Uma Arquitetura de Controle para Sistemas Complexos de Visão Computacional.

Reinaldo A. C. Bianchi[1]
Anna Helena Reali Costa Rillo[2]

[1]Divisão de Automação e Inteligência Artificial - Laboratório de Sistemas Integráveis
[2]Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo - EPUSP
Av. Prof. Luciano Gualberto, Travessa 3, nº 158. 05508-900 São Paulo - SP - Brasil
rbianchi@lsi.usp.br arillo@pcs.usp.br

Abstract: According to new approaches to complex computer vision systems, vision should not be considered as a self-contained system, but as a system interacting with its environment through perception and action. In the past years some architectures (top-down, bottom-up, etc.) has been frequently used to control this class of systems. This paper briefly reviews these architectures and presents a heterarchic, multi-agent purposive architecture for the control of complex computer vision systems.

Resumo: Atualmente, visão tem sido abordada como sendo um sistema que interage com o ambiente através de percepção e ação e não mais como um sistema contido em si mesmo. Algumas arquiteturas (*top-down, bottom-up, etc.*) foram tradicionalmente usadas para o controle de sistemas de visão computacional. Este artigo faz uma breve revisão destas arquiteturas, para então apresentar uma arquitetura com abordagem heterárquica, multi-agentes e propositada para o controle de sistemas complexos de visão computacional.

1. Introdução

Dentro das novas abordagens para a construção de sistemas de visão computacional complexos, visão não é mais vista como um problema contido em si mesmo, mas sim como um sistema que interage com o ambiente através de percepção e ação. Para isso, torna-se necessário que o sistema tenha seu laço de controle fechado, adotando abordagens ativas ou propositadas, com diferentes demandas temporais e computacionais.

As arquiteturas tradicionalmente usadas para sistemas de visão computacional têm se mostrado insuficientes para atender os requisitos dessas novas abordagens.

Este artigo revisa algumas arquiteturas tradicionais, classificando-as de acordo com a estratégia de controle e sua forma de computação. Assim, tem-se: controle hierárquico *versus* controle heterárquico e computação serial *versus* distribuída. Baseado nesse estudo, este artigo propõe uma nova arquitetura, seguindo uma abordagem heterárquica, multi-agentes e propositada para o controle dos sistemas de visão computacional complexos.

2. Arquiteturas para Sistemas de Visão Computacional

O sucesso de um sistema de visão computacional depende em grande parte da adoção de uma arquitetura adequada. Esta seção discute as arquiteturas e estratégias de controle tradicionalmente usadas nos sistemas de visão computacional e sua adequação às novas abordagens para a solução do problema de visão. Primeiramente, a análise será feita sob o foco da estratégia de controle utilizada, iniciando pelo controle hierárquico e suas formas.

As arquiteturas de controle hierárquico são aquelas onde o fluxo de informações está estruturado hierarquicamente, de forma que somente módulos entre níveis consecutivos da hierarquia podem trocar informações entre si. Dessa forma, pode-se dividir o controle hierárquico em *bottom-up*, *top-down* e misto, de acordo com o sentido que as informações fluem no sistema (figura 1).

Segundo [Ballard e Brown, 1982], as arquiteturas de controle *bottom-up*, também chamadas *data-driven*, são aquelas onde o processo de interpretação inicia com a construção da imagem generalizada, gerando então estruturas segmentadas e finalmente descrições.

