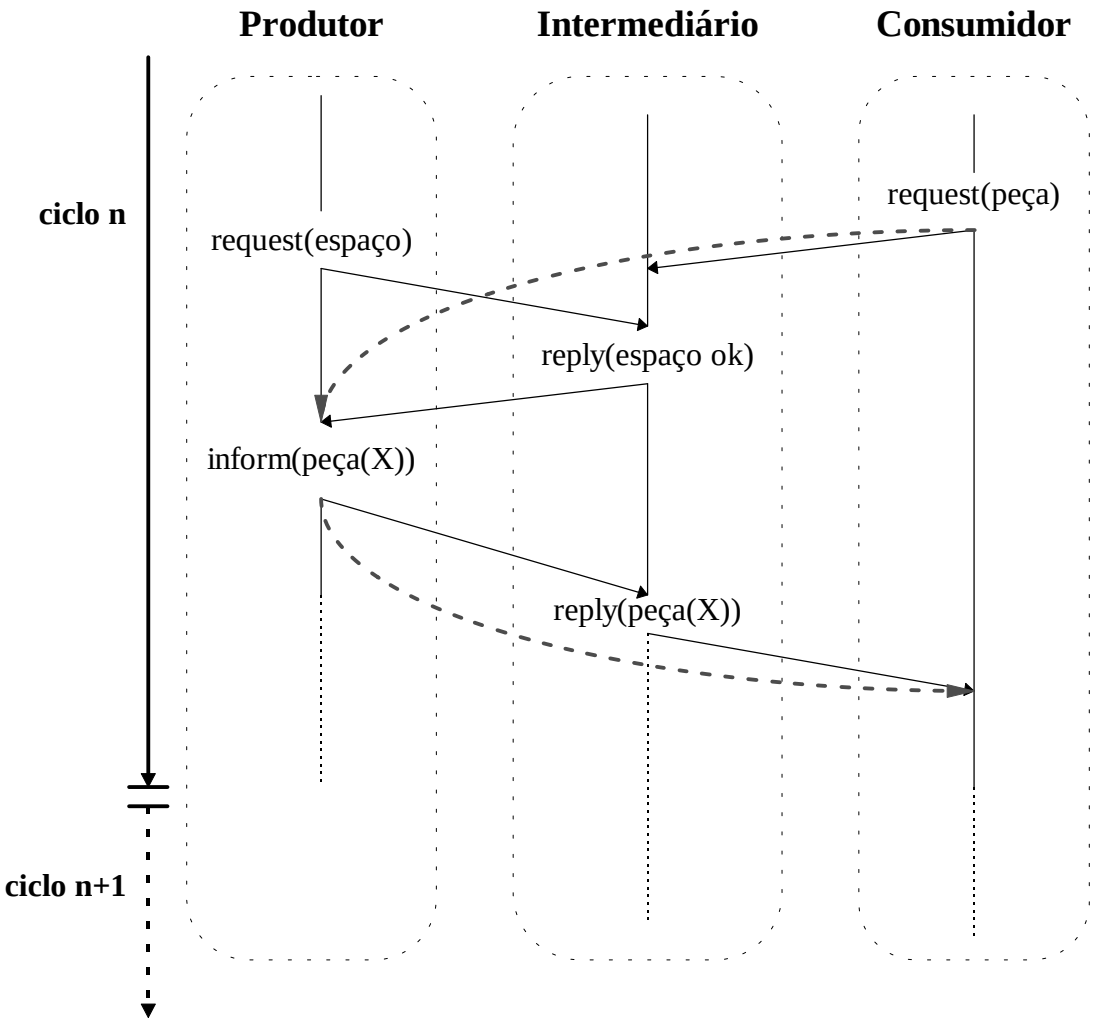

Figura 3.3 - Protocolo geral da sociedade PIC

Como pode ser observado na figura 3.3, o papel de Produtor é relativo a um único papel, o de Intermediário. O mesmo acontece com o papel de Consumidor, que também é relativo ao papel de Intermediário. Já o papel de Intermediário é relativo a dois papéis, o de Produtor e o de Consumidor, simultaneamente.

3.3 Implementação da sociedade PIC

3.3.1 Definição da sociedade

A implementação da sociedade PIC, para fins de validação e teste dos mecanismos propostos, foi feita na bancada de simulação de sociedades de agentes desenvolvida por Luiz Muniz [MON 93] para ambiente MacProlog do Macintosh.[17] O seguinte trecho de código define os tipos dos agentes da sociedade (que no caso PIC, correspondem aos tipos de papéis) e quais deles são ativos:

```
:- def_Produtor('Rocha').
:- def_Produtor('Rosa').
:- def_Consumidor( 'Jomi').
:- def_Consumidor( 'Rafa').
:- def_Intermediario('Renata').

:- def_Inspetor( 'Dalton', processos).
:- def_Inspetor( 'Jr.', planos).


:- remember(agent_windows,['Consumidor',
                           'Produtor',
                           'Intermediário',
                           'Inspetor'
                          ]),

   remember(agentes_activos,[
             'Jomi','Rafa',
             'Rocha','Rosa',
             'Renata',
             'Dalton',
             'Jr.'
   ]).
```

Na parte inicial do código, os agentes são definidos instanciando nomes aos tipos de agentes, na segunda parte a bancada é informada de quais agentes serão ativados. No exemplo apresentado acima, dois agentes são Consumidores (Jomi e Rafa) dois são Produtores (Rocha e Rosa) e um é Intermediário (Renata), estes são os agentes que fazem a sociedade funcionar. Além destes cinco, mais dois agentes, Dalton e Jr, participam da sociedade, apenas observando a sociedade e utilizando os mecanismos de identificação de papel.

---

17 Detalhes complementares desta implementação também podem ser consultados em [HÜB 94], como por exemplo: como os agentes são definidos, alguns fontes, acompanhamentos da simulação, etc.

3.3.2 Definição dos agentes

A implementação de cada agente na bancada[18]consiste em definir uma estrutura para o agente que é composta dos seguintes componentes (a figura 3.4 mostra o fluxo de controle na estrutura do agente).

i) **Controle**: corresponde à *gestão* realizada sobre as unidades que compõem o agente, ou dito de outra forma, determina quando as unidades serão ativadas e quando inibidas.

ii) **Unidades**: são componentes independentes que funcionam em paralelo. São definidas de acordo com o tipo de *comportamento* ou função que se deseja obter. As unidades são construídas com as funções pré-definidas na bancada para as unidades e com a chamada de módulos.

iii) **Módulos**: correspondem às *ações* que um agente pode executar. Uma ação muda ou o estado interno do agente ou do ambiente. Os módulos são construídos com as funções pré-definidas para eles na bancada e com chamadas de métodos.

iv) **Métodos**: correspondem a *procedimentos básicos*; são usados na construção dos módulos. Na sua construção, além das funções básicas definidas na bancada, os métodos podem chamar outros métodos e cláusulas Prolog.

v) **Estados internos**: correspondem à *memória* do agente, e são constituídos de conhecimento a respeito do mundo, dos outros agentes e de si mesmos; incluem ainda uma lista com as mensagens recebidas.

18 Cf. [MUN 93] para maiores detalhes da espeficicação de agentes na bancada.

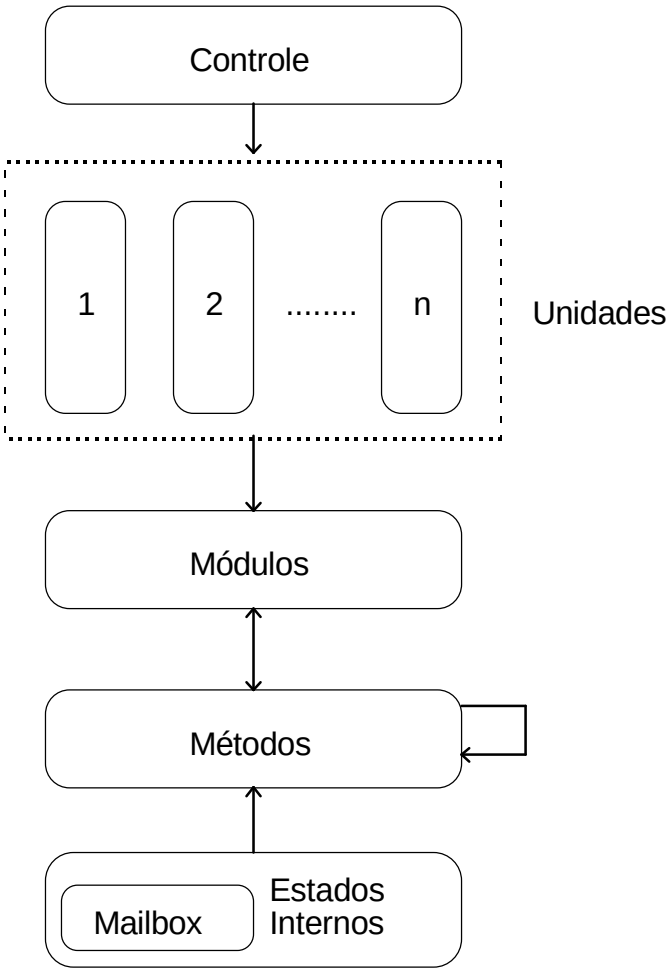

Figura 3.4 - Estrutura de um agente na bancada

Exemplo 3.10: para dar uma noção de como são definidos os agentes, o seguinte trecho de código[19] descreve o tipo Consumidor (o que possui descrição mais simples):

```
Agente Consumidor

Controle
se(mensagem,executar(1),nada)
se(sensor(estado(inicial)),executar(2),nada)
se(sensor(estado(mandar_pedido)),executar(3),nada)
se(sensor(estado(consumir)),executar(4),nada)

Unidades
1: recebe_mensagem
2: pede_apresentação
```

19 Esta listagem não corresponde literalmente ao código implemenado porque foi reescrita para facilitar a leitura, embora nela constem as mesmas informações.

```
3: pede_produto
4: consome
```

Módulos

recebe_mensagem

```
    proc_msg(M,O) <- le_mensagem(M,O)
```

pede_apresentação

```
    envia_mensagem(request(role),todos) <-
         escreve(['Mandou pedido de Identificação']) &
         muda_estado(espera_apresentação)
```

pede_produto

```
    envia_mensagem(request(peca), Inter) <-
         procura_intermediario(Inter) &
         escreve(['Mandou pedido']) &
         muda_estado(esperar_peca)
```

consome

```
   escreve(['Tempo restante de Consumo = ',X]) <-
         diminui_tempo_peca(X)
```

Métodos

```
% Se apresenta aos outros, caso seja solicitado
%
proc_msg(request(role),O) <-
        meuPapel(P) &
        envia_mensagem(reply(role(P)),O),

% Recebe apresentação dos outros agentes
%
proc_msg(reply(role(Papel)),O) <-
        papeis(L) &
        external(removeEle(L,papel(O,_), L2)) &
        external(append(L2,[papel(O,Papel)],NovoL)) &
        substitui(papeis(_),papeis(NovoL)) &
        verifica_mudanca_estado(Papel) &
        escreve(['
        Identificação de papel (por apresentação) ',O,NovoL]),

% Recebe peca do intermediario
%
proc_msg(reply(peca(X)),Inter) <-
          guarda(peca(4)) &
          muda_estado(consumir) &
          escreve(['Recebeu produto:',peca(X)]),

% Muda para estado mandar pedido se ja existe um intermediario
%
verifica_mudanca_estado('Intermediário') <-
        estado(espera_apresentação) &
        muda_estado(mandar_pedido),
verifica_mudanca_estado(_) <- true,

muda_estado(E) <-
        escreve(['Mudou para estado:',E]) &
        substitui(estado(_), estado(E)),

procura_intermediario(Inter) <-
        papeis(L) &
        external(
        findall(I, member(papel(I,'Intermediário'),L),LInt)) &
        external(escolhe(LInt, Inter)),
```

```
diminui_tempo_peca(NovoTempo) <-
        peca(X) &
        external(X > 0) &
        external(NovoTempo is X - 1) &
        substitui(peca(_),peca(NovoTempo)),

diminui_tempo_peca(terminou) <-
        peca(0) &
        aumenta_consumido &
        muda_estado(mandar_pedido) &
        esquece(peca(_)),

aumenta_consumido <-
        consumido(E) & external(E1 is E +1) &
         substitui(consumido(_), consumido(E1)) &
        escreve(['Pecas consumidas=',E1]),

escreve(Mem) <-
        external(append([A,':::: '],Mem,Mens)) &
        external(escreve(Mens)),
escreve(_) <- true


Estados Internos
meuPapel('Consumidor'),
true,
papeis([]),
consumido(0),
estado(inicial)
```

Se o código acima for analisado, observa-se que o ciclo de vida do Consumidor passa pelos estados:

i) inicial: quando o agente está neste estado uma das unidades executáveis[20] é `pede_apresentação` que, assim que termina sua execução, passa o agente para o estado `espera_apresentação`;

ii) espera apresentação: neste estado a única unidade executável é `recebe_mensagem` que, assim que recebe a apresentação de algum Intermediário, muda para o estado `mandar_pedido`;

iii) mandar pedido: a unidade executável é `pede_produto` que manda um pedido de peça a algum Intermediário e passa para o estado `espera_peca`;

iv) espera peça: a única unidade executável é `recebe_mensagem` que, assim que recebe uma peça do Intermediário, muda para o estado `consumir`;

v) consumir: a unidade executável é `consome`, o agente fica por um tempo consumindo a peça e, assim que termina, muda para o estado `mandar_pedido`, recomeçando o ciclo.

---

20 As unidades executáveis são aquelas permitidas pelo nível de controle.

Além das funções descritas pelos estados pelos quais o agente Consumidor passa, pode-se observar que a unidade `recebe_mensagem` realiza outras funções que não são próprias do papel de consumir. Por exemplo, o código acima permite que o agente se apresente a outros quando recebe a mensagem “request(role)”. Convém notar também que este é o caso de um agente natural da sociedade, ou seja, que utiliza o protocolo de apresentação (está explícito no código a forma do protocolo) e tem conhecimento de qual papel está desempenhando (informação que está nos estados internos do agente).[21]

3.4 Identificação de papéis na sociedade PIC

3.4.1 Identificação de papéis por apresentação

Conforme a implementação realizada, na sociedade PIC a identificação de papéis é feita das duas formas básicas apresentadas no capítulo 2, ou seja, diretamente por meio de um protocolo de apresentação ou por observação. O mecanismo por apresentação foi implementado conforme proposto na seção 2.3.1. Na simulação, este mecanismo é utilizado pelos agentes “natos”, aqueles que estão ativos deste o início da simulação. Assim, antes de um Produtor iniciar seu funcionamento normal, ele utiliza o protocolo de apresentação para identificar o papel de Intermediário em algum agente. Uma vez identificado pelo menos um agente Intermediário, o Produtor pode iniciar a produção da primeira peça. O mesmo vale para os Consumidores. Como os Intermediários não tomam iniciativa nas comunicações, não passam pelo processo de descobrir os papéis dos agentes com quem se comunicam. A princípio, apenas se comportam conforme as regras dos protocolos de comunicação (se recebeu determinada mensa-

---

21 A descrição para os outros agentes da sociedade PIC é bastante semelhante à feita para o Consumidor. Maiores detalhes podem ser consultados no anexo 5.1 onde estão as descrições de todos os agentes.

gem responde com a mensagem ditada no protocolo sem verificar o papel do outro agente).

Como foi colocado na apresentação deste mecanismo, aos agentes é imposto:

- que saibam qual é seu papel atual;
- que tenham descrições comuns para os papéis (‘Produtor’, ‘Consumidor’ e ‘Intermediário’, isto é, os papéis são identificados por *strings* que têm a mesma interpretação para todos os agentes “natos”);
- que possam responder a um pedido de apresentação, ou seja, que saibam utilizar o protocolo de apresentação.

Como este mecanismo somente é utilizado pelos agentes “natos” estes requisitos não são pertinentes, sendo resolvidos na implementação dos agentes. Todavia, este mecanismo, por ser o mais rápido dos propostos, coloca a sociedade em funcionamento com maior rapidez. Podem ser observadas na simulação colocada no anexo 5.1.1 a parte inicial do funcionamento da sociedade e a respectiva identificação por apresentação.