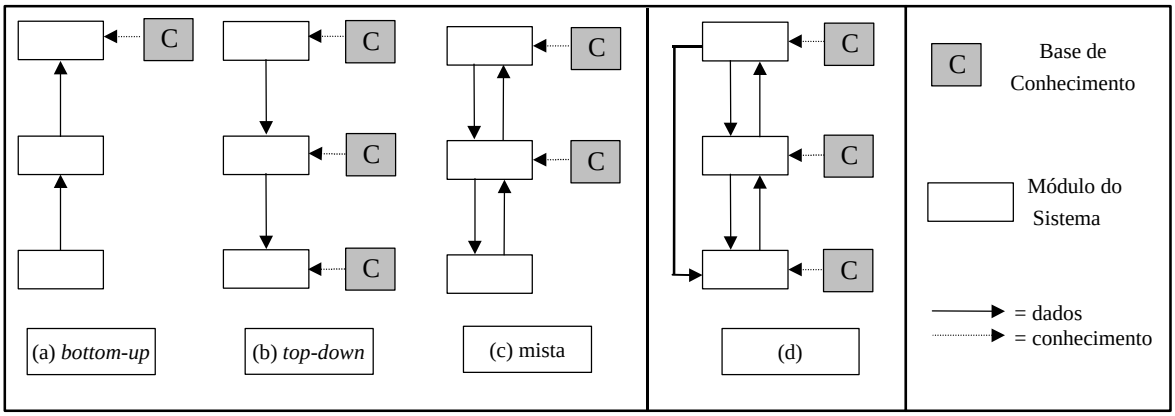

Figura 1 - Esquema do fluxo de dados, do fluxo de controle e do uso do conhecimento em várias arquiteturas.

Nestas arquiteturas, os projetistas procuram desenvolver algoritmos e estruturas de dados genéricos, buscando independência do domínio do sistema, além de utilizar o conhecimento somente no último estágio de interpretação da cena. A figura 1(a) esquematiza o uso do conhecimento e o fluxo de informação em uma arquitetura *bottom-up*.

Um exemplo típico de sistema de visão computacional que usa a arquitetura *bottom-up* é o sistema de rotulação de linhas proposto por [Waltz , 1975], cujo objetivo consiste em produzir uma interpretação para as linhas em imagens de cenas compostas por poliedros, através de uma rotulação que identifique a origem e as propriedades físicas de cada linha na imagem. Os rótulos indicam se a linha é gerada por um objeto ocludente, por uma borda côncava, convexa ou por sombras.

Para gerar essa rotulação, o sistema faz um pré-processamento da imagem para encontrar as linhas, um agrupamento para reunir as linhas pertencentes a um mesmo objeto e, finalmente, é usado um algoritmo de propagação de restrições que, a partir de uma tabela de junções possíveis e de restrições físicas, encontra uma descrição plausível para as linhas do objeto.

A arquitetura *bottom-up,* apesar de largamente utilizada e com êxitos reconhecidos em sistemas simples e bem estruturados, é de difícil aplicação em sistemas com ciclos percepção-ação em ambientes complexos, por exigir algoritmos robustos e genéricos o suficiente em cada módulo.

Nas arquiteturas *top-down,* também chamadas arquiteturas *model-driven,* "modelos de alto nível em uma base de conhecimentos geram hipóteses ou previsões de estruturas geométricas, segmentadas ou generalizadas da imagem de entrada, sendo que a interpretação desta imagem resulta da verificação dessas predições" [Ballard e Brown, 1982].

As arquiteturas *top-down* são, portanto, fortemente baseadas no conhecimento do domínio do sistema. Assim, o projetista utiliza este conhecimento para direcionar os tratamentos dos dados e construir algoritmos, visando otimizar o funcionamento do sistema. A utilização do conhecimento no sistema se dá, geralmente, em todos os estágios de interpretação da cena, de maneira implícita e/ou explícita. A figura 1(b) esquematiza o uso do conhecimento e o fluxo de informações em uma arquitetura *top-down*.

Muitos sistemas que utilizam o controle *top-down* são aplicados no campo da visão de máquina para aplicações industriais. Isto porque, nesta área, o domínio de aplicação dos sistemas geralmente é simples e estruturado (ou pode se tornar simples por manipulação do ambiente) e bem conhecido, com tarefas de reconhecimento visual sobre objetos bem definidos.

Buscando arquiteturas com organizações mais flexíveis, incorporando as abordagens *top-down* e *bottom-up* como forma de melhor adequação às tarefas visuais mais complexas, arquiteturas mistas são utilizadas.