3.4.2 Identificação de papéis por observação

Para identificação de papéis por observação criou-se um tipo especial de agente, o tipo “Inspetor”. Este agente pode utilizar um dos dois mecanismos de observação, e para sua construção foram feitas modificações na bancada de modo que todas as mensagens enviadas fiquem armazenadas para consulta. Estas mensagem são armazenadas conforme o formato utilizado na bancada que é composto pelas seguintes informações: momento do envio da mensagem, agente emissor, corpo da mensagem e agente destino; sendo representado conforme a definição

```
<mensagem> ::= mensagem (<momento>, <emissor>,
                         <mensagem>, <destino>)
```

Por exemplo, a mensagem “mensagem(40, Rocha, request(espaço), Renata)” representa o envio da mensagem “request(espaço)” para a agente Renata, pelo agente Rocha, no momento 40 da simulação.

Tendo estas informações disponíveis, antes de aplicar um dos dois mecanismos de observação, as mensagens recebem um tratamento para transformá-las em *traces* como segue abaixo:

i) Por se procurar identificar o papel de um agente alvo, são selecionadas somente as mensagens onde este agente participou, seja como emissor ou como destinatário.

ii) São retiradas as eventuais mensagens de apresentação, por não serem relevantes para a identificação do papel.

iii) As mensagens restantes são transformadas em uma lista de ações pelas regras:
Para cada mensagem do agente alvo,
se o agente é emissor:
mensagem(T, E, M, D) ⇒ envia_mensagem(M’, D)
se o agente é destinatário:
mensagem(T, E, M, D) ⇒ recebe_mensagem(M’, E).
Sendo que na transformação M ⇒ M’ os números das peças são retirados das mensagens, conforme as seguintes transformações:
inform(peça(X)) ⇒ inform(peça(_))[22] ou
reply(peça(X)) ⇒ reply(peça(_)) ou
M ⇒ M

Exemplo 3.11: Sendo as seguintes mensagens de uma sociedade com três agentes, Jomi, Rocha e Renata:

mensagem(0, Rocha, request(role), Jomi),
mensagem(0, Rocha, request(role), Rocha),
mensagem(0, Rocha, request(role), Renata),
mensagem(0, Jomi, request(role), Jomi),
mensagem(0, Jomi, request(role), Rocha),

---

22 Estas transformações no conteúdo da mensagem são necessárias para que, duas mensagens de envio de peças sejam consideradas iguais para fins de identificação de ciclos. Caso contrário, as ações
envia_mensagem(inform(peça(1)),Int) e
envia_mensagem(inform(peça(2)),Int)
não seriam consideradas iguais e não formariam um ciclo.

mensagem(0, Jomi, request(role), Renata),
mensagem(1, Rocha, reply(role(Produtor)), Rocha),
mensagem(1, Jomi, reply(role(Consumidor)), Rocha),
mensagem(2, Rocha, reply(role(Produtor)), Jomi),
mensagem(2, Jomi, reply(role(Consumidor)), Jomi),
mensagem(4, Renata, reply(role(Intermediário)), Rocha),
mensagem(6, Renata, reply(role(Intermediário)), Jomi),
mensagem(6, Jomi, request(peca), Renata),
mensagem(10, Rocha, request(espaço), Renata),
mensagem(12, Renata, reply(espaço_ok), Rocha),
mensagem(13, Rocha, inform(peça(1)), Renata),
mensagem(17, Renata, reply(peca(1)), Jomi),
mensagem(18, Rocha, request(espaço), Renata),
mensagem(20, Renata, reply(espaço_ok), Rocha),
mensagem(21, Rocha, inform(peca(2)), Renata),
mensagem(22, Jomi, request(peca), Renata),
mensagem(26, Renata, reply(peca(1)), Jomi).

O passo (i) seguido do passo (ii), para a agente Renata, transformarão esta lista de mensagens em

mensagem(6,Jomi,request(peça),Renata),
mensagem(10,Rocha,request(espaço),Renata),
mensagem(12,Renata,reply(espaço_ok),Rocha),
mensagem(13,Rocha,inform(peça(1)),Renata),
mensagem(17,Renata,reply(peça(1)),Jomi),
mensagem(18,Rocha,request(espaço),Renata),
mensagem(20,Renata,reply(espaço_ok),Rocha),
mensagem(21,Rocha,inform(peça(2)),Renata),
mensagem(22,Jomi,request(peça),Renata),
mensagem(26,Renata,reply(peça(2)),Jomi).

Do passo (iii) obtem-se a lista de ações para a agente Renata

recebe_mensagem(request(peça),Jomi),
recebe_mensagem(request(espaço),Rocha),
envia_mensagem(reply(espaço_ok),Rocha),
recebe_mensagem(inform(peça(_)),Rocha),
envia_mensagem(reply(peça(_)),Jomi),
recebe_mensagem(request(espaço),Rocha),
envia_mensagem(reply(espaço_ok),Rocha),
recebe_mensagem(inform(peça(_)),Rocha),
recebe_mensagem(request(peça),Jomi),
envia_mensagem(reply(peça(_)),Jomi).

3.4.2.1 Identificação por observação e classificação

O agente Inspetor, que utiliza este mecanismo[23] para identificar o papel dos agentes da sociedade, tem modelos de processos para os papéis. Estes modelos são descritos, em Prolog, pelo predicado

papel(Nome_do_Papel, Lista_de_Processos_de_Interação).

onde, os elementos de Lista_de_Processos_de_Interação são

pi(PRPI, Lista_de_Processos).

Estes processos de interação (PI) representam as interações que o papel descrito tem com um outro papel, chamado papel relativo ao processo de interação (PRPI). Por exemplo, um agente Produtor tem um processo de interação onde o PRPI é Intermediário; um agente Intermediário tem, no mínimo, dois processos de interação, um com PRPI Produtor e outro com PRPI Consumidor. Cada processo de interação, além do PRPI, possui uma lista de processos que descreve como se dá esta interação. Por sua vez, os processos são descritos pelo predicado

p(Nome_do_Processo, Corpo),

onde o Corpo pode ser:

- o processo **SKIP**;
- o processo **STOP**;
- a chamada de outro processo, na forma "p(Nome_Processo)";
- o operador "→", na forma "p(prefix, Evento → Processo)";
- o operador "|", na forma "Processo ou Processo";
- o operador ";", na forma "Processo seq Processo".

Na figura 3.4 pode-se ver um exemplo simples de uso desta notação, no caso, descrevendo o papel de Consumidor. As descrições das figuras 3.4, 3.5 e 3.6 formam o conjunto de modelos que o inspetor possui inicialmente. O papel de Produtor tem duas descrições, uma que considera a resposta negativa de espaço e outra que não a considera. Isto se faz necessário porque um Produtor pode nunca receber essa resposta ne-

---

23 O mecanismo de identificação de papel por observação e classificação foi apresentado na seção 2.3.2.

gativa (pelo fato do Intermediário sempre ter espaço). Se isso acontecer, o papel deste agente nunca será conhecido sem o primeiro modelo de Produtor. O mesmo comentário é válido para o papel de Intermediário. Nos modelos de papéis apresentados, vale notar que o papel de Intermediário possui dois processos de interação porque seu papel é relativo tanto ao papel de Consumidor como ao de Produtor.

```
papel('Consumidor',[

pi( 'Intermediário', [

 p('MAIN',  p('CICLO')),
 p('CICLO', p('CICLO'(1)) seq p('CICLO')),
 p('CICLO'(1), p(prefix, envia_mensagem(request(peca))->
               p(prefix, recebe_mensagem(reply(peca(X)))->
               p(skip))))
])  % Fim do pi
]). % Fim do papel
```

Figura 3.5 - Processo para os agentes com papel Consumidor

```
papel('Produtor', [
pi( 'Intermediário', [

   p('MAIN', p('CICLO')),
   p('CICLO', p('CICLO'(1)) seq p('CICLO')),
   p('CICLO'(1), p(prefix, envia_mensagem(request(espaco)) ->
                  p(prefix, recebe_mensagem(reply(espaco_ok)) ->
                  p(prefix, envia_mensagem(inform(peca(X))) ->
                  p(skip)))))
])  % Fim do pi
]). % Fim do papel

papel('Produtor', [
pi( 'Intermediário', [

   p('MAIN', p('CICLO')),
   p('CICLO', p('CICLO'(1)) seq p('CICLO')),
   p('CICLO'(1), p(prefix, envia_mensagem(request(espaco)) ->
                 (p(prefix, recebe_mensagem(reply(espaco_ok)) ->
                  p(prefix, envia_mensagem(inform(peca(X))) ->
                  p(skip)))
                 ou
                  p(prefix, recebe_mensagem(reply(espaco_nok))->
                  p(skip))
                 )))
])  % Fim do pi
]). % Fim do papel
```

Figura 3.6 - Processos para os agentes com papel Produtor

```
papel( 'Intermediário', [

pi( 'Consumidor', [

   p('MAIN',  p('CICLO')),
   p('CICLO', p('CICLO'(1)) seq p('CICLO')),
   p('CICLO'(1), p(prefix, recebe_mensagem(request(peca))->
                  p(prefix, envia_mensagem(reply(peca(X)))->
                  p(skip))))
]),  % Fim do pi

pi( 'Produtor', [

   p('MAIN', p('CICLO')),
   p('CICLO', p('CICLO'(1)) seq p('CICLO')),
   p('CICLO'(1), p(prefix, recebe_mensagem(request(espaco)) ->
                  p(prefix, envia_mensagem(reply(espaco_ok)) ->
                  p(prefix, recebe_mensagem(inform(peca(X))) ->
                  p(skip)))))
])  % Fim do pi
]). % Fim do papel


papel( 'Intermediário', [

pi( 'Consumidor', [

   p('MAIN',  p('CICLO')),
   p('CICLO', p('CICLO'(1)) seq p('CICLO')),
   p('CICLO'(1), p(prefix, recebe_mensagem(request(peca))->
                  p(prefix, envia_mensagem(reply(peca(X)))->
                  p(skip))))
]),  % Fim do pi

pi( 'Produtor', [

   p('MAIN', p('CICLO')),
   p('CICLO', p('CICLO'(1)) seq p('CICLO')),
   p('CICLO'(1), p(prefix, recebe_mensagem(request(espaco)) ->
                  (p(prefix, envia_mensagem(reply(espaco_ok)) ->
                   p(prefix, recebe_mensagem(inform(peca(X))) ->
                   p(skip)))
                  ou
                   p(prefix, envia_mensagem(reply(espaco_nok))->
                   p(skip))
                  )))
])  % Fim do pi
]). % Fim do papel
```

Figura 3.7 - Processos para os agentes com papel Intermediário

Apresentados os modelos de papéis, surge a necessidade de construir os *traces* referentes à observação do agente alvo para poder compará-los aos modelos. Para isto são acrescentadas as seguintes transformações à lista de ações resultante dos passos de transformação na lista de mensagem (veja figura 3.8):

i) Para cada agente com quem o agente alvo teve interações é criado um *trace*.

ii) Para cada *trace* observado é construído um processo de interação associado.[24] De todos os PIs, tem-se uma lista de PIs.

iii) A lista de PIs observada é comparada com a lista de PIs dos modelos. Esta comparação consiste em verificar se os processos de interação do modelo são subconjunto dos processos de interação observados.

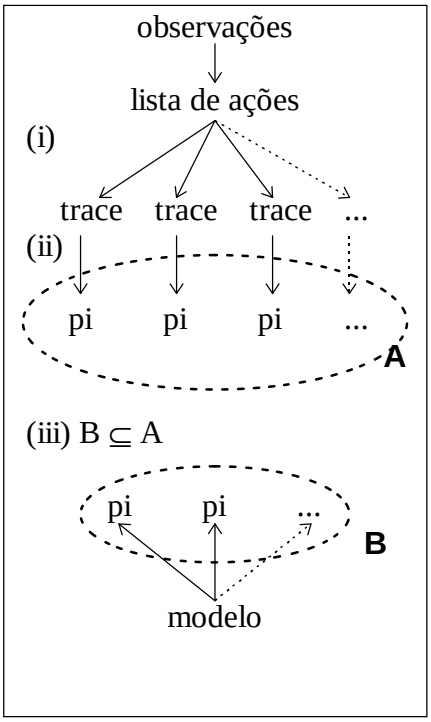

Figura 3.8 - Esquema da identificação de papéis por observação e classificação.

Exemplo 3.12: dando seqüência ao exemplo 3.1, os passos acima realizam as seguintes transformações:

Pelo passo (i), a agente Renata tem os seguintes traces[25]:

24 A construção dos processos é realizada como apresentado na seção 2.3.2.2.

25 Os *traces* apresentam o formato: t(<nome do agente>, <eventos observados>).

```
t(Rocha,[       recebe_mensagem(request(espaço)),
                envia_mensagem(reply(espaço_ok)),
                recebe_mensagem(inform(peca(_))),
                recebe_mensagem(request(espaço)),
                envia_mensagem(reply(espaço_ok)),
                recebe_mensagem(inform(peca(_)))
        ])
t(Jomi, [       recebe_mensagem(request(peca)),
                envia_mensagem(reply(peca(_))),
                recebe_mensagem(request(peca)),
                envia_mensagem(reply(peca(_)))
        ])
t(Renata, []).
```

O passo (ii) cria os processos:

Processo de Renata relacionado ao agente Rocha

```
MAIN    = CICLO.
CICLO   = (CICLO(1)) ; (CICLO).
CICLO(1) =
        recebe_mensagem(request(espaço)) ->
        envia_mensagem(reply(espaco_ok)) ->
        recebe_mensagem(inform(peca(_))) ->
        SKIP.
```

Processo de Renata relacionado ao agente Jomi

```
MAIN = CICLO.
CICLO = (CICLO(1)) ; (CICLO).
CICLO(1) =
        recebe_mensagem(request(peca)) ->
        envia_mensagem(reply(peca(_))) ->
        SKIP.
```

O passo (iii) junta estes dois processos de interação formando um conjunto do qual a descrição de Intermediário feita na figura 3.6 é subconjunto, logo, o agente observado é Intermediário. Como a agente Renata poderia ter interações com mais de um Produtor e/ou Consumidor, poderiam existir vários processos de interação (mais PIs do que no modelo), mas para caracterizar a agente como Intermediária seriam necessários pelo menos um PI com um Consumidor e um PI com um Produtor.

O passo três coloca um problema interessante. É condição que o Inspetor conheça o papel dos agentes relacionados com o agente alvo, ou melhor, os processos de interação tem um PRPI que, a princípio, o Inspetor pode não conhecer. No exemplo 3.3, o Inspetor deve conhecer os papéis de Rocha e Jomi (Produtor e Consumidor,

respectivamente) para que o papel de Renata possa ser classificado como Intermediário. Sendo assim, a primeira vez que o agente Inspetor vai classificar os papéis dos agentes não pode considerar o PRPI.

Considerando o problema da primeira identificação de papel e o problema da parada, faz-se necessária uma revisão futura dos papéis já identificados, quando um maior número de papéis for conhecido (o que pode resolver o problema da primeira identificação) e existirem mais eventos a serem observados (o que atenua o problema da parada). Como o problema da parada não tem solução, o agente Inspetor periódicamente revisa os papéis observados.

Para o exemplo 3.3, no passo (ii), a revisão do papel de Renata passaria a ser:

Processo de Renata relacionado ao **papel Produtor** [26]

```
MAIN    = CICLO.
CICLO   = (CICLO(1)) ; (CICLO).
CICLO(1) =
        recebe_mensagem(request(espaço)) ->
        envia_mensagem(reply(espaco_ok)) ->
        recebe_mensagem(inform(peca(_))) ->
        SKIP.
```

Processo de Renata relacionado ao **papel Consumidor**

```
MAIN = CICLO.
CICLO = (CICLO(1)) ; (CICLO).
CICLO(1) =
        recebe_mensagem(request(peca)) ->
        envia_mensagem(reply(peca(_))) ->
        SKIP.
```

Estes dois processos de interação realmente caracterizam um Intermediário, por considerarem o papel dos agentes relacionados.

Na implementação deste mecanismo, em vez de construir os PI para os *traces* e compará-los a modelos (passos (ii) e (iii)), procura-se verificar se os *traces* observados pertencem aos *traces* definidos por algum processo dos modelos de papéis. Esta forma de implementação é adotada porque é mais rápida e simples que a construção do processo e sua comparação com os modelos. Como método para implementação desta

---

26 Agora é considerada a relação com Produtor e não mais com Rocha, já que o papel deste é conhecido.

abordagem foram utilizadas as noções de verificação sintática, comum na implementação de linguagens de programação.

Assim como uma gramática define um conjunto de sentenças de uma linguagem, um processo define um conjunto de *traces*, ou seja, processos são vistos como especificações de uma gramática. Cada processo é visto como equivalente a um não-terminal e os eventos destes processos como equivalentes aos terminais de uma gramática. Uma vez feita esta relação, constrói-se um verificador sintático que tem como entradas um processo e um *trace* e dá como resultado um valor verdade: se o *trace* pertence ou não a linguagem definida pelo processo. O algoritmo abaixo descreve a forma geral de funcionamento deste mecanismo de identificação de papel. [27]

Para cada papel do conjunto de modelos,
    para cada PI deste papel e
        para cada *trace* observado
            utilizar o verificador sintático com o PI e
            o *trace* selecionados
            Se *trace* pertence ao PI
            então passa para o próximo PI
        Se nenhum *trace* pertence ao PI
        então desiste do papel

Se algum papel corresponde aos *traces* observados,
então retorna o papel encontrado como sendo o papel do agente observado.
Senão utiliza o procedimento de construção de processos a partir de *traces* para conseguir, pelo menos, uma descrição do papel observado, sem conseguir classificá-lo.

Na simulação do anexo 5.1.1 pode-se observar o funcionamento deste mecanismo. No caso, o agente Dalton (tipo Inspetor) identifica e revisa papéis por observação e classificação, sendo que ele tem descrições de papéis para todos os agentes da sociedade. Já na simulação do anexo 5.2.2, o Inspetor não tem modelos para todos os agentes observados, assim, para os agentes que não consegue classificar utiliza o procedimento de construção de processos.

---

27 No anexo 5.1.3.1 pode-se conferir a implementação deste mecanismo em Prolog.

3.4.2.2 Identificação por observação e reconhecimento de intenções em planos

A implementação do mecanismo de observação e reconhecimento de planos e respectivas intenções segue a proposta da seção 2.3.3. As regras de reconhecimento de planos implementadas (r1 a r4) criam uma lista de estados pelos quais o agente observado passou num determinado período de tempo. Além da lista de estados, as regras produzem também uma lista de ações executadas que é composta pelas ações observadas mais aquelas que podem ser induzidas a partir dos estados que o agente passou. Esta lista de ações é justamete o plano do agente observado e cada ação é uma intenção do plano, embora seja dada especial atenção às ações que não são observadas diretamente mas induzidas. A relação de papéis e suas respectivas intenções na sociedade PIC é

Papel	Intenções
Consumidor	consumir
Produtor	produzir
Intermediário	enviar peça e receber peça

No caso do Consumidor e do Produtor as intenções não são diretamente observadas, mas precisam ser inferidas, ou seja, existe uma seqüência de ações que podem ser observadas e que levam o agente à ação objetivo. Por exemplo, a intenção de consumir pode ser obtida de ações anteriores (como pedir peças) que já bucavam satisfazer determinada intenção. O Intermediário é caracterizado por possuir duas intenções que são observáveis, logo, a noção de planejamento e intenção ficam um pouco perdidas. Isto faz com que este mecanismo (para o caso do Intermediário) se aproxime do mecanismo de observação e classificação, já que está descrevendo as ações e não a intenção implícita nestas.

As ações conhecidas pelo agente Inspetor para a sociedade PIC tem como pré-condição e efeito o estado interno do agente observado. Obviamente estas informações são crenças do Inspetor a respeito das ações realizadas pelos outros agentes. As ações possuem o formato

```
<acao> ::= acao ( <Cód. ação>, <Nome>, <Pré-condições>,
                                        <Efeitos>,
                                        <Corpo>)
```

e são as seguintes:

```
acao(a1,pedir_peca,     [estado(sem_peca)],
                        [estado(espera_peca)],
                        [envia_mensagem(request(peca),_)]).
acao(a2,receber_peca,   [estado(espera_peca)],
                        [estado(tem_peca)],
                        [recebe_mensagem(reply(peca(X)),_)]).
acao(a3,consumir,       [estado(tem_peca)],
                        [estado(sem_peca)],
                        []).

acao(a4,produzir,       [estado(produzir)],
                        [estado(pedir_espaco)],
                        []).
acao(a5,pedir_espaco,   [estado(pedir_espaco)],
                        [estado(espera_resposta)],
                        [envia_mensagem(request(espaco),_)]).
acao(a6,receber_espaco,[estado(espera_resposta)],
                        [estado(enviar)],
                        [recebe_mensagem(reply(espaco_ok),_)]).
acao(a7,receber_espaco,[estado(espera_resposta)],
                        [estado(pedir_peca)],
                        [recebe_mensagem(reply(espaco_nok),_)]).
acao(a8,enviar,         [estado(enviar)],
                        [estado(produzir)],
                        [envia_mensagem(inform(peca(X)),_)]).

acao(a20,enviar_peca,   [estado(com_peca)],
                        [estado(espera)],
                        [envia_mensagem(reply(peca(X)),_)]).
acao(a21,receber_peca, [estado(espera)],
                        [estado(espera)],
                        [recebe_mensagem(inform(peca(X)),_)]).
```

Observa-se que a ação `receber_espaco` tem duas especificações, isto porque a resposta de um pedido de espaço pode tanto ser positiva como negativa.

Como apresentado na seção 2.3.3, a noção de ciclos de atividade no agente não é considerada, por exemplo, basta que o agente intencione uma vez consumir e já é considerado um Consumidor. Para resolver este problema, no momento de comparar as ações observadas com as intenções de um papel, a intenção deve aparecer pelo menos duas vezes.

De forma geral, o algoritmo deste mecanismo é:

- Na a lista de ações do agente alvo (obtida pelos passos descritos no final da seção 3.4.2) acrescentar as ações induzidas pelas regras de indução de planos.
- Para cada papel, pegar a lista de intenções correspondentes e duplicá-la.
- Verificar se esta lista de intenções é subconjunto da lista de ações obtida no primeiro passo. Se é subconjunto, tem-se um papel, senão tenta-se com outro papel.

No simulação do anexo 5.1.1 pode ser observado o agente Jr. (tipo Inspetor) utilizando este mecanismo de identificação de papel.

3.4.3 Testes com agente tipo Transformador

Com o objetivo de testar os mecanismos de identificação de papéis indiretos (que utilizam observação) em situações mais complexas, foram acrescentados a sociedade Produtor - Consumidor agentes do tipo Transformador. Este tipo de agente torna a identificação de papéis mais difícil devido ao seu comportamento que assemelha-se as vezes a um Consumidor e outras a um Produtor. O ciclo de funcionamento do Transformador consiste em pedir duas vezes peças a um Intermediário formando um sub-ciclo de consumo (parte em que é Consumidor), transformar estas duas peças numa terceira e enviar a peça resultante da transformação de volta para o Intermediário (parte em que é Produtor). O seguinte protocolo descreve as interações que compõem o comportamento deste tipo de agente:

```
protocol transformador-intermediário;
        T: Transformador;
        I: Intermediário;
        (T.init,I.wait) request(peça) ->
        (T.wait,I.send) reply(peça(X)) ->
        (T.request,I.wait) request(peça) ->
        (T.wait,I.send) reply(peça(X)) ->
        (T.send,I.wait) request(espaço) ->
                ((T.wait,I.send) reply(espaço nok) ->
                 (T.init,I.wait))
                or
                ((T.wait,I.send) reply(espaço ok) ->
                 (T.send,I.wait) inform(peça(X)) ->
                 (T.init,I.wait)).
```

Na primeira simulação onde este agente participa (cf. por exemplo o anexo 5.1.1), o agente Inspetor utiliza o mecanismo de identificação por classificação e não conhece o papel de Transformador, ou seja, não existe um modelo para ele. Desta simulação se obteve os seguintes resultados:

- inicialmente, o Inspetor identificou o papel do Transformador como sendo Consumidor, já que no início de suas atividades o Transformador consome duas peças, exatamente como um Consumidor.
- Na revisão do papel, se o agente Transformador já enviou alguma peça, o Inspetor não pode identificar-lhe o papel, podendo apenas criar uma descrição para ele.

Numa segunda simulação, onde o Inspetor conhece a seguinte descrição para o tipo Transformador:

```
papel('Transformador', [

pi( 'Intermediário', [

 p('MAIN',  p('CICLO')),
 p('CICLO', p('CICLO'(1)) seq p('CICLO')),
 p('CICLO'(1), p(prefix,envia_mensagem(request(peca))->
               p(prefix,recebe_mensagem(reply(peca(X)))->
               p(prefix,envia_mensagem(request(peca))->
               p(prefix,recebe_mensagem(reply(peca(X)))->

               p(prefix,envia_mensagem(request(espaco)) ->
               (p(prefix,recebe_mensagem(reply(espaco_ok)) ->
                p(prefix,envia_mensagem(inform(peca(X))) ->
                p(skip)))
               ou
                p(prefix,recebe_mensagem(reply(espaco_nok)) ->
                p(skip))
               )))))))
])])
```

foram estes os resultados:

- inicialmente, o Inspetor também identificou o papel do Transformador como sendo Consumidor, pelo mesmo motivo apresentado acima.
- Na revisão, se o agente Transformador já enviou alguma peça, o Inspetor pode determinar-lhe o papel correto.

Na terceira simulação, o Inspetor utiliza o mecanismo de identificação de papel por indução de intenções e não conhece o papel Transformador. Resultados:

- inicialmente, o Inspetor identificou o papel do Transformador como sendo Consumidor, já que no início de suas atividades o Transformador intenciona por duas vezes consumir peças, exatamente como um Consumidor.
- Na revisão do papel, como as duas intenções de consumir ainda são consideradas, o Inspetor continua classificando o Transformador como Consumidor. Neste aspecto os dois mecanismos de identificação diferem para a mesma condição (não conhecer uma descrição para o papel).

Na quarta simulação, o Inspetor pode identificar o agente Transformador como um agente que tem as intenções de consumir e produzir. Chegou-se aos seguintes resultados:

- inicialmente, o Inspetor identificou o papel do Transformador como sendo Consumidor.
- Na revisão, se o agente Transformador já enviou alguma peça, o Inspetor pode determinar-lhe o papel correto.

Ainda assim, esta quarta simulação levanta uma nova necessidade ao mecanismo por indução, ou seja, a ordem em que o agente Inspetor consulta as intenções dos papéis passa a ser relevante. Se as intenções para Consumidor forem consultadas antes das intenções para Transformador, mesmo na revisão, o Inspetor irá classificar o papel do Transformador como Consumidor.

3.4.4 Resultados obtidos a partir das experimentações

Com a implementação da sociedade PIC, pode-se verificar detalhadamente o funcionamento e as características de cada um dos três mecanismos de identificação de papéis. Como já foi apresentado na seção 2.3.4, a utilização de um protocolo de apresentação, apesar de impor várias restrições às sociedades abertas, é o mecanismo mais eficiente e, por isto, foi utilizado pelos agentes, aqui chamados, “natos”.

Para o mecanismo de identificação por observação e classificação, a noção de processo de interação (PI) foi de grande importância, pois constitui a base para o processo de identificação de papéis. Os PIs permitem ver a sociedade como um conjunto

de papéis e as relações entre estes, obtendo-se uma visão organizacional da sociedade. Justamente por possibilitar tal visão, este mecanismo se mostrou como o mais eficaz dos três.

A implementação comprovou também que o mecanismo de reconhecimento de intenções em planos pode não apresentar o resultado esperado. Como ficou comprovado com a inclusão do agente tipo Transformador e sua classificação como Consumidor, mesmo tendo subsídios para, pelo menos, classifica-lo como “indeterminado” (como faz o mecanismo de classificação na mesma situação). Apesar deste problema, o mecanismo continua sendo uma uma boa solução por não impor tantas restrições, como acontece com o mecanismo por apresentação, e por ser mais eficiênte que o mecanismo de classificação.

4 CONCLUSÕES

Este trabalhou apresentou mecanismos que contribuem para solucionar os problemas de conhecimento e atuação na migração de agentes, sendo a migração uma característica necessária em sistemas complexos, heterogêneos, adaptativos, evolutivos, continuamente em funcionamento e abertos. Na base desta solução está a noção de papéis que os agentes podem assumir. Desta forma, foram desenvolvidos três mecanismos de identificação de papéis, bem como a comparação entre eles e sua adequação a tipos de migração. Estes três mecanismos foram testados em simulações na sociedade Produtor - Consumidor, onde puderam ser comprovadas as características de cada um.

Os três mecanismos apresentados são os seguintes:

i) **Identificação de papéis por protocolo de apresentação**: nesta solução, foi proposta uma linguagem de descrição de protocolos (LDP) e uma especificação de protocolo de apresentação nesta LDP. Os agentes que utilizam este mecanismo conseguem se identificar com rapidez, porém necessitam conhecer várias informações "locais" da sociedade, o que pode ser muito restritivo para um agente migrante.

ii) **Identificação de papéis por observação e classificação**: esta solução procura classificar o agente observado em um papel de um conjunto pré-definido de papéis. Neste conjunto, os papéis são descritos por meio de processos de interação (PI). Para isto, desenvolveu-se a noção de PI. Foram desenvolvidas duas formas de proceder a classificação: construir uma especificação do agente a partir da observação das suas ações e verificar se esta pertence ao conjunto pré-definido de papéis; e, verificar se o comportamento do agente confere com as execuções possíveis para algum dos papéis pré-definidos. Este mecanismo é mais adequado para sociedades abertas e tem boa precisão no resultado apresentado, porém, a identificação do papel de um agente pode ser demorada.

iii) **Identificação de papéis por reconhecimento de intenções em planos**: este mecanismo baseia-se na existência de uma relação entre intenções e papéis. A partir das ações observadas para o agente, procura-se saber qual seu plano, sua intenção e, conseqüentemente, seu papel. Para isto foi implementado um procedimento de indução de planos. Este mecanismo também é adequado para sociedades abertas, no entanto, a identificação, embora satisfatória, nem sempre é completamente correta.

O trabalho também apresenta alguns resultados intermediários que serviram de base para o resultado principal (a identificação de papéis):

i) Especificação de uma linguagem de descrição de protocolos.

ii) Elaboração da noção de papel social, sua definição e forma de descrição em termos de processos de interação.

iii) Desenvolvimento de um algoritmo para criação de modelos a partir de observação, no caso, obter a descrição do papel. Este algoritmo, com algumas adaptações, pode ter utilidade em outras áreas, por exemplo: na área de orientação a objetos, poderia ser utilizado para criar um modelo de um objeto, ou melhor, a forma de interação com este; em Processamento de Língua Natural, poderia ser utilizado para extrair a gramática de um texto.

4.1 Trabalhos futuros

No decorrer dos estudos foram identificados alguns assuntos para trabalhos futuros que são listados abaixo.

- Assim como os agentes precisam conhecer-se, também as sociedades precisam conhecer os agentes das outras sociedades para poder solicitar-lhe uma migração.
- Uma restrição comum aos três mecanismos são as crenças do agente que identifica papéis. No mecanismo por apresentação, o agente que responde ao pedido de identificação tem uma crença no seu papel atual; no mecanismo por classificação, o agente observador tem crenças a respeito dos modelos de papéis; e, no mecanismo por indução de planos, o agente observador tem crenças a respeito das ações realizadas na sociedade. Em virtude desta característica comum, faz-se necessário atribuir aos agentes um mecanismo de revisão de crenças que lhes permita também revisar os papéis identificados.