Os sistemas com arquitetura de controle mista (figura 1c) possuem informações fluindo em ambos os sentidos entre os níveis de representação. Por exemplo, o sistema procura de maneira *top-down* uma região de interesse em uma imagem e depois processa esta região seguindo uma estratégia *bottom-up*. Um exemplo de sistema misto é o algoritmo de detecção de tumores pulmonares em imagens radiográficas proposto por [Ballard, 1976]. O sistema trabalha em 3 passagens: na primeira, encontra as fronteiras do pulmão seguindo um controle *top-down*; na segunda, procura candidatos a tumores de maneira *bottom-up*; encontrados os candidatos, estes são analisados novamente seguindo uma estratégia *top-down*.

As arquiteturas de controle heterárquicas (figura 1d) são aquelas onde o fluxo de informações não está estruturado hierarquicamente, de forma que a troca de informação entre módulos em quaisquer níveis é permitida.

Com relação à forma de computação, as arquiteturas podem ser seriais ou distribuídas. Na serial todas as operações são realizadas seqüencialmente no tempo, desconsiderando o fato de serem independentes ou não. Nas arquiteturas distribuídas (ou paralelas) existe o uso da concorrência em suas computações, onde, por

exemplo, diferentes partes de uma imagem podem ser tratadas ao mesmo tempo. O amplo uso da concorrência é a solução preferida pela evolução dos sistemas biológicos; por exemplo, a visão de baixo nível é altamente paralela.

Entre as várias maneiras conhecidas de implementar as arquiteturas distribuídas, duas são mais utilizadas: *blackboard* e Multi-Agentes.

As arquiteturas *blackboard* são compostas por diversos módulos de processamento que utilizam uma área comum, chamada de quadro-negro, para compartilhar o conhecimento necessário.

Um exemplo da implementação de uma arquitetura *blackboard* é o trabalho de [Elfes, 1986]. Neste trabalho, um sistema é dividido em níveis de processamento (controle robótico, aquisição de informação sensorial, interpretação sensorial, integração sensorial, etc.) que se comunicam através de um *blackboard*.

Diferentemente da arquitetura *blackboard*, um sistema Multi-Agentes pode ser visto como uma comunidade de especialistas, chamados agentes, cooperando, coabitando ou competindo em uma sociedade. Cada agente realiza uma tarefa e interage com os outros agentes quando necessário. A comunicação tanto de dados quanto de controle entre os agentes é feita de maneira direta e o conhecimento é distribuído entre os agentes.

Um dos problemas centrais para a construção de sistemas complexos que interagem com o ambiente através de percepção e ação é a organização e o controle dos módulos do sistema. Dentre todas as abordagens vistas para o controle de um sistema de visão computacional, a que possibilita uma melhor solução para esse problema é a multi-agentes propositada.

O paradigma propositado para a visão computacional [Rillo, A. et. Al., 1996; Aloimonos, 1994] considera a visão dentro de um contexto de tarefas que um agente deve realizar, retirando dos propósitos do agente as restrições para solucionar o problema de visão. Desta forma, este paradigma leva a soluções de tarefas específicas e não de uso geral, sendo que uma visão genérica (*multi-purpose)* surge da organização de diversas soluções dedicadas a diferentes tarefas visuais, tornando uma tendência natural a implementação deste paradigma em uma arquitetura multi-agentes, onde o processamento distribuído, aliado às restrições advindas do controle dos sensores e da tarefa a ser realizada, permite alcançar a eficiência necessária no desempenho de sistemas complexos de ação-percepção.

3. Uma Arquitetura Multi-Agentes

A arquitetura multi-agentes proposta consiste numa sociedade de agentes autônomos (AAs), comunicando entre si através de uma rede de comunicação descentralizada e totalmente conectada (figura 2), sendo cada qual responsável por um comportamento diverso, organizados segundo regras de comportamento e estrutura de autoridade.