- A identificação de intenções em planos poderia ser aperfeiçoada considerando os seguintes casos: o agente alvo possuir mais de um plano e executá-los simultaneamente; o agente alvo, em função de alterações em suas crenças e/ou objetivos, alterar seu plano atual; etc.
- Em relação ao mecanismo de identificação por classificação, pederia-se estudar ainda os seguintes aspectos:
    - Determinar critérios para equivalência entre Processos de Interação, assim o processo construído não precisa ser exatamente igual ao do modelo, basta serem equivalentes.
    - Fornecer resultados parciais, ou seja, não considerar toda a especificação do papel; ou não considerar todo o trace observado.
    - Estudar melhor o caso em que o agente planeja suas ações, i.e., não tem um comportamento muito constante; ou, vendo o problema de outra forma, o caso de um agente que muda de papel.
    - Como um agente poderia aprender novos modelos de papéis.
- Em [BOR 94] é utilizado um Sub-Agente de Comunicação (SAC) para tratar dos problemas de comunicação do agente, novos trabalhos de implementação poderiam utilizar esta idéia criando um Sub-Agente de Apresentação responsável pelos problemas de apresentação que o agente possui.
- Sendo possível identificar os papéis de um agente, e assim montar uma estrutura da organização da sociedade, a solução para o problema da integridade funcional pode ser procurada.

5 ANEXOS

5.1 Fontes dos agentes da sociedade PIC

Como já existe um descrição detalhada dos agentes do tipo Consumidor no exemplo 3.1, a descrição não será refeita aqui. Igualmente, o agente Transformador também não é descrito por ser uma composição de outros agentes. Os demais tipos básicos de agentes (Produtor, Intermediário) e o agente Inspetor (no qual pode ser vista a implementação dos mecanismos de identificação de papéis) são apresentados em código igual ao da implementação, embora alguns predicados não ligados diretamente ao problema tenham sido suprimidos.

5.1.1 Produtor

```
def_Produtor(A) :-

del_props(A),
set_prop(A,id_agente,A),
set_prop(A,incremento_tempo,1),
set_prop(A,metanivel,[se(mensagem,executar(1),nada),
                      se(sensor(estado(inicial)),executar(2),nada),
                      se(sensor(estado(produzir)),executar(3),nada),
                      executar(4)]),


% ----------------------------
%           Unidades
% ----------------------------
set_prop(A,unidade1, [recebe_mensagem]),
set_prop(A,unidade2, [pede_apresentação]),
set_prop(A,unidade3, [produz]),
set_prop(A,unidade4, [pede_espaco, envia]),

% ----------------------------
%            Modulos
% ----------------------------
set_prop(A,recebe_mensagem,[
```

```
    proc_msg(M,O) <- le_mensagem(M,O)
]),

% Pede apresentação para os outros agentes
%
set_prop(A,pede_apresentação, [
      envia_mensagem(request(role),todos) <-
         external(escreve([A,'+++  Mandou pedido de Identificação'])) &
         muda_estado(espera_apresentação)
]),

set_prop(A,produz,[

% Caso em que cria a peca (se nao ha peca sendo produzida)
% peca(X), X = numero da peca
%
     guarda(peca(X)) <-
            estado(produzir) &
            ªpeca(_) &
            produzidas(X) &
            guarda(tempo_peca(4)) &
            external(escreve([A,'+++  Iníciou produção da peça',X])),

% Caso em que faz um passo de producao (se ha peca e seu tempo > 0)
%
     substitui(tempo_peca(_),tempo_peca(NovoT)) <-
            estado(produzir) &
            peca(_) &
            tempo_peca(T) &
            external(T > 0) &
            external(NovoT is T - 1) &
            external(escreve([A,'+++  Passo de produção',NovoT])) &
            verifica_mudanca_estado(NovoT)
]),

set_prop(A,pede_espaco,[
      envia_mensagem(request(espaco),Inter) <-
              estado(pedir_espaco) &
              procura_intermediario(Inter) &
              external(
              escreve([A,'+++  Mandou pedido de espaco para',Inter])) &
              muda_estado(espera_espaco)
]),

set_prop(A,envia,[
      envia_mensagem(inform(peca(X)),Inter) <-
              estado(enviar) &
              espaco_reservado_em(Inter) &
              esquece(peca(X)) &
              esquece(tempo_peca(_)) &
              aumenta_produzidas &
              external(escreve([A,'+++  Mandou peça',X,'para',Inter])) &
              muda_estado(produzir)
]),

% ----------------------------
%            Metodos
% ----------------------------
set_prop(A,metodos,[

proc_msg(M,O) <-
      external(escreve([A,'+++  Recebida mensagem',M,'de',O])) & fail,

% Se apresenta aos outros, caso seja solicitado
%
proc_msg(request(role),O) <-
        meuPapel(P) &
```

```
        envia_mensagem(reply(role(P)),O),

% Recebe apresentação dos outros agentes
%
proc_msg(reply(role(Papel)),O) <-
        papeis(L) &
        external(removeEle(L,papel(O,_), L2)) &
        external(append(L2,[papel(O,Papel)],NovoL)) &
        substitui(papeis(_),papeis(NovoL)) &
        external(escreve(
        [A,'+++  Identificação  (por apresentação) do papel de ',O,NovoL]))
&
        verifica_mudanca_estado(Papel),

% Recebe resposta quanto ao espaco
%
proc_msg(reply(espaco_ok), O) <-
        substitui(espaco_reservado_em(_), espaco_reservado_em(O)) &
        muda_estado(enviar),
proc_msg(reply(espaco_nok), O) <-
        muda_estado(pedir_espaco),

% Muda para estado produzir se j  existe um Intermediário
%
verifica_mudanca_estado('Intermediário') <-
        estado(espera_apresentação) &
        muda_estado(produzir),
% Muda para estado enviar se acabou o tempo de producao da peca
%
verifica_mudanca_estado(0) <-
        estado(produzir) &
        muda_estado(pedir_espaco),
verifica_mudanca_estado(_) <- true,

muda_estado(E) <-
        external(escreve([A,'+++  Mudou para estado:',E])) &
        substitui(estado(_), estado(E)),

procura_intermediario(Inter) <-
        papeis(L) &
        external(findall(I, member(papel(I,'Intermediário'),L), LInt)) &
        external(escolhe(LInt, Inter)),

aumenta_produzidas <-
        produzidas(Nr) &
        external(NovoNr is Nr + 1) &
        substitui(produzidas(_), produzidas(NovoNr))
]),

% ----------------------------
%       Estados Internos
% ----------------------------
set_prop(A,estados_internos, [
      meuPapel('Produtor'),
      true,
      produzidas(1),                % numero de pecas produzidas
      papeis([]),
      estado(inicial),
      espaco_reservado_em(ninguem)  % local para onde a peca sera enviada
      % Alem destes ainde existem os seguintes estados internos (dinamicos)
      %    peca(X), onde X , o n£mero da peça
      %    tempo_peca(X), onde X , o tempo de consumo que falta para
      %                   a peca atual
]).
```

5.1.2 Intermediário

```
def_Intermediario(A) :-

del_props(A),

set_prop(A,id_agente,A),

set_prop(A,incremento_tempo,1),

set_prop(A,metanivel,[executar(1)]),

% ----------------------------
%           Unidades
% ----------------------------
set_prop(A,unidade1, [recebe_mensagem,atende_pendencias]),

% ----------------------------
%            Modulos
% ----------------------------
set_prop(A,recebe_mensagem,[
    proc_msg(M,O) <- le_mensagem(M,O)
]),

set_prop(A,atende_pendencias,[
    substitui(pendencias(_),pendencias(R)) <-
         estoque(X) & external(X =\= 0) &
         pendencias([C|R]) &
         proc_msg(request(peca),C) &
         escreve(['Lista pendencias diminuida para: ',R])
]),

% ----------------------------
%            Metodos
% ----------------------------
set_prop(A,metodos,[

proc_msg(M,O) <-
      escreve(['Recebida mensagem',M,'de',O]) & fail,

% Recebe pedido de peca de um consumidor
%
proc_msg(request(peca),Cons) <-
      diminui_estoque(EAtual) &
      pecas([H|Resto]) &
      substitui(pecas(_), pecas(Resto)) &
      envia_mensagem(reply(H),Cons) &
      escreve(['Estoque diminuido para: ',EAtual,Resto]) &
      escreve(['Enviado ',H,' para:',Cons]) ,

% Caso em que nao ha estoque para mandar
% entao, colocar na lista de pendencias
%
proc_msg(request(peca),Cons) <-
     pendencias(P) &
     external(append(P,[Cons],NovoP)) &
     substitui(pendencias(_), pendencias(NovoP)) &
     escreve(['Lista de pendencias aumentada para:', NovoP]),

% Recebe pedido de espaco
%
proc_msg(request(espaco),Prod) <-
    estoque(E) &
    reservas(Re) &
    capacidade_max(CP) &
```

```
    external(ETotal is E + Re) &
    external(ETotal < CP) &
    % Ok, pode confirmar a reserva
    envia_mensagem(reply(espaco_ok), Prod) &
    external(ReN is Re + 1) &
    substitui(reservas(_), reservas(ReN)) &
    escreve(['Confirmado espaco para',Prod,'Reservas=',ReN]),

% Nao ha  espaco
proc_msg(request(espaco),Prod) <-
    envia_mensagem(reply(espaco_nok), Prod) &
    escreve(['Negado espaco para',Prod]),

% Recebe peca do produtor
%
proc_msg(inform(M),Prod) <-
    % Tira a reserva
    reservas(Re) &
    external(ReN is Re - 1) &
    substitui(reservas(_), reservas(ReN)) &
    % coloca a peca no estoque
    pecas(P) &
    external(append(P,[M],P1)) &
    substitui(pecas(_), pecas(P1)) &
    aumenta_estoque(EAtual) &
    escreve(['Estoque aumentado para: ',EAtual,P1]),


% Responde a solicitacao de apresentação
%
proc_msg(request(role),O) <-
        meuPapel(P) &
        envia_mensagem(reply(role(P)),O) &
        escreve(['Resposta de apresentação para ',O]),

% Recebe apresentação dos outros agentes
%
proc_msg(reply(role(Papel)),O) <-
        papeis(L) &
        external(removeEle(L,papel(O,_), L2)) &
        external(append(L2,[papel(O,Papel)],NovoL)) &
        substitui(papeis(_),papeis(NovoL)) &
        escreve(['Identificação de papel',O,NovoL]),

    aumenta_estoque(EAtual) <-
         estoque(E) &
         capacidade_max(Max) &
         external(E < Max) &
         external(EAtual is E + 1) &
         substitui(estoque(_),estoque(EAtual)),

    diminui_estoque(EAtual) <-
         estoque(E) &
         external(E > 0) &
         external(EAtual is E - 1) &
         substitui(estoque(_),estoque(EAtual)),

escreve(Mem) <-
        external(append([A,'....  '],Mem,Mens)) &
        external(escreve(Mens)),
escreve(_) <- true

]),

% -----------------------------
%       Estados Internos
% -----------------------------
```

```
set_prop(A,estados_internos, [
        meuPapel('Intermediário'),
        true,
        estoque(0),    % quantidade atual de pecas no estoque
        reservas(0),   % quantidade atual de espacos reservados
        pecas([]),
        capacidade_max(3),
        pendencias([]),
        papeis([])
]).
```

5.1.3 Inspetor

```
def_Inspetor(A,TipoIdentPapel)  :-
% Tipo de Identificação de papel pode ser
%     processos ou planos
%

del_props(A),

set_prop(A,id_agente,A),

set_prop(A,incremento_tempo,10),

set_prop(A,metanivel,[executar(1), executar(2)]),


% -----------------------------
%           Unidades
% -----------------------------
set_prop(A,unidade1, [Identificação]),
set_prop(A,unidade2, [revisao]),

% -----------------------------
%            Módulos
% -----------------------------
set_prop(A,Identificação,[
    substitui(papeis(_), papeis(NovoLP)) <-
      external(recall(agentes_activos,AA)) &
      papeis(P) &
      identParaTodos(AA,P,NovoLP) &
      external(escreve([A,'***  Papeis Identificados: '])) &
      external(writel(NovoLP))
]),


set_prop(A,revisao,[
    substitui(papeis(_), papeis(NovoLP)) <-
      papeis(P) &
      diminuiTempoRevisao(P,NovoLP) %&
      %external(escreve([A,'***  Tempos para revisao de papeis: '])) &
      %external(writel(NovoLP))
]),

% -----------------------------
%            Métodos
% -----------------------------
set_prop(A,metodos,[

identParaTodos([],P,P),
identParaTodos([Ag|RestoA],Papeis,NovoLP) <-
        external(not(member(papel(Ag,_,_),Papeis))) &
        identificaParaUm(Ag, TipoIdentPapel,PapelAg) &
        tempoParaRevisao(TmpRev) &
```

```
        external(append(Papeis,[papel(Ag,PapelAg,TmpRev)],L3)) &
        identParaTodos(RestoA,L3,NovoLP),
identParaTodos([_|RestoA],P,NovoLP) <-
        identParaTodos(RestoA,P,NovoLP),

identificaParaUm(Ag, processos, PapelAg) <-
        external(escreve(
        [A,'***  Papeis para: ',Ag, ' (por processos)'])) &
        external(identificaPapelPorProcessos(Ag,PapelAg)),
identificaParaUm(Ag, planos, PapelAg) <-
        external(escreve([A,'***  Papeis para: ',Ag, ' (planos)'])) &
        external(identificaPapelPorPlanos(Ag,PapelAg)),

diminuiTempoRevisao([],[]),
diminuiTempoRevisao([papel(Ag,P,Tempo)|RP],
                    [papel(Ag,NovoPapel,NovoTempo)|RLP]) <-
        revisaPapel(TipoIdentPapel, Ag, P, NovoPapel, Tempo,
                    NovoTempo) &
        diminuiTempoRevisao(RP,RLP),

revisaPapel(processos,Ag,P,NovoPapel,Tempo,NTmp) <-
        external(NovoTempo is Tempo - 1) &
        external(NovoTempo == 0) &
        tempoParaRevisao(NTmp) &
        papeis(LPapeis) &
        external(escreve([A,'*** Revisando o papel de ',Ag])) &
        external(revisaPapelPorProcessos(Ag,NovoPapel,LPapeis)),
revisaPapel(planos,Ag,P,NovoPapel,Tempo,NTmp) <-
        external(NovoTempo is Tempo - 1) &
        external(NovoTempo == 0) &
        tempoParaRevisao(NTmp) &
        papeis(LPapeis) &
        external(escreve([A,'*** Revisando o papel de ',Ag])) &
        external(revisaPapelPorPlanos(Ag, NovoPapel, LPapeis)),
revisaPapel(_, Ag,P,P,Tempo,NovoTempo) <-
        external(NovoTempo is Tempo - 1)
]),

% ----------------------------
%       Estados Internos
% ----------------------------
set_prop(A,estados_internos, [
        meuPapel('Inspetor'),
        true,
        % A lista de papeis tem o seguinte formato:
        %   papel(NomeAg,Papel,TempoRestanteParaReverPapel)
        %
        papeis([papel(A,'Inspetor',0)]),
        tempoParaRevisao(3)
]).
```

5.1.3.1 Código para identificação de papéis por observação e classificação[28]

```
% Identifica a o papel pela primeira vez
%
identificaPapelPorProcessos(Ag,Papel) :-
     comunicacoesA(Ag,LCom),
     transformaEmAcoes(Ag,LCom,LAcoes),
     findall(t(OutAg,Trace),
```

---

28 Foi omitido desta parte de código aquela que descreve os papéis dos agentes porque já foi descrita no corpo da dissertação.

```
             transformaEmTrace(LAcoes,Trace,OutAg),
             LTraces), %writel(LTraces),

     ( (papel(Papel, PIs),
       verifPIs(PIs, LTraces, []))
     ; (transLTracesToLProcs( LTraces, LProcs),
        impLProcsRel(Ag, LProcs)),
        fail
     ),
     escreve(['Papel de ',Ag,' e'' ', Papel, '.']),nl,!.


% Revisa o papel identificado
% (considera o papel dos ag. com quem tem interacoes)
%
revisaPapelPorProcessos(A, Papel, LPapeis) :-
     comunicacoesA(A,LCom),              % writel(LCom),
     transformaEmAcoes(A,LCom,LAcoes), % writel(LAcoes),
     findall(t(Ag,Trace),
             transformaEmTrace(LAcoes,Trace,Ag),
             LTraces),
     ( revisaPorPertinencia( A, LTraces, Papel, LPapeis)
     ; (trocaAgPorPapel(LTraces,LTracesP,LPapeis),
        revisaPorConstrucao(  A, LTracesP, Papel, LPapeis))
     ),
     comparaPapeis( A, Papel, LPapeis),!.

% por default, nao muda o papel
%
revisaPapelPorProcessos(A, Papel, LPapeis) :-
     write('Nao foi possível revisar o papel de '), write(A), nl,
     member(papel(A,Papel,_),LPapeis).

revisaPorPertinencia( Ag, LTraces, Papel, LPapeis) :-
     papel(Papel, PIs),
     verifPIs(PIs, LTraces, LPapeis).

revisaPorConstrucao( Ag, LTraces, indeterminado, LPapeis) :-
     transLTracesToLProcs( LTraces,LProcs),
     impLProcsRel(Ag, LProcs).

trocaAgPorPapel([],[],_) :- !.
trocaAgPorPapel([t(Ag,Proc)|R],[t(Papel,Proc)|RP],LPapeis) :-
     member(papel(Ag,Papel,_),LPapeis),
     trocaAgPorPapel(R,RP,LPapeis), !.
% Se nao existe papel para algum ag. ele eh tirado da lista
%
trocaAgPorPapel([_|R], RP, LPapeis) :-
     trocaAgPorPapel(R,RP,LPapeis).

comparaPapeis(A,Papel,LPapeis) :-
     member(papel(A,Papel,_),LPapeis),
     write('O Agente '), write(A),
     write(' continua com papel de '), write(Papel), nl, nl, !.

comparaPapeis(A,Papel,LPapeis) :-
     write('O Agente '), write(A),
     write(' mudou para o papel de '), write(Papel), nl, nl, !.

% **
% **  Predicados para pre-processar a lista de comunicacoes
% **

% Monta uma lista com todas as comunicacoes de um agente
%
comunicacoesA(A,L) :- recall(mensagens,LM),
                      filtraParaAgente(A,LM,L).
```

```
filtraParaAgente(A,[],[]) :- !.
filtraParaAgente(A,[mensagem(T,Ay,M,Dy)|R],[m(T,Ay,Mf,Dy)|RA]) :-
     (Ay == A; Dy == A),
     M \= request(role),
     M \= reply(role(_)),
     transfMem(M,Mf),
     filtraParaAgente(A,R,RA), !.
filtraParaAgente(A,[_|R],RA) :-
     filtraParaAgente(A,R,RA).

transfMem(inform(peca(_)), inform(peca(_))) :- !.
transfMem(reply(peca(_)) , reply(peca(_))) :- !.
transfMem(M,M).

% Transforma uma lista de mensagens de um agente em uma
% lista de acoes deste agente
%
transformaEmAcoes(_,[],[]).
transformaEmAcoes(A,[m(T,A,M,R)|RCom],[envia_mensagem(M,R)|Resto]) :-
     transformaEmAcoes(A,RCom,Resto), !.
transformaEmAcoes(A,[m(T,E,M,A)|RCom],[recebe_mensagem(M,E)|Resto]) :-
     transformaEmAcoes(A,RCom,Resto), !.

% Transforma uma lista acoes em traces
%
transformaEmTrace([],[],null).
transformaEmTrace(LAcoes,Trace,Ag) :-
     recall(agentes_activos,AA),
     on(Ag,AA),
     procOcorParaAg(LAcoes,Ag,Trace).

procOcorParaAg([],_,[]) :- !.
procOcorParaAg([Acao|R],Ag,[Acao2|R2]) :-
     Acao =.. [TpAcao,Mem,Ag],
     Acao2 =.. [TpAcao,Mem],
     procOcorParaAg(R,Ag,R2), !.
procOcorParaAg([_|R],Ag,R2) :- procOcorParaAg(R,Ag,R2).

% Trasforma uma lista de traces em uma lista de processos
%
transLTracesToLProcs([],[]) :- !.
transLTracesToLProcs([t(Ag,Trace)|LTraces],[pi(Ag,Proc)|LProc]) :-
     Trace \= [],
     transTraceProc(Trace,Proc), !,
     transLTracesToLProcs(LTraces,LProc), !.
transLTracesToLProcs([_|LTraces],LProc) :-
     transLTracesToLProcs(LTraces,LProc).

% Imprime processos identificados
%
impLProcsRel(_,[]) :- !.
impLProcsRel(A,[pi(AgRelacionado, Proc) | LProcs]) :-
     nl, write('Processo de '), write(A),
     write(' relacionado ao agente '), write(AgRelacionado),
     impProc(Proc), nl, impLProcsRel(A,LProcs).

% **
% **    transTraceProc: transforma um Trace num Processo
% **                    transTraceCiclo(+Trace, +-Proc)
% **
transTraceProc(Trace, Processo) :-
        transTraceProc(Trace, 'MAIN', 0, Processo).

transTraceProc(Trace, NomeProc, Nivel, Processo) :-
        transTraceProcCiclo(Trace, NomeProc, Nivel, Processo) ;
        transTraceProcSeq(  Trace, PSeq),
```

```
        Processo = [p(NomeProc,PSeq)].

% Regras para ciclos
%
transTraceProcCiclo(Trace, NomeProc, Nivel, Processo) :-
        idCiclo(Trace,LCiclos,TIni,[]),
        LCiclos \= [],

        % Transforma parte inicial
        transTraceProcSeq(TIni,  PIni),
        substituiProc(PIni,stop,skip,Ini), %write('Ini = '), write(Ini),
nl,

        % Transforma os Traces em Processos
        transLTraceLProc( LCiclos, LProcsSeq),

        % Procura escolha
        idEscolha(LProcsSeq,LProcsSeq1),

        % Transforma os Processos em Ciclos
        transLProcLCiclos(LProcsSeq1, 'CICLO', Nivel, LProcs),

        inc(Nivel,Nivel1),
        transLProcChamaCiclos(LProcsSeq1, Nivel1, ChamaCiclos),

        normalizaProc([p(NomeProc,Ini seq p('CICLO')),
                       p('CICLO', ChamaCiclos seq p('CICLO')) |
                        LProcs
                       ], Processo).

% regra para lista de traces de ciclos
%
transLTraceLProc([],[]) :- !.
transLTraceLProc([TCiclo|T],[P|T2]) :-
        transTraceProcSeq(TCiclo, P),
        transLTraceLProc(T,T2).

transLProcLCiclos([],_,_,[]) :- !.
transLProcLCiclos([PCiclo|T],Nome,Nivel,[Proc|T2]) :-
        inc(Nivel,Nivel1),
        substituiProc(PCiclo,stop,skip,Ciclo),
        Proc = p(Nome(Nivel1), Ciclo),
        transLProcLCiclos(T,Nome,Nivel1,T2).

transLProcChamaCiclos([_]  , Nivel, p('CICLO'(Nivel))) :- !.
transLProcChamaCiclos([_|T], Nivel, p('CICLO'(Nivel)) ou T2) :-
        inc(Nivel,Nivel1),
        transLProcChamaCiclos(T,Nivel1,T2).

% regras para sequencia
%
transTraceProcSeq([],p(stop))  :- !.
transTraceProcSeq([C|R],p(prefix,C -> R2)) :- transTraceProcSeq(R,R2), !.

%
%       idEscolha: identifica escolha num conjunto de processos
%
%       formato: idEscolha(+ListProcs, +-ListProcs)
%

idEscolha([C1|L],LEscolha) :-
        deleteEle(C2,L,LsemC2),
        not(var(C2)),
        prefixProc(C1,C2,Prefix),
        Prefix \= p(stop),

        % Pega a parte final de C1
```

```
        deleteProc(C1,Prefix,FinalC1),
        % Pega a parte final de C2
        deleteProc(C2,Prefix,FinalC2),

        appendProc(Prefix,(FinalC1 ou FinalC2), P),
        idEscolha([P|LsemC2], LEscolha).

idEscolha(L,L).



% Deixa o processo numa forma normal, tira coisas inuteis
%
% Trata: lista de processos
normalizaProc([],[]) :- !.
normalizaProc([P|RP],[Pf|RPf]) :-
        normalizaProc(P,Pf),
        normalizaProc(RP,RPf).

% (p(skip) ; Proc) ==> Proc
normalizaProc(p(Nome,p(skip) seq Proc), p(Nome, ProcF)) :-
        normalizaProc(Proc,ProcF), !.

% Arruma sequencias
normalizaProc(P1 seq P2, P1f seq P2f) :-
        normalizaProc(P1,P1f),
        normalizaProc(P2,P2f), !.

normalizaProc(p(Nome,P), p(Nome,Pf)) :-
        normalizaProc(P,Pf), !.

normalizaProc(P,P).


% **
% **    idCiclo: identifica ciclo em traces bem bomo
% **             partes antes de depois do ciclo
% **
% **    formato: idCiclo(+Trace, +-ListaCiclos, +-Inicio, +-RestoTrace)
% **     RestoTrace e' aquilo que nao se achou ciclos. Ideal e'que seja []


%       Caso final
%
idCiclo([],[],[],[]).

%       Tenta pegar o ciclo do inicio do trace
%
idCiclo(Trace,[CicloR|OutrosCiclos],[],Fim) :-
        length(Trace,TamTrace),
        Meio is int(TamTrace / 2) - 1, Meio > 0,
        numeros(Meio,1,-1,N),
        subTrace(Trace,0,N,Ciclo),

        % Procura outra ocorrencia de Ciclo no resto do trace
        inc(N,N1),
        dec(TamTrace,TamTrace1),
        subTrace(Trace,N1,TamTrace1,RestoTrace),
        list_search(Ciclo,RestoTrace,_),

        % Tenta um trace menor no ciclo encontrado
        idCiclo(Ciclo,CicloIn,Ini,[]),
        ( Ciclo  = Ini, CicloR = Ciclo % nao tem ciclo menor
        ; first(CicloIn,CicloR)        % encontrou ciclo menor
        ),

        % Tira outras ocorrencias do ciclo no trace
```

```
        delsublist(Ciclo,RestoTrace,TraceTmp1),

        % Se a parte final do trace eh parte do ciclo, tirar.
        ( list_search(TraceTmp1,Ciclo,0), TraceTmp2 = []
        ; TraceTmp2 = TraceTmp1),

        % Procura outros ciclos no restante do trace
        idCiclo(TraceTmp2, OutrosCiclos, Fim, _).


% Tentar pegar o ciclo nao no comeco, no meio do trace
%
idCiclo([C|Trace],Ciclo,[C|R],Fim) :-
        idCiclo(Trace,Ciclo,R,Fim).

subTrace(Trace,I,J,[]) :-
        length(Trace,Tam),(I >= Tam ; I > J), !.

subTrace(Trace,I,I,[C]) :-
        nessimo(I,Trace,C), !.
subTrace(Trace,I,J,[C|R]) :-
        length(Trace,Tam),I < Tam,
        nessimo(I,Trace,C),
        I2 is I + 1,
        subTrace(Trace,I2,J,R).

% **
% ** Verifica se para todos os pis do papel existe um trace
% **
% ** (usado para comparar traces com processos de interacao de um papel)
% **

% Formato: verifPIs(Processos de Interacao, Traces, Papeis)
%
verifPIs([],_,_) :- !.
verifPIs([PI|RPI],LTraces,LPapeis) :-
     verifPI(PI,LTraces,LPapeis),
     verifPIs(RPI, LTraces,LPapeis), !.

% Verifica se existe um trace para um pi
%
verifPI(pi(PapelRel,Procs),[t(Ag,Trace)|RTrace],LPapeis) :-
     Trace \== [],
     verifTrace( Trace, [p('MAIN')|Procs], 'CICLO', Ocorr),
     Ocorr1 is Ocorr - 1, write('Nro Ciclos '), write(Ocorr1),nl,
     Ocorr1 > 1,
     verifPapelRel(PapelRel,Ag,LPapeis),
     !.
verifPI(PI,[_|RTrace],LPapeis) :-
     verifPI( PI, RTrace,LPapeis).

% Verifica se o Ag possui PapelRel na LPapeis
% (nao faz o teste se lista papeis e vazia)
%
verifPapelRel(PapelRel, Ag, []) :- !.
verifPapelRel(PapelRel, Ag, LPapeis) :-
     member_check(papel(Ag,PapelRel,_), LPapeis),
     write('Papel relacionado com PI '), write(PapelRel),
     write(', fechou com o agente '), write(Ag), write('.'),nl,!.

% **
% **    verifTrace: Verifica se um trace pertence a um processo
% **    e conta o nro de ocorrencia do processo P
% **    formato: verifTrace(+Trace, +ListaProc, +P, -Nro)
% **
verifTrace(Trace,[ProcAtu|RestoProcs],PConta,X) :-
        verifTrace0(Trace,[],ProcAtu,[ProcAtu|RestoProcs],PConta,0,X), !.
```

```
% Somente tem um processo
%
verifTrace(Trace,ProcAtu,PConta,X) :-
        verifTrace0(Trace,[],ProcAtu,[ProcAtu],PConta,0,X).

% verifTrace0(E,_,PA, _) :- write(E),write(PA),nl, fail.

% Verifica prefix
%
verifTrace0([],[],p(stop), _, _,X,X) :- write(stop),nl,!.
verifTrace0(Ev,Ev,p(skip), _, _,X,X) :- !.
verifTrace0([Evento|RestoEventos], Re2, p(prefix,Evento -> Proc1),
       Procs, PConta,Xin,Xout) :-
       verifTrace0(RestoEventos, Re2, Proc1,Procs,PConta,Xin,Xout), !.


% Sequencia
%
verifTrace0(Eventos, Re2, P1 seq P2, Procs, PConta,Xin,Xout) :-
        verifTrace0(Eventos, Re, P1, Procs, PConta,Xin,X1),
        Eventos \= Re,
        verifTrace0(Re,      Re2,P2, Procs, PConta,X1,Xout), !.
verifTrace0(Eventos, Re2, p(Nome,P1 seq P2), Procs, PConta,Xin,Xout) :-
        verifTrace0(Eventos, Re2,P1 seq P2, Procs, PConta,Xin,Xout), !.

% Alternativa
%
verifTrace0(Eventos, Re2, Proc1 ou Proc2, Procs, PConta,Xin,Xout) :-
        % tenta no primeiro processo
        verifTrace0(Eventos, Re, Proc1, Procs, PConta,Xin,X1),
        % tenta no segundo processo
        verifTrace0(Re, Re2, Proc2, Procs, PConta,X1,Xout), !.

verifTrace0(Eventos, Re2, p(Nome, P1 ou P2), Procs, PConta,Xin,Xout) :-
        verifTrace0(Eventos, Re2, P1 ou P2,  Procs, PConta,Xin,Xout), !.

% Chamada de outro processo
%
verifTrace0(Eventos, Re2, p(Nome), Procs, PConta,Xin,Xout) :-
        Nome \= stop, Nome \= skip,
        member(p(Nome,ProcAtu), Procs),
( (Nome == PConta, X1 is Xin + 1)
 ; X1 is Xin),

        verifTrace0(Eventos, Re2, ProcAtu, Procs, PConta,X1,Xout), !.

verifTrace0(Eventos, Eventos, _, _, PConta,X,X).
```

5.1.3.2 Código para identificação de papéis por indução de intenções em planos[29]

```
identificaPapelPorPlanos(A,Papel) :-
     comunicacoesA(A,LCom),
     transformaEmAcoes(A,LCom,LAcoes), %writel(LAcoes),
     induzObservacoes(LAcoes, [], LEstados, [], AcoesPret),
     write('Acoes predentidas:'), write(AcoesPret), nl,

     papelPlanos(Papel, AcoesPapel),
     % Procura duas ocorrencias das Acoes Papel
```