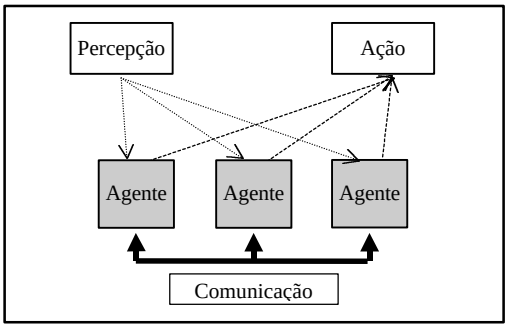

Figura 2 - Topologia da arquitetura proposta.

Nesta arquitetura, a definição da sociedade de agentes é dinâmica, podendo os AAs serem inseridos ou retirados da sociedade durante a operação do sistema, segundo o controle de um ou vários agentes desta sociedade. Assim, caso se deseje criar um sistema auto-gerenciado e multi-propósito, basta definir os agentes necessários para cada propósito e criar um agente que controle a troca de propósitos, com a função ainda de gerenciar os AAs necessários a cada propósito. Os agentes autônomos se relacionam com as camadas da *Subsumption Architecture* de [Brooks, 1986], pois percebem e agem diretamente no mundo e podem suprimir as ações uns dos outros.

A estrutura de autoridade da sociedade é o que torna possível a decisão sobre como os recursos do sistema devem ser alocados, decidindo qual agente deve ter o controle de um recurso em um determinado momento. A maneira pela qual esta decisão é tomada depende das regras de comportamento definidas para a sociedade em uma implementação específica e pode ser, por exemplo, o resultado de uma competição entre agentes.

Um recurso é definido como sendo parte do sistema com uso compartilhado pelos agentes e que pode ser controlada por somente um agente em um determinado momento. Por exemplo, um manipulador robótico em uma célula de montagem é um recurso e o sistema de direção de um robô móvel, outro; por outro lado, uma câmera fixa (e seu hardware e software de aquisição de dados) não é um recurso, pois todos os agentes podem ter a imagem capturada ao mesmo tempo. Caso esta câmera faça parte de um sistema de visão ativa, ela torna-se um recurso, uma vez que os agentes competem pelo controle do processo de aquisição de dados.

A estrutura de autoridade de um sistema possui uma profunda relação com a precedência e a dependência entre as ações dos comportamentos dos agentes presentes

na sociedade, e sua definição deve ser feita a partir de um estudo da linearização de um plano de atividades, que tem nos agentes autônomos seus operadores e cujo objetivo é o de realizar as intenções do sistema. Vale notar que é esta estrutura que relaciona a arquitetura proposta à de Brooks, permitindo que os agentes autônomos inibam e suprimam uns aos outros, retirando os recursos que necessitam.

3.1 Agentes Autônomos (AAs)

Os agentes autônomos constituem uma das principais características da arquitetura proposta, sendo modelados com base na teoria de DAI-MAS e possuindo fortes raízes no trabalho [Boissier e Demazeau, 1994]. Define-se um AA como:

<Agente> ::= <regras> <agentes> <estado> <linguagem de comunicação> <capacidades decisórias>

Onde:

<regras> são as regras de comportamento da sociedade na qual o agente está inserido;

<agentes> são os agentes na sociedade e a estrutura de autoridade entre eles;

<estado> é a descrição do estado dos elementos pertinentes que façam parte do domínio, assim como da alocação e do estado dos recursos da sociedade;

<linguagem de comunicação> é a descrição da linguagem de comunicação usada pelos agentes;

<capacidades decisórias> são as capacidades que um AA possui que o tornam apto a decidir qual agente deve ter o controle de um recurso em um determinado momento, baseadas, entre outros, na estrutura de autoridade.

Cada agente autônomo é ainda dividido em 5 módulos, que são: Sensor, Atuador, Comunicação, Decisão e Raciocínio, apresentados na figura 3.

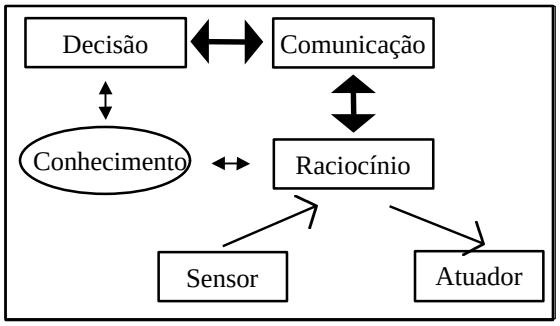

Figura 3 - Modelo de um agente autônomo.

Exemplificando a interação entre estes módulos, caso necessite de um recurso, o módulo de raciocínio o pede para seu módulo de comunicação. Este comunica aos outros agentes a requisição do recurso. O agente que tem o recurso recebe o pedido em seu módulo de comunicação e, junto com o modulo de decisão, decide se interrompe o seu trabalho e cede o recurso. Se o agente requisitante conseguir o recurso, ele avisa o seu módulo de raciocínio, que então passa a utilizá-lo, mas checando a cada utilização se ele não o perdeu para outro agente, o que lhe é indicado pelo módulo de comunicação.

Com este modelo não é necessário interromper a execução do comportamento a cada mensagem recebida, além de permitir que os módulos de comunicação e decisão sejam definidos e implementados da mesma maneira para todos os agentes. Finalmente, dispensa o conhecimento da sociedade no módulo de raciocínio, que só precisa conhecimento sobre os recursos.

Outra característica essencial relacionada aos agentes em um ambiente multi-agentes é a sua capacidade de comunicação. Isto torna a definição da linguagem de comunicação a ser usada um tópico fundamental.

3.2 A Linguagem de Comunicação dos AAs

Com base na linguagem de comunicação descrita em [Boissier e Demazeau, 1994], foi definida a seguinte linguagem de interação entre os agentes autônomos do sistema:

<interação> ::= *<tipo> <conteúdo>*

onde:

<tipo> é o tipo da comunicação;

<conteúdo> é o conteúdo da comunicação.

A implementação dos elementos da linguagem de comunicação depende das regras de comportamento da sociedade de AAs de um sistema, porém existem alguns tipos básicos necessários:

adição de novo agente (***addAgent***): usado para adicionar um agente à sociedade;

remoção de agente (***deleteAgent***): usada para retirar um agente da sociedade;

informação (***inform***): envia informações sobre a sociedade para um novo agente;

confirmação (***acknowledge***): usada para que todos os outros agentes confirmem que receberam a mensagem de adição ou remoção.

requisição (***request***): usado por um agente para requisitar o controle de um recurso do sistema a outro agente;

transferência (***transfer***) usado para transferir o controle de um recurso de um AA para outro.

liberação (***free***): usado por um agente para avisar aos outros que ele não mais necessita o controle de um recurso;

A tabela 1 resume a linguagem de comunicação definida para os agentes autônomos.

<tipo>	*<conteúdo>*
addAgent	<nome do novo agente> <nova estrutura de autoridade>
deleteAgent	<nome do agente>
inform	{<recurso> <ocupado|livre> <agente controlador>}
acknowledge	<nome do agente>
request	<nome do agente> <recurso>
transfer	<agente destino> <recurso> [estado]
free	<recurso>

Tabela 1 - Linguagem de comunicação.

4. Uma implementação da arquitetura.

A arquitetura aqui proposta está sendo implementada em uma Célula Flexível de Montagem [Rillo, M. et al., 1992], da Divisão de Automação e Inteligência Artificial do Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo, que é composta por várias estações de trabalho, câmeras de vídeo e placas digitalizadoras, dois manipuladores robóticos de 5 graus de liberdade, com todos os computadores interligados por uma rede ethernet local, em um sistema realizando tarefas de montagens simples e visualmente guiadas.

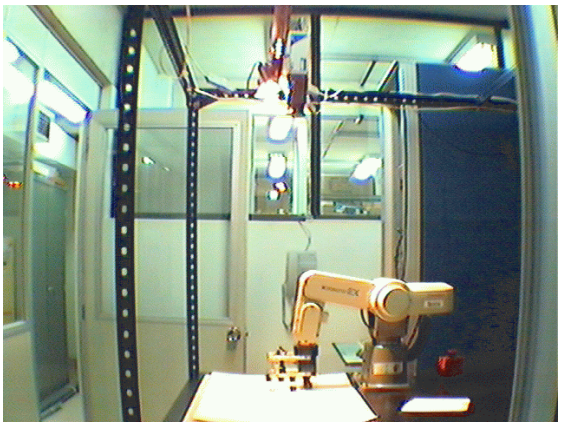

Figura 4 - A Célula Flexível de Montagem.