29 Foi tirado do código original a relação das ações, porque estas já estão colocadas na seção 3.4.2.2.

```
     subset( AcoesPapel, AcoesPret),
     delsubset( AcoesPapel, AcoesPret, AP2), !,
     subset( AcoesPapel, AP2),

     escreve(['Papel do agente ',A, ' , ', Papel, '. ']),
     escreve(['Identificada(s) intencao/oes ciclicas de', AcoesPapel]),
     !.

revisaPapelPorPlanos(A, NPapel, LPapeis) :-
     identificaPapelPorPlanos(A, NPapel).
revisaPapelPorPlanos(A, Papel, LPapeis) :-
     write('Nao foi possível revisar o papel de '), write(A), nl,
     member(papel(A,Papel,_),LPapeis).


induzObservacoes([],L,L,A,A).
induzObservacoes([C|R], LIn, LOut, AcoesIn, AcoesOut) :-
     % Regra 2
     % Pega a acao realizada
     %
     regra( r2, C, CdAcao, Acao),   %write(Acao),nl,

     % Regra r1 inv
     % Verifica o estado anterior desta acao
     %
     regra( r1_Inv, CdAcao, EstadoI), %write(EstadoI),nl,
     incluiPreEstado(EstadoI,LIn,L1),

     % Regra 3
     % Coloca o estado efeito da acao
     %
     regra(r3, CdAcao, EstadoG), %write(EstadoG),nl,

     % Regra 1
     % Coloca acoes possiveis, dado este ultimo estado
     %
     regra( r1, EstadoG, AcoesPossiveis), %write(AcoesPossiveis),nl,
     incluiAcoesPossiveis( AcoesIn, Acao, AcoesPossiveis, LAcoes),

     append(L1, [EstadoG], LVai),   %write(LVai),nl,

     induzObservacoes(R, LVai, LOut, LAcoes, AcoesOut), !.

induzObservacoes([C|R], LIn, LOut, AcaoIn, AcaoOut) :-
     induzObservacoes(R, LIn, LOut, AcaoIn, AcaoOut), !.


incluiPreEstado(E,LIn,LOut) :-
     not(last(LIn,E)),
     append(LIn,[E],LOut), !.
incluiPreEstado(E,L,L).

% Caso em que a lista de acoes possiveis inclui a ultima acao
%
incluiAcoesPossiveis( AcoesIn, Acao, AcoesPossiveis, LAcoes) :-
     not(last(AcoesIn,Acao)),
     deleteEle(Acao, AcoesPossiveis, AP2),
     append(AcoesIn, [Acao|AP2], LAcoes), !.
% Caso normal
incluiAcoesPossiveis( AcoesIn, Acao, AcoesPossiveis, LAcoes) :-
     not(last(AcoesIn,Acao)),
     append(AcoesIn, [Acao|AcoesPossiveis], LAcoes), !.
incluiAcoesPossiveis( AcoesIn, Acao, AcoesPossiveis, LAcoes) :-
     append(AcoesIn, AcoesPossiveis, LAcoes), !.

% Retorna as acoes possiveis, dada uma pre-condicao
%
```

```
regra(r1, Pre, LA) :-
     findall(Acao,
            (acao(CdAcao, Acao, PreAcao,_,_) , member(Pre,PreAcao)),
             LAcao),
     remove_duplicates(LAcao, LA), !.

% Retorna o estado pre-condicao da acao (inv de r1)
%
regra(r1_Inv, CdAcao, estado(X)) :-
     acao( CdAcao,_,Pre,_,_),
     member(estado(X),Pre), !.

% Se uma acao observada faz parte do corpo de uma acao
% entao o agente esta fazendo aquela acao
%
regra(r2, Observacao, CdAcao, Acao) :-
     acao(CdAcao,Acao,PreCond,Efeito,Corpo),
     member(Observacao,Corpo), !.

% Retorna o estado Efeito da acao
%
regra(r3, Acao, estado(X)) :-
     acao(Acao,_,_,Efeito,_),
     member(estado(X),Efeito), !.


% Relacao de _____papeis____ da sociedade
% formato: papel(Papel, intencoes).
% Obs.: a ordem das clausulas deve ser: do maior para o menor (quanto
%       ao nro de intencoes)
%
papelPlanos(transformador,[consumir,produzir]).
papelPlanos(intermediario,[enviar_peca,receber_peca]).
papelPlanos(consumidor,   [consumir]).
papelPlanos(produtor,     [produzir]).
```

5.2 Acompanhamento de simulações

5.2.1 Caso 1

Na seguinte simulação participam dois agentes Produtor (Rocha e Rosa), uma agente Intermediária (Renata), um agente Consumidor (Jomi), um agente Transformador (Edson) e dois agentes Inspetor (Jr, por indução de intenções, e Dalton, por reconhecimento de processo). As linhas da simulação apresentam a seguinte forma:

<Nome do agente> <Tipo de agente> <Mensagem>

onde <Tipo de agente> segue a correspondência:

+++	Produtor
::::	Consumidor
***	Inspetor
....	Intermediário
>>>	Transformador

Jr. *** Papeis para: Jomi (planos)
Jr. *** Papeis para: Rocha (planos)
Jr. *** Papeis para: Rosa (planos)
Jr. *** Papeis para: Renata (planos)
Jr. *** Papeis para: Edson (planos)
Jr. *** Papeis para: Dalton (planos)
Jr. *** Papeis Identificados:
papel(Jr., Inspetor, 0),

Dalton *** Papeis para: Jomi (por processos)
Dalton *** Papeis para: Rocha (por processos)
Dalton *** Papeis para: Rosa (por processos)
Dalton *** Papeis para: Renata (por processos)
Dalton *** Papeis para: Edson (por processos)
Dalton *** Papeis para: Jr. (por processos)
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, 0), [30]

Edson >>> Mandou pedido de Identificação [31]

---

30 Inicialmente não há nenhuma ação a ser observada.

31 Aqui os agentes iniciam o período de apresentação.

Renata ..... Recebida mensagem request(role) de Edson
Renata ..... Resposta de apresentação para Edson
Rosa ++++ Recebida mensagem request(role) de Edson
Rosa ++++ Mandou pedido de Identificação
Rocha ++++ Recebida mensagem request(role) de Edson
Rocha ++++ Mandou pedido de Identificação
Jomi :::: Mandou pedido de Identificação
Rosa ++++ Recebida mensagem request(role) de Rosa
Rocha ++++ Recebida mensagem request(role) de Rosa
Edson >>> Identificado papel de Renata : Intermediário (por apresentação)
Edson >>> Mandou pedido de peca
Renata ..... Recebida mensagem request(role) de Rosa
Renata ..... Resposta de apresentação para Rosa
Rosa ++++ Recebida mensagem request(role) de Rocha
Rocha ++++ Recebida mensagem request(role) de Rocha
Edson >>> Identificado papel de Rosa : Produtor (por apresentação)
Rosa ++++ Recebida mensagem request(role) de Jomi
Rocha ++++ Recebida mensagem request(role) de Jomi
Renata ..... Recebida mensagem request(role) de Rocha
Renata ..... Resposta de apresentação para Rocha
Rosa ++++ Recebida mensagem reply(role(Produtor)) de Rosa
Rosa ++++ Identificado papel de Rosa : Produtor (por apresentação)
Rocha ++++ Recebida mensagem reply(role(Produtor)) de Rosa
Rocha ++++ Identificado papel de Rosa : Produtor (por apresentação)
Jomi :::: Identificado papel de Rosa : Produtor (por apresentação)
Edson >>> Identificado papel de Rocha : Produtor (por apresentação)
Rosa ++++ Recebida mensagem reply(role(Produtor)) de Rocha
Rosa ++++ Identificado papel de Rocha : Produtor (por apresentação)
Rocha ++++ Recebida mensagem reply(role(Produtor)) de Rocha
Rocha ++++ Identificado papel de Rocha : Produtor (por apresentação)
Jomi :::: Identificado papel de Rocha : Produtor (por apresentação)
Renata ..... Recebida mensagem request(role) de Jomi
Renata ..... Resposta de apresentação para Jomi
Rosa ++++ Recebida mensagem reply(role(Consumidor)) de Jomi
Rosa ++++ Identificado papel de Jomi : Consumidor (por apresentação)
Rocha ++++ Recebida mensagem reply(role(Consumidor)) de Jomi
Rocha ++++ Identificado papel de Jomi : Consumidor (por apresentação)
Jomi :::: Identificado papel de Jomi : Consumidor (por apresentação)
Edson >>> Identificado papel de Jomi : Consumidor (por apresentação)
Rosa ++++ Recebida mensagem reply(role(Intermediário)) de Renata
Rosa ++++ Identificado papel de Renata : Intermediário (por apresentação)
Rosa ++++ Iniciou produção da peça 1
Rocha ++++ Identificado papel de Renata : Intermediário (por apresentação) [32]
Rocha ++++ Iniciou produção da peça 1
Jomi :::: Identificado papel de Renata : Intermediário (por apresentação) [33]
Jomi :::: Mandou pedido
Renata ..... Recebida mensagem request(peca) de Edson

---

32 A partir deste momento o agente Rocha pode iniciar a produção da primeira peça, já que ele conhece um Intermediário para mandar a peça.

33 Neste momento o agente Jomi identificou um agente com papel Intermediário, logo pode pedir-lhe peças, saindo do estado de apresentação.

Renata ..... Lista de pendencias aumentada para: [Edson]
Rosa ++++ Identificado papel de Edson : Transformador (por apresentação)
Rosa ++++ Passo de produção 3
Rocha ++++ Identificado papel de Edson : Transformador (por apresentação)
Rocha ++++ Passo de produção 3
Jomi :::: Identificado papel de Edson : Transformador (por apresentação)
Rosa ++++ Passo de produção 2
Rocha ++++ Passo de produção 2
Jr. *** Papeis para: Jomi (planos)
Acoes predentidas:[pedir_peca, receber_peca]
Jr. *** Papeis para: Rocha (planos)
Jr. *** Papeis para: Rosa (planos)
Jr. *** Papeis para: Renata (planos)
Jr. *** Papeis para: Edson (planos)
Acoes predentidas:[pedir_peca, receber_peca]
Jr. *** Papeis para: Dalton (planos)
Jr. *** Papeis Identificados:
papel(Jr., Inspetor, -1),[34]

Dalton *** Papeis para: Jomi (por processos)
Processo de Jomi relacionado ao agente Renata
MAIN = envia_mensagem(request(peca)) ->
STOP. [35]

Dalton *** Papeis para: Rocha (por processos)
Dalton *** Papeis para: Rosa (por processos)
Dalton *** Papeis para: Renata (por processos)
Processo de Renata relacionado ao agente Jomi
MAIN = recebe_mensagem(request(peca)) ->
STOP.

Processo de Renata relacionado ao agente Edson
MAIN = recebe_mensagem(request(peca)) ->
STOP.

Dalton *** Papeis para: Edson (por processos)
Processo de Edson relacionado ao agente Renata
MAIN = envia_mensagem(request(peca)) ->
STOP.

Dalton *** Papeis para: Jr. (por processos)
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -1),

Renata ..... Recebida mensagem request(peca) de Jomi
Renata ..... Lista de pendencias aumentada para: [Edson, Jomi]

---

34 Na sua segunda tentativa de identificar papéis, o agente Jr. ainda não consegue identifica-los, pois não existem observações suficientes.

35 Segunda tentativa do agente Dalton. Como não foi possível verificar se o *trace* do agente Jomi até o momento pertence a algum modelo, pois não foi observado mais de uma ocorrência de ciclo, o Inspetor construiu um processo para o agente Jomi.

```
Rosa ++++  Passo de produção 1
Rocha ++++  Passo de produção 1
Rosa ++++  Passo de produção 0
Rocha ++++  Passo de produção 0
Rosa ++++  Mandou pedido de espaco para Renata
Rocha ++++  Mandou pedido de espaco para Renata
Renata .....  Recebida mensagem request(espaco) de Rosa
Renata .....  Confirmado espaco para Rosa Reservas= 1
Rosa ++++  Recebida mensagem reply(espaco_ok) de Renata
Rosa ++++  Mandou peça 1 para Renata
Renata .....  Recebida mensagem request(espaco) de Rocha
Renata .....  Confirmado espaco para Rocha Reservas= 2
Rosa ++++  Iniciou produção da peça 2
Rocha ++++  Recebida mensagem reply(espaco_ok) de Renata
Rosa ++++  Passo de produção 3
Rocha ++++  Mandou peça 1 para Renata
Renata .....  Recebida mensagem inform(peca(1)) de Rosa
Renata .....  Estoque aumentado para:  1 [peca(1)]
Rosa ++++  Passo de produção 2
Rocha ++++  Iniciou produção da peça 2
Renata .....  Recebida mensagem request(peca) de Edson
Renata .....  Estoque diminuido para:  0 []
Renata .....  Enviado  peca(1)  para: Edson
Renata .....  Lista pendencias diminuida para:  [Jomi]
Rosa ++++  Passo de produção 1
Rocha ++++  Passo de produção 3
Jr. ***  Papeis para:  Jomi  (planos)
Acoes predentidas:[pedir_peca, receber_peca]
Jr. ***  Papeis para:  Rocha  (planos)
Acoes predentidas:[pedir_espaco, receber_espaco, enviar, produzir]
Jr. ***  Papeis para:  Rosa  (planos)
Acoes predentidas:[pedir_espaco, receber_espaco, enviar, produzir]
Jr. ***  Papeis para:  Renata  (planos)
Acoes predentidas:[receber_peca, receber_peca, enviar_peca, receber_peca]
Jr. ***  Papeis para:  Edson  (planos)
Acoes predentidas:[pedir_peca, receber_peca, consumir]
Jr. ***  Papeis para:  Dalton  (planos)
Jr. ***  Papeis Identificados:
papel(Jr., Inspetor, -2),

Dalton ***  Papeis para:  Jomi  (por processos)
Processo de Jomi relacionado ao agente Renata
MAIN = envia_mensagem(request(peca)) ->
STOP.

Dalton ***  Papeis para:  Rocha  (por processos)
Processo de Rocha relacionado ao agente Renata
MAIN = envia_mensagem(request(espaco)) ->
recebe_mensagem(reply(espaco_ok)) ->
envia_mensagem(inform(peca(_5489))) ->
STOP.

Dalton ***  Papeis para:  Rosa  (por processos)
```

Processo de Rosa relacionado ao agente Renata
MAIN = envia_mensagem(request(espaco)) ->
recebe_mensagem(reply(espaco_ok)) ->
envia_mensagem(inform(peca(_5915))) ->
STOP.

Dalton *** Papeis para: Renata (por processos)
Processo de Renata relacionado ao agente Jomi
MAIN = recebe_mensagem(request(peca)) ->
STOP.

Processo de Renata relacionado ao agente Rocha
MAIN = recebe_mensagem(request(espaco)) ->
envia_mensagem(reply(espaco_ok)) ->
recebe_mensagem(inform(peca(_6595))) ->
STOP.

Processo de Renata relacionado ao agente Rosa
MAIN = recebe_mensagem(request(espaco)) ->
envia_mensagem(reply(espaco_ok)) ->
recebe_mensagem(inform(peca(_6628))) ->
STOP.

Processo de Renata relacionado ao agente Edson
MAIN = recebe_mensagem(request(peca)) ->
envia_mensagem(reply(peca(_6659))) ->
STOP.

Dalton *** Papeis para: Edson (por processos)
Processo de Edson relacionado ao agente Renata
MAIN = envia_mensagem(request(peca)) ->
recebe_mensagem(reply(peca(_6718))) ->
STOP.