O sistema atualmente utiliza somente um dos manipuladores e uma câmera e está sendo implementado em várias estações de trabalho da Célula e em um microcomputador (para o controle do manipulador), onde os agentes serão executados como processos independentes e paralelos, comunicando-se através da rede da célula.

Apesar do domínio da implementação ser o de uma célula de montagens, a arquitetura pode ser aplicada em outros domínios, como o dos robôs móveis autônomos.

Para a implementação, foram definidos três comportamentos diferentes, cada um correspondendo a um agente autônomo:

1. Agente **Montador** (***Assembler***): ele deve realizar uma montagem pegando peças na área de trabalho com o manipulador e colocando-as nas posições desejadas. O objetivo da montagem e a forma e tipo das peças podem mudar, por exemplo, de uma montagem de um objeto conhecido para a seleção de peças por forma ou cor. Para isso, o agente tem que ser capaz de planejar as atividades envolvidas na montagem. Este planejamento pode ser simples ou complexo, reativo ou temporal, dependendo da complexidade que se deseja do sistema. No estado atual, o agente não possui capacidades de planejamento automático de atividades, sendo o plano, por isso, previamente definido.

2. Agente **Desobstrutor** (***Cleaner***): este agente deve desobstruir a área de trabalho. A área de trabalho é um espaço, previamente definido como importante para a montagem. Assim, objetos não desejados que uma pessoa ou outro manipulador possam ter colocado nesta área devem ser retirados pelo agente.

3. Agente **Evitador de Colisões** (***Collision avoider***): este agente deve evitar colisões do manipulador com objetos que se movam na área de trabalho (outro manipulador, uma mão), com o objetivo de preservar a integridade física do sistema.

Os recursos são definidos, conforme dito anteriormente, como sendo as partes do sistema compartilhadas pelos agentes e que podem ser controlados por apenas um agente em um determinado momento; desse modo, definiu-se que o único recurso do sistema atual seria o manipulador, pois todos os agentes podem usar a câmera ao mesmo tempo, já que nesta implementação a câmera não é usada de modo ativo.

Ainda, a sociedade dos agentes possui três regras de comportamento, que fazem os mesmos competir pelo controle do recurso do sistema:

Regra # 1: somente um agente pode controlar o manipulador em um determinado momento.

Regra # 2: qualquer agente pode requisitar o controle do manipulador para um agente com menor autoridade que ele, em qualquer momento.

Regra # 3: um agente só pode requisitar o controle do manipulador para um agente de maior autoridade se este estiver cedendo o controle.

A estrutura de autoridade definida para os agentes é definida da seguinte maneira: o agente *Evitador de Colisões* é o agente com maior autoridade, o *Desobstrutor* é o segundo e o *Montador* é o agente com menor autoridade. Pode-se notar que a estrutura de autoridade tem por objetivo principal a preservação da integridade física do sistema e, secundária, a tarefa global a ser realizada. Quando um sistema cresce, a estrutura é modificada para incorporar mais agentes.