Dalton *** Papeis para: Jr. (por processos)
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -2),

Edson >>> Recebeu produto: peca(1)
Edson >>> Mandou pedido de peca
Renata ..... Recebida mensagem inform(peca(1)) de Rocha
Renata ..... Estoque aumentado para: 1 [peca(1)]
Rosa ++++ Passo de produção 0
Rosa ++++ Mandou pedido de espaco para Renata
Rocha ++++ Passo de produção 2
Renata ..... Recebida mensagem request(peca) de Jomi
Renata ..... Estoque diminuido para: 0 []
Renata ..... Enviado peca(1) para: Jomi
Renata ..... Lista pendencias diminuida para: []
Rocha ++++ Passo de produção 1
Jomi :::: Recebeu produto: peca(1)
Jomi :::: Tempo restante de Consumo = 3
Renata ..... Recebida mensagem request(peca) de Edson

Renata ..... Lista de pendencias aumentada para: [Edson]
Rocha ++++ Passo de produção 0
Rocha ++++ Mandou pedido de espaco para Renata
Jomi :::: Tempo restante de Consumo = 2
Jomi :::: Tempo restante de Consumo = 1
Renata ..... Recebida mensagem request(espaco) de Rosa
Renata ..... Confirmado espaco para Rosa Reservas= 1
Rosa ++++ Recebida mensagem reply(espaco_ok) de Renata
Jomi :::: Tempo restante de Consumo = 0
Rosa ++++ Mandou peça 2 para Renata
Jomi :::: Pecas consumidas= 1
Jomi :::: Tempo restante de Consumo = terminou
Renata ..... Recebida mensagem request(espaco) de Rocha
Renata ..... Confirmado espaco para Rocha Reservas= 2
Rosa ++++ Iniciou produção da peça 3
Rocha ++++ Recebida mensagem reply(espaco_ok) de Renata
Jomi :::: Mandou pedido
Rosa ++++ Passo de produção 3
Rocha ++++ Mandou peça 2 para Renata
Renata ..... Recebida mensagem inform(peca(2)) de Rosa
Renata ..... Estoque aumentado para: 1 [peca(2)]
Rosa ++++ Passo de produção 2
Rocha ++++ Iniciou produção da peça 3
Renata ..... Recebida mensagem request(peca) de Edson
Renata ..... Estoque diminuido para: 0 []
Renata ..... Enviado peca(2) para: Edson
Renata ..... Lista pendencias diminuida para: []
Rosa ++++ Passo de produção 1
Rocha ++++ Passo de produção 3
Jr. *** Papeis para: Jomi (planos)
Acoes predentidas:[pedir_peca, receber_peca, consumir, pedir_peca, receber_peca]
Jr. *** Papeis para: Rocha (planos)
Acoes predentidas:[pedir_espaco, receber_espaco, enviar, **produzir**, pedir_espaco, receber_espaco, enviar, **produzir**][36]
Papel do agente Rocha , produtor .
Identificada(s) intencao/oes ciclicas de ' [produzir] '.

Jr. *** Papeis para: Rosa (planos)
Acoes predentidas:[pedir_espaco, receber_espaco, enviar, **produzir**, pedir_espaco, receber_espaco, enviar, **produzir**]
Papel do agente Rosa , produtor .
Identificada(s) intencao/oes ciclicas de ' [produzir] '.

Jr. *** Papeis para: Renata (planos)
Acoes predentidas:[**receber_peca**, **receber_peca**, **enviar_peca**, receber_peca, **enviar_peca**, receber_peca, receber_peca, receber_peca, enviar_peca, receber_peca]
Papel do agente Renata , intermediario .
Identificada(s) intencao/oes ciclicas de ' [enviar_peca, receber_peca] '.

Jr. *** Papeis para: Edson (planos)

---

36 O agente Rocha já intencionou duas vezes produzir, logo, é identificado como Produtor.

Acoes predentidas:[pedir_peca, receber_peca, **consumir**, pedir_peca, receber_peca, **consumir**]
Papel do agente Edson , consumidor .
Identificada(s) intencao/oes ciclicas de ' [consumir] '. [37]

Jr. *** Papeis para: Dalton (planos)
Jr. *** Papeis Identificados:
papel(Jr., Inspetor, -3),
papel(Rocha, produtor, 3),
papel(Rosa, produtor, 3),
papel(Renata, intermediario, 3),
papel(Edson, consumidor, 3), [38]

Dalton *** Papeis para: Jomi (por processos)
Papel de Jomi e' Consumidor .

Dalton *** Papeis para: Rocha (por processos)
Papel de Rocha e' Produtor .

Dalton *** Papeis para: Rosa (por processos)
Papel de Rosa e' Produtor .

Dalton *** Papeis para: Renata (por processos)
Papel de Renata e' Intermediário .

Dalton *** Papeis para: Edson (por processos)
Papel de Edson e' Consumidor .

Dalton *** Papeis para: Jr. (por processos)
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -3),
papel(Jomi, Consumidor, 3),
papel(Rocha, Produtor, 3),
papel(Rosa, Produtor, 3),
papel(Renata, Intermediário, 3),
papel(Edson, Consumidor, 3) [39]

Edson >>> Recebeu produto: peca(2)
Edson >>> Pecas peca(1) e peca(2) transformadas na peca(3)
Renata ..... Recebida mensagem request(peca) de Jomi
Renata ..... Lista de pendencias aumentada para: [Jomi]
Rosa ++++ Passo de produção 0
Rosa ++++ Mandou pedido de espaco para Renata
Rocha ++++ Passo de produção 2
Rocha ++++ Passo de produção 1

---

37 Como o agente Edson intencionou consumir, a princípio, é identificado como Consumidor. Embora, na verdade seja Transformador. Isto é devido ao problema da parada. A revisão do papel de Edson irá resolver este problema.

38 O último número nesta relação agente x papel indica quanto tempo falta para revisar o papel deste agente.

39 O mecanismo por classificação chegou aos mesmos resultados que o por indução.

Edson >>> Mandou pedido de espaco para Renata
Renata ..... Recebida mensagem inform(peca(2)) de Rocha
Renata ..... Estoque aumentado para: 1 [peca(2)]
Rocha ++++ Passo de produção 0
Rocha ++++ Mandou pedido de espaco para Renata
Renata ..... Recebida mensagem request(peca) de Jomi
Renata ..... Estoque diminuido para: 0 []
Renata ..... Enviado peca(2) para: Jomi
Renata ..... Lista pendencias diminuida para: []
Jomi :::: Recebeu produto: peca(2)
Jomi :::: Tempo restante de Consumo = 3
Renata ..... Recebida mensagem request(espaco) de Rosa
Renata ..... Confirmado espaco para Rosa Reservas= 1
Rosa ++++ Recebida mensagem reply(espaco_ok) de Renata
Jomi :::: Tempo restante de Consumo = 2
Rosa ++++ Mandou peça 3 para Renata
Jomi :::: Tempo restante de Consumo = 1
Renata ..... Recebida mensagem request(espaco) de Edson
Renata ..... Confirmado espaco para Edson Reservas= 2
Rosa ++++ Iniciou produção da peça 4
Jomi :::: Tempo restante de Consumo = 0
Edson >>> Mandou peça 3 para Renata
Rosa ++++ Passo de produção 3
Jomi :::: Pecas consumidas= 2
Jomi :::: Tempo restante de Consumo = terminou
Edson >>> Mandou pedido de peca
Renata ..... Recebida mensagem request(espaco) de Rocha
Renata ..... Confirmado espaco para Rocha Reservas= 3
Rosa ++++ Passo de produção 2
Rocha ++++ Recebida mensagem reply(espaco_ok) de Renata
Jomi :::: Mandou pedido
Rosa ++++ Passo de produção 1
Rocha ++++ Mandou peça 3 para Renata
Jr. *** Papeis para: Jomi (planos)
Acoes predentidas:[pedir_peca, receber_peca, **consumir**, pedir_peca, receber_peca, **consumir**,
pedir_peca, receber_peca]
Papel do agente Jomi , consumidor .
Identificada(s) intencao/oes ciclicas de ' [consumir] '.

Jr. *** Papeis para: Dalton (planos)
Jr. *** Papeis Identificados:
papel(Jr., Inspetor, -4),
papel(Rocha, produtor, 2),
papel(Rosa, produtor, 2),
papel(Renata, intermediario, 2),
papel(Edson, consumidor, 2),
papel(Jomi, consumidor, 3),

Dalton *** Papeis para: Jr. (por processos)
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -4),
papel(Jomi, Consumidor, 2),
papel(Rocha, Produtor, 2),

```
papel(Rosa, Produtor, 2),
papel(Renata, Intermediário, 2),
papel(Edson, Consumidor, 2),

Rosa ++++  Passo de produção 0
Rosa ++++  Mandou pedido de espaco para Renata
Rocha ++++  Iniciou produção da peça 4
Rocha ++++  Passo de produção 3
Renata .....  Recebida mensagem request(espaco) de Rosa
Renata .....  Negado espaco para Rosa
Rosa ++++  Recebida mensagem reply(espaco_nok) de Renata
Rosa ++++  Mandou pedido de espaco para Renata
Rocha ++++  Passo de produção 2
Rocha ++++  Passo de produção 1
Renata .....  Recebida mensagem request(espaco) de Rosa
Renata .....  Negado espaco para Rosa
Rosa ++++  Recebida mensagem reply(espaco_nok) de Renata
Rosa ++++  Mandou pedido de espaco para Renata
Rocha ++++  Passo de produção 0
Rocha ++++  Mandou pedido de espaco para Renata
Renata .....  Recebida mensagem request(espaco) de Rosa
Renata .....  Negado espaco para Rosa
Rosa ++++  Recebida mensagem reply(espaco_nok) de Renata
Rosa ++++  Mandou pedido de espaco para Renata
Renata .....  Recebida mensagem request(espaco) de Rocha
Renata .....  Negado espaco para Rocha
Rocha ++++  Recebida mensagem reply(espaco_nok) de Renata
Rocha ++++  Mandou pedido de espaco para Renata
Jr. ***  Papeis para:  Dalton  (planos)
Jr. ***  Papeis Identificados:
papel(Jr., Inspetor, -5),
papel(Rocha, produtor, 1),
papel(Rosa, produtor, 1),
papel(Renata, intermediario, 1),
papel(Edson, consumidor, 1),
papel(Jomi, consumidor, 2),

Jr. *** Revisando o papel de  Rocha
Papel do agente  Rocha  ,  produtor .
Identificada(s) intencao/oes ciclicas de ' [produzir] '.

Jr. *** Revisando o papel de  Rosa
Papel do agente  Rosa  ,  produtor .
Identificada(s) intencao/oes ciclicas de ' [produzir] '.

Jr. *** Revisando o papel de  Renata
Papel do agente  Renata  ,  intermediario .
Identificada(s) intencao/oes ciclicas de ' [enviar_peca, receber_peca] '.

Jr. *** Revisando o papel de  Edson
Acoes predentidas:[pedir_peca, receber_peca, consumir, pedir_peca, receber_peca, consumir,
                   pedir_espaco, receber_espaco, enviar, produzir, pedir_peca, receber_peca]
Nao foi possível revisar o papel de Edson
```

Dalton *** Papeis para: Jr. (por processos)
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -5),
papel(Jomi, Consumidor, 1),
papel(Rocha, Produtor, 1),
papel(Rosa, Produtor, 1),
papel(Renata, Intermediário, 1),
papel(Edson, Consumidor, 1),

Dalton *** Revisando o papel de Jomi [40]
Papel relacionado com PI **Intermediário**, fechou com o agente Renata.
O Agente Jomi continua com papel de Consumidor

Dalton *** Revisando o papel de Rocha
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Rocha continua com papel de Produtor

Dalton *** Revisando o papel de Rosa
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Rosa continua com papel de Produtor

Dalton *** Revisando o papel de Renata
Papel relacionado com PI Consumidor, fechou com o agente Jomi.
Papel relacionado com PI Consumidor, fechou com o agente Jomi.
Papel relacionado com PI Produtor, fechou com o agente Rocha.
O Agente Renata continua com papel de Intermediário

Dalton *** Revisando o papel de Edson
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Edson mudou para o papel de Transformador [41]

Renata ..... Recebida mensagem request(espaco) de Rosa
Renata ..... Negado espaco para Rosa

{ continuação do funcionamento normal dos agentes }

Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -8),
papel(Jomi, Consumidor, 1),
papel(Rocha, Produtor, 1),
papel(Rosa, Produtor, 1),
papel(Renata, Intermediário, 1),
papel(Edson, Transformador, 1),

Dalton *** Revisando o papel de Jomi
Nro Ciclos 3
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Jomi continua com papel de Consumidor

---

40 O mecanismo por classificação, na revisão, considera o papel dos agentes relacionados ao agente alvo.

41 Nesta revisão do papel de Edson já foi possível identificar seu papel correto.

Dalton *** Revisando o papel de Rocha
Nro Ciclos 11
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Rocha continua com papel de Produtor

Dalton *** Revisando o papel de Rosa
Nro Ciclos 14
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Rosa continua com papel de Produtor

Dalton *** Revisando o papel de Renata
Nro Ciclos 3
Papel relacionado com PI Consumidor, fechou com o agente Jomi.
Nro Ciclos 3
Papel relacionado com PI Consumidor, fechou com o agente Jomi.
Nro Ciclos 11
Papel relacionado com PI Produtor, fechou com o agente Rocha.
O Agente Renata continua com papel de Intermediário

Dalton *** Revisando o papel de Edson
Nro Ciclos 2
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Edson continua com papel de Transformador

{ continuação do funcionamento normal dos agentes }

Jr. *** Papeis Identificados:
papel(Jr., Inspetor, -11),
papel(Rocha, produtor, 1),
papel(Rosa, produtor, 1),
papel(Renata, intermediario, 1),
papel(Edson, consumidor, 1),
papel(Jomi, consumidor, 2),

Jr. *** Revisando o papel de Rocha
Papel do agente Rocha , produtor .
Identificada(s) intencao/oes ciclicas de ' [produzir] '.

Jr. *** Revisando o papel de Rosa
Papel do agente Rosa , produtor .
Identificada(s) intencao/oes ciclicas de ' [produzir] '.

Jr. *** Revisando o papel de Renata
Papel do agente Renata , intermediario .
Identificada(s) intencao/oes ciclicas de ' [enviar_peca, receber_peca] '.

Jr. *** Revisando o papel de Edson
Acoes predentidas:[pedir_peca, receber_peca, **consumir**, pedir_peca, receber_peca, **consumir**,
pedir_espaco, receber_espaco, enviar, **produzir**, pedir_peca, receber_peca, consumir,
pedir_peca, receber_peca, consumir, pedir_espaco, receber_espaco, enviar, **produzir**]
Papel do agente Edson , transformador .

Identificada(s) intencao/oes ciclicas de ' [consumir, produzir] '. [42]

5.2.2 Caso 2

Neste caso o agente Dalton (tipo Inspetor) utiliza o mecanismo de identificação de papéis por reconhecimento de processos para identificar papéis de agentes tipo Produtor (Rocha e Rosa), Consumidor (Jomi) e Intermediário (Renata). O que torna esta segunda simulação interessante é o fato do agente Dalton não possuir modelo para um agente Produtor que recebe resposta negativa.

```
Dalton ***  Papeis para:  Jomi  (por processos)
Dalton ***  Papeis para:  Rocha  (por processos)
Dalton ***  Papeis para:  Rosa  (por processos)
Dalton ***  Papeis para:  Renata  (por processos)
Dalton ***  Papeis Identificados:
papel(Dalton, Inspetor, 0),

{ Etapa de apresentação }

Rocha ++++  Identificado papel de Renata : Intermediário (por apresentacao)
Rocha ++++  Iniciou produção da peça 1
Jomi ::::   Mandou pedido
Rosa ++++  Iniciou produção da peça 1
Rocha ++++  Passo de produção 3
Renata .....   Recebida mensagem request(peca) de Jomi
Renata .....   Lista de pendencias aumentada para: [Jomi]
Rosa ++++  Passo de produção 3
Rocha ++++  Passo de produção 2
Rosa ++++  Passo de produção 2
Rocha ++++  Passo de produção 1
Dalton ***  Papeis para:  Jomi  (por processos)
Processo de Jomi relacionado ao agente Renata
MAIN = envia_mensagem(request(peca)) ->
STOP.

Dalton ***  Papeis para:  Rocha  (por processos)
Dalton ***  Papeis para:  Rosa  (por processos)
Dalton ***  Papeis para:  Renata  (por processos)
Processo de Renata relacionado ao agente Jomi
MAIN = recebe_mensagem(request(peca)) ->
STOP.

Dalton ***  Papeis Identificados:
papel(Dalton, Inspetor, -1),
```