Abaixo é apresentado um exemplo do funcionamento do sistema (exemplo 1) que descreve, em uma linguagem baseada em LISP, a seqüência de mensagens que são trocadas quando algum objeto é colocado na área de trabalho durante uma montagem. Inicialmente, o agente *Evitador* requisita o controle do sistema ao agente *Montador*, para evitar uma colisão; o agente *Montador* transfere o controle ao agente *Evitador*; ao final de perigo de colisão, o agente *Evitador de colisão* cede o controle do sistema, que é requisitado pelos agentes *Desobstrutor* e *Montador*; o agente *Evitador* transfere o controle ao de maior autoridade, o *Desobstrutor*; ao terminar de desobstruir a área de trabalho, o *Desobstrutor* cede o controle do sistema; finalmente, o agente *Montador* requisita o controle, que é transferido pelo *Desobstrutor* para ele, que pode então retornar ao seu trabalho.

```
comentário: o controle está com Assembler.
(request (collisionAvoider)(manipulator))
(transfer (collisionAvoider) (manipulator)
   (piece in grip))
comentário: a colisão é evitada.
(free (manipulator))
(request (assembler) (manipulator))
(request (cleaner) (manipulator))
(transfer (cleaner) (manipulator) (piece in
   grip))
comentário: o objeto é removido.
(free (cleaner) (manipulator))
(request (assembler) (manipulator))
(transfer (assembler) (manipulator) (piece
   in grip))
```

Exemplo 1 - Troca de mensagens durante a interrupção de uma montagem.

No exemplo 2, é apresentada a troca de mensagens que ocorre durante a adição de um novo agente na sociedade. O nome do novo agente é *NovoAgente*, sendo inserido na estrutura de autoridade acima dos agentes *Desobstrutor* e *Montador* e abaixo do *Evitador de Colisões*.

```
(addAgent (newAgent) (collisionAvoider
   newAgent assembler cleaner))
(acknowledge (collisionAvoider))
(acknowledge (cleaner))
(acknowledge (assembler))
(acknowledge (newAgent))
```

Exemplo 2 - Troca de mensagens durante a adição de um novo agente.

5. Conclusão

Visão computacional é considerada, segundo o paradigma propositado, dentro de um contexto de tarefas que um agente deve realizar, retirando dos propósitos do agente as restrições para solucionar o problema de visão. Desta forma, este paradigma leva a soluções de tarefas específicas, executadas por agentes que exibam determinados comportamentos.

A organização dos diversos comportamentos dedicados a diferentes tarefas exige uma arquitetura distribuída e modular, tornando uma tendência natural a implementação deste paradigma propositado em uma arquitetura multi-agentes, onde o processamento distribuído, aliado às restrições advindas do controle dos sensores e da tarefa a ser realizada, permite alcançar a eficiência necessária no desempenho de sistemas complexos de ação-percepção, facilitando a decomposição de cada propósito do sistema em um conjunto de comportamentos e explicitando a interação entre estes comportamentos para atingir seus propósitos.

6. Referências

ALOIMONOS, Y. What I have learned. **CVGIP: Image Understanding**, v.60, n.1, p.74-85, July 1994.

BALLARD, D. H. **Hierarchic Recognition of Tumor in Chest Radiographs.** Basel, Birkhauser-Verlag, 1976.

BALLARD, D. H.; BROWN, C. M. **Computer Vision.** Englewood Cliffs, Prentice-Hall, 1982.

BOISSIER, O; DEMAZEAU, Y. ASIC: An architecture for social and individual control and its application to Computer Vision. In: EUROPEAN WORKSHOP ON MODELING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 1994. **Proceedings.** 1994. p.107-18.

BROOKS, R. A. A robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, VRA-2, n.1, p.14-23, March 1986.

ELFES, A. A distributed control architecture for an autonomous mobile robot. **Artificial Intelligence**, Computational Mechanics Publications, v.1, n2,1986, p135-44.

RILLO, M.; RILLO, A.H.R.C.; COSTA, L.A.R. The LSI assembly cell. In: IFAC/IFIP/IFORS/ IMACS/ISPE Symposium on information control problems in manufacturing technology, 7º, Toronto, 1992. **Proceedings.** IFAC, 1992. p. 361-5.

RILLO, A. H. R. C., BIANCHI, R. A. C., MOREIRA Jr, B., FERRAZ, F. "Integrando Visão e Comportamento: Uma aplicação de reconstrução propositiva". **XI**. **Congresso Brasileiro de Automática**, São Paulo, 2-6 Set., 1996. SBA, pp. 573-578.

WALTZ, D. I. "*Generating semantic descriptions from drawings of scenes with shadows*". In **The Psicology of Computer Vision**. New York, McGraw-Hill, 1975.