---

42 O mecanismo por indução consegui identificar o papel correto de Edson.

```
Rosa ++++  Passo de produção 1
Rocha ++++  Passo de produção 0
Rocha ++++  Mandou pedido de espaco para Renata
Rosa ++++  Passo de produção 0
Renata .....  Recebida mensagem request(espaco) de Rocha
Renata .....  Confirmado espaco para Rocha Reservas= 1
Rosa ++++  Mandou pedido de espaco para Renata
Rocha ++++  Recebida mensagem reply(espaco_ok) de Renata
Rocha ++++  Mandou peça 1 para Renata
Renata .....  Recebida mensagem request(espaco) de Rosa
Renata .....  Confirmado espaco para Rosa Reservas= 2
Rosa ++++  Recebida mensagem reply(espaco_ok) de Renata
Rocha ++++  Iniciou produção da peça 2
Rosa ++++  Mandou peça 1 para Renata
Rocha ++++  Passo de produção 3
Renata .....  Recebida mensagem inform(peca(1)) de Rocha
Renata .....  Estoque aumentado para:  1 [peca(1)]
Rosa ++++  Iniciou produção da peça 2
Rocha ++++  Passo de produção 2
Renata .....  Recebida mensagem request(peca) de Jomi
Renata .....  Estoque diminuido para:  0 []
Renata .....  Enviado  peca(1)  para: Jomi
Renata .....  Lista pendencias diminuida para:  []
Rosa ++++  Passo de produção 3
Rocha ++++  Passo de produção 1
Jomi ::::  Recebeu produto: peca(1)
Jomi ::::  Tempo restante de Consumo =  3
Renata .....  Recebida mensagem inform(peca(1)) de Rosa
Renata .....  Estoque aumentado para:  1 [peca(1)]
Rosa ++++  Passo de produção 2
Rocha ++++  Passo de produção 0
Rocha ++++  Mandou pedido de espaco para Renata
Jomi ::::  Tempo restante de Consumo =  2
Rosa ++++  Passo de produção 1
Jomi ::::  Tempo restante de Consumo =  1

{ Funcionamento normal da sociedade }

Dalton ***  Papeis para:  Jomi  (por processos)
Nro Ciclos 2
Papel de  Jomi  e'  Consumidor .

Dalton ***  Papeis para:  Rosa  (por processos)
Nro Ciclos 3
Papel de  Rosa  e'  Produtor .

Dalton ***  Papeis para:  Renata  (por processos)
Papel de  Renata  e'  Intermediário .

Dalton ***  Papeis Identificados:
papel(Dalton, Inspetor, -3),
papel(Rocha, Produtor, 2),
papel(Jomi, Consumidor, 3),
```

papel(Rosa, Produtor, 3),
papel(Renata, Intermediário, 3),

Renata ..... Recebida mensagem request(espaco) de Rocha
Renata ..... Confirmado espaco para Rocha Reservas= 1
Rocha ++++ Recebida mensagem reply(espaco_ok) de Renata
Jomi :::: Pecas consumidas= 2
Jomi :::: Tempo restante de Consumo = terminou
Rocha ++++ Mandou peça 3 para Renata
Jomi :::: Mandou pedido
Renata ..... Recebida mensagem request(espaco) de Rosa
Renata ..... Negado espaco para Rosa
Rosa ++++ Recebida mensagem reply(espaco_nok) de Renata [43]
Rosa ++++ Mandou pedido de espaco para Renata
Rocha ++++ Iniciou produção da peça 4
Rocha ++++ Passo de produção 3
Renata ..... Recebida mensagem inform(peca(3)) de Rocha
Renata ..... Estoque aumentado para: 3 [peca(2), peca(2), peca(3)]
Rocha ++++ Passo de produção 2
Rocha ++++ Passo de produção 1
Renata ..... Recebida mensagem request(peca) de Jomi
Renata ..... Estoque diminuido para: 2 [peca(2), peca(3)]
Renata ..... Enviado peca(2) para: Jomi
Rocha ++++ Passo de produção 0
Rocha ++++ Mandou pedido de espaco para Renata
Jomi :::: Recebeu produto: peca(2)
Jomi :::: Tempo restante de Consumo = 3
Jomi :::: Tempo restante de Consumo = 2
Renata ..... Recebida mensagem request(espaco) de Rosa
Renata ..... Confirmado espaco para Rosa Reservas= 1
Rosa ++++ Recebida mensagem reply(espaco_ok) de Renata
Jomi :::: Tempo restante de Consumo = 1
Rosa ++++ Mandou peça 3 para Renata
Jomi :::: Tempo restante de Consumo = 0
Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -4),
papel(Rocha, Produtor, 1),
papel(Jomi, Consumidor, 2),
papel(Rosa, Produtor, 2),
papel(Renata, Intermediário, 2),

Dalton *** Revisando o papel de Rocha
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Rocha continua com papel de Produtor

Renata ..... Recebida mensagem request(espaco) de Rocha
Renata ..... Negado espaco para Rocha
Rosa ++++ Iniciou produção da peça 4
Rocha ++++ Recebida mensagem reply(espaco_nok) de Renata
Rocha ++++ Mandou pedido de espaco para Renata

---

43 A agente Rosa recebeu a primeira resposta negativa de espaço.

```
Jomi ::::   Pecas consumidas= 3
Jomi ::::   Tempo restante de Consumo =  terminou
Rosa ++++  Passo de produção 3
Jomi ::::   Mandou pedido
Renata .....   Recebida mensagem inform(peca(3)) de Rosa
Renata .....   Estoque aumentado para:  3 [peca(2), peca(3), peca(3)]
Rosa ++++  Passo de produção 2
Rosa ++++  Passo de produção 1
Renata .....   Recebida mensagem request(espaco) de Rocha
Renata .....   Negado espaco para Rocha
Rosa ++++  Passo de produção 0
Rosa ++++  Mandou pedido de espaco para Renata
Rocha ++++  Recebida mensagem reply(espaco_nok) de Renata
Rocha ++++  Mandou pedido de espaco para Renata
Renata .....   Recebida mensagem request(peca) de Jomi
Renata .....   Estoque diminuido para:  2 [peca(3), peca(3)]
Renata .....   Enviado  peca(2)  para: Jomi
Jomi ::::   Recebeu produto: peca(2)
Jomi ::::   Tempo restante de Consumo =  3
Jomi ::::   Tempo restante de Consumo =  2
Renata .....   Recebida mensagem request(espaco) de Rosa
Renata .....   Confirmado espaco para Rosa Reservas= 1
Rosa ++++  Recebida mensagem reply(espaco_ok) de Renata
Jomi ::::   Tempo restante de Consumo =  1
Rosa ++++  Mandou peça 4 para Renata
Jomi ::::   Tempo restante de Consumo =  0
Dalton ***  Papeis Identificados:
papel(Dalton, Inspetor, -5),
papel(Rocha, Produtor, 3),
papel(Jomi, Consumidor, 1),
papel(Rosa, Produtor, 1),
papel(Renata, Intermediário, 1),

Dalton *** Revisando o papel de  Jomi
Papel relacionado com PI Intermediário, fechou com o agente Renata.
O Agente Jomi continua com papel de Consumidor

Dalton *** Revisando o papel de  Rosa
Processo de Rosa relacionado ao agente Intermediário
MAIN = CICLO.
CICLO = (   CICLO(1) |  CICLO(2)) ; (CICLO).
CICLO(1) =
envia_mensagem(request(espaco)) ->
recebe_mensagem(reply(espaco_ok)) ->
envia_mensagem(inform(peca(_))) ->
envia_mensagem(request(espaco)) ->
SKIP.
CICLO(2) =
recebe_mensagem(reply(espaco_ok)) ->
envia_mensagem(inform(peca(_))) ->
```

SKIP. [44]

Dalton *** Revisando o papel de Renata
Papel relacionado com PI Consumidor, fechou com o agente Jomi.
Papel relacionado com PI Consumidor, fechou com o agente Jomi.
Papel relacionado com PI Produtor, fechou com o agente Rocha.
O Agente Renata continua com papel de Intermediário

{ Funcionamento normal da sociedade }

Dalton *** Papeis Identificados:
papel(Dalton, Inspetor, -7),
papel(Rocha, Produtor, 1),
papel(Jomi, Consumidor, 2),
papel(Rosa, indeterminado, 2),
papel(Renata, Intermediário, 2),

Dalton *** Revisando o papel de Rocha
Processo de Rocha relacionado ao agente Intermediário

```
MAIN = CICLO.
CICLO = (CICLO(1)) ; (CICLO).
CICLO(1) =
envia_mensagem(request(espaco)) ->
  recebe_mensagem(reply(espaco_ok)) ->
  envia_mensagem(inform(peca(_))) ->
  SKIP
|
  recebe_mensagem(reply(espaco_nok)) ->
  SKIP).
```

[45]

---

44 A agente Rosa mudou para o papel de indeterminado.

45 O agente Rocha mudou para o papel de indeterminado porque o Inspetor não tem descrito um papel para Produtor que recebe resposta negativa de espaço.

6 BIBLIOGRAFIA

[ALL 80] ALLEN, James F.; PERRAULT, C. Raymond. Analyzing Intention in Utterances. **Artificial Intelligence**, Amsterdam, v.15, n.3, p.143-178, Dec. 1980.

[BER 92] BERTHET, Sabine; DEMAZEAU, Yves; BOISSIER, Oliver. Knowing Each Other Better. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED ARTIFICIAL INTELLIGENCE, 11., 1992. **Proceedings...** [S.l.]:Glen Arbor, 1992. p. 1-20.

[BOR 94] BORDINI, Rafael Heitor. **Suporte Lingüístico para Migração de Agentes**. Porto Alegre: CPGCC da UFRGS, 1994. 128p. Dissertação de Mestrado.

[BRA 84] BRATMAN, Michael. Two Faces of Intention. **The Philosophical Review**, [S.l.], v.93, n.3, p.275-405, July 1984.

[CAS 90] CASTELFRANCHI, Cristiano. Social Power: A Point Missed in Multi-Agent, DAI and HCI.. In: DEMAZEAU, Yves; MÜLLER, Jean-Pierre (Eds.). **Decentralized Artificial Intelligence**. North-Holland: Elsevier, 1990.

[CHA 91] CHANG, Man Kit, WOO; Carson C.. SANP: A Communication Level Protocol for Negotiations. In: MAAMAW WORKSHOP, 3., 1991. Vancouver, Canada. **Proceedings...** [S.l.]: Kaiser-slautern, 1991.

[COH 87] COHEN, Philip R.; LEVESQUE, Hector J. Intention = Choice + Commitment. In: NATINAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 6., 1987. **Proceedings...** [S.l.: s.n.], 1987.

[COR 92] CORRÊA, Milton. **Arquiteturas para organização de Conversas entre Agentes Autônomos.** Lisboa, Portugal: INESC - Instituto de Engenharia de Sistemas e Computadores, 1992.

[COS 93] COSTA, A. C. Rocha. **Some principles for a functionalist account of complex systems,** Part I: informal statement of the instrumental notions. Porto Alegre: CPGCC da UFRGS, dec. 1993. (Não publicado).

[COS93a] COSTA, Antônio Carlos da Rocha; CASTILHO, José Mauro Volkmer de; CLÁUDIO, Dalcídio Moraes. Functional Processes and Functional Roles in Societies of Computing Agents. In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 10., 1993, Porto Alegre. **Anais...** Porto Alegre: SBC, 1993.

[COS93b] COSTA, Antônio Carlos da Rocha. **Inteligência de Máquina**: Esboço de uma Abordagem Construtivista. Porto Alegre: CPGCC da UFRGS, 1993. 168p. Tese de Doutorado.

[COS 94] COSTA, Antônio Carlos da Rocha; HÜBNER, Jomi Fred; BORDINI, Rafael Heitor. On Entering an Open Society. In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 11., 1994, Fortaleza. **Anais...** Fortaleza: SBC, 1994.

[COS94a] COSTA, Antônio Carlos da Rocha; CASTILHO, José Mauro Volkmer de; CLÁUDIO, Dalcídio Moraes. **Toward a constructive notion of functionality**. Porto Alegre: CPGCC da UFRGS, 1994. (Não publicado).

[DEM 90] DEMAZEAU, Yves; MÜLLER, Jean Pierre. **Decentralized Artificial Intelligence.** In: DEMAZEAU, Yves; MÜLLER, Jean-Pierre (Eds.). **Decentralized Artificial Intelligence**. Amsterdam: Elsevier, 1990.

[DEM 93] DEMAZEAU, Yves. **Distributed Artificial Intelligence & Multi-Agent Systems.** In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 10., 1993, Porto Alegre. **Anais...** Porto Alegre: SBC, 1993.

[FIN 92] FININ, Tim; FRITZSON, Rich; McKAY, Don. A Language and Protocol to Support Intelligent Agent Interoperability. In: CE & CALS CONFERENCE, 1992, Washington. **Proceedings...** Washington: [s.n.], 1992.

[GAS 88] GASSER, Les. Distribution and Coordination of Tasks Among Intelligent Agents. In: JCAI CONFERENCE ON AI, 1988, Amsterdam. **Proceedings...** Amsterdam: Springfield, 1988.

[GAS 91] GASSER, Les. Social conceptions of knowledge and action: DAI foundations and open systems semantics. **Artificial Intelligence**, Amsterdam, v.47, p.107-138, 1991.

[GAS91a] GASPAR, Graça. Communication and Belief Changes in a Society of Agents: Towards a Formal Model of an Autonomous Agent. In: DEMAZEAU, Yves; MÜLLER, Jean-Pierre (Eds.). **Decentralized Artificial Intelligence - 2**. Amsterdam: Elsevier, 1991.

[HEW 91] HEWITT, Carl. Open Information System Semantics for Distributed Artificial Intelligence. **Artificial Intelligence**, Amsterdam, v.47, p.79-106, 1991.

[HOA 85] HOARE, Charles Antony Richard. **Communicating Sequential Processes.** New Jersey: Prentice-Hall, 1985. 256p.

[HÜB 94] HÜBNER, Jomi Fred. **Estudo da Entrada de Agentes na Sociedade Produtor-Consumidor**. Porto Alegre: CPGCC da UFRGS, 1994.

[KOS 94] KOSKIMIES, Kai; MÄKINEN, Erkki. Automatic Synthesis of State Machines from Trace Diagrams. **Software-Practice and Experience**, Sussex, England, v.24, n.7, p.643-58, 1994.

[MAR 90] MARUICHI, Takeo; ICHIKAWA, Masaki; TOKORO, Mario. Modeling Autonomous Agents and Their Groups. In: DEMAZEAU, Yves; MÜLLER, Jean-Pierre (Eds.). **Decentralized Artificial Intelligence**. Amsterdam: Elsevier, 1990.

[MAY 95] MAYFIELD, James; LABROU, Yannis; FININ, Tim. Desiderrata for Agent Communication Languages. In: AAAI SYMPOSIUM ON INFORMATION GATHERING FROM HETEROGENEOUS, DISTRIBUTED ENVIRONMENTS, 1995, Stanford. **Proceedings...** Stanford: Stanford University, 1995.

[MON 93] MONIZ, Luis Manuel Ferreira Fernandes. **SSAA: Sistema para Simulação de Agentes e Ambientes.** Lisboa: Universidade Técnica de Lisboa, jun. 1993. Dissertação de Mestrado.

[POL 92] POLLACK, Martha E.. The uses of plans. **Artificial Intelligence**, Amsterdam, v.57, p.43-68, 1992.

[POP 93] POPULAIRE, Philippe et al. Description et Implementation de Protocoles de Communication en Univers Multi-Agents. In: IÈRES JOURNÉES FRANCOPHONES SUR I'INTELLIGENCE ARTIFICIELLE DISTRIBUÉE ET LES SYSTÈMES MULTI-AGENTS, 1993, Toulouse. **Proceedings...** Toulouse:AFCET & AFIA, 1993.

[SIC 92] SICHMAN, Jaime Simão; DEMAZEAU, Yves; BOISSIER, Oliver. When can knowledge-based sytems be called agents? In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 9., 1992, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1992.

[SIC 94] SICHMAN, Jaime Simão et al. A Social Reasoning Mechanism Based On Dependence Networks. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, 11., 1994. **Proceedings...** [S.l.]:John Wiley & Sons, 1994.

[TAN 92] TANENBAUM, Andrew S. **Modern Operating Systems**. Amsterdam: Prentice Hall, 1992.

[WER 87] WERNER, Eric. Cooperating Agents: A Unified Theory of Communication and Social Structure. In: DENNET, Daniel C. (Org.). **The Intentional Stance**. London: MIT Press, 1987. p. 1-35.

[WOO 94] WOOLDRIDGE, Michael J.; JENNIGS, Nicholas R. Agent Theories, Architectures, and Languagees: A Survey. In: ECAI - WORKSHOP ON AGENT THEORIES, ARCHITECTURES & LANGUAGES, 1994, Amesterdam. **Proceedings...** Amsterdam: [s.n.], 1994. p. 1-32.