Desenvolvimento de uma arquitetura para simulação do Funcionamento distribuído e paralelo do cérebro

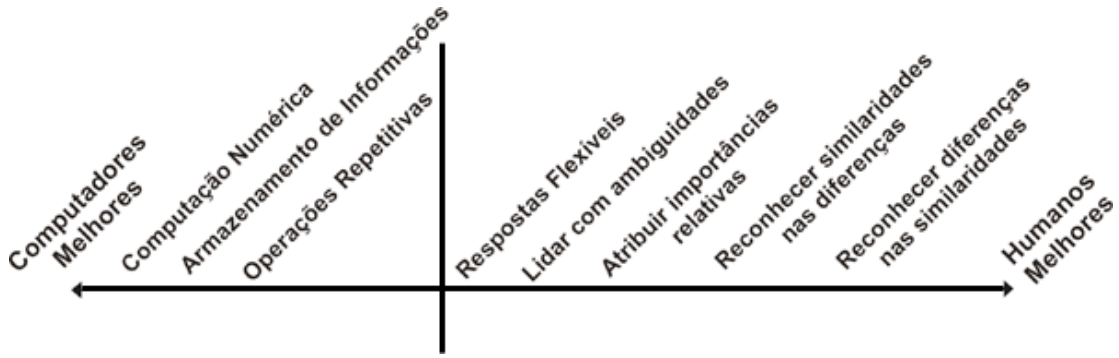
Anderson de Rezende Rocha – 5º período de Ciência da Computação
Prof. Antônio Maria Pereira de Resende – Depto. Ciência da Computação
{undersun,tonio}@comp.ufla.br

1. Introdução

O cérebro humano tem despertado um grande fascínio ao longo das últimas décadas. Estas, apesar de todas as coisas ruins que nos ofereceram, proporcionaram uma explosão inédita de estudos científicos relacionados ao cérebro. Devido a esses estudos, é possível, hoje, obter imagens desse órgão em pleno funcionamento. Tornou-se visível, pela primeira vez na história humana, o que sempre fora um mistério: como atua essa intrincada quantidade de células enquanto pensamos, imaginamos e sonhamos. Essa inundação de dados neurobiológicos permite o entendimento e o raciocínio sobre o mundo [1].

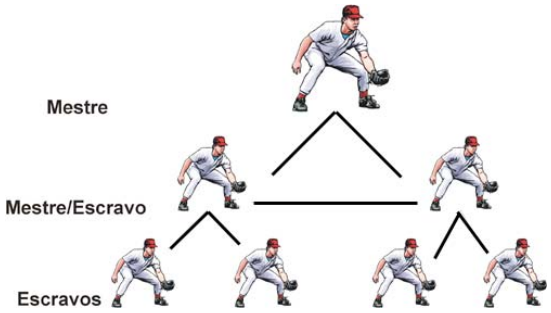
Apesar desses estudos, muitas questões ainda continuam sem respostas. Não cabe a este trabalho entrar nesse mérito.

Numa comparação entre os computadores e o cérebro humano pode-se notar claramente áreas em que um supera o outro e vice-versa. Acompanhe no diagrama abaixo:

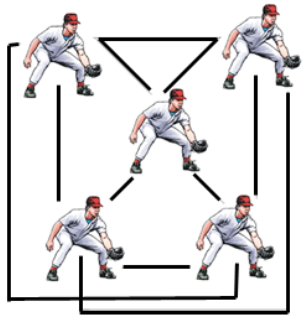


Devido a exaustivos estudos hoje se sabe que o cérebro é capaz de realizar inúmeras atividades com mais habilidade que os computadores, tais como inferências, raciocínio lógico, reconhecimento de padrões, processamento de imagens. Para isto o cérebro possui inúmeras arquiteturas, tais como:

a) diversos processos em execução paralela com um outro processo central controlando os demais.
b) diversos processos em execução sem um controlador central.



Arquitetura centralizada de processos


Arquitetura distribuída de processos

Ambas arquiteturas podem operar cooperativamente, situação em que os processos cooperaram visando atingir os objetivos comuns ou competitivamente, situação em que os processos competem visando atingir o bem próprio antes do bem geral.

O tema estudado tem grande relevância, pois, permite entender de forma simples e visual algumas das formas pelas quais o cérebro resolve problemas. Possibilita uma introdução à inteligência artificial e suas sub-áreas dotando qualquer leitor de argumentos suficientes para possuir uma posição crítica sobre os

problemas enfrentados atualmente nestas áreas. Além disso este estudo torna possível um pensamento sobre o que é ser inteligente? Para um robô, por exemplo, o conceito de inteligência tem o mesmo teor que para um ser humano? O que caracteriza a inteligência? Quais os problemas enfrentados quando se tenta "mapear" um processo natural para uma interpretação artificial? Quais as melhores formas de se fazer este dito "mapeamento"? Propostas de solução a estas questões são apresentadas ao longo do texto.

Para facilitar o entendimento e tornar mais agradável a leitura, o texto está disposto da seguinte forma:

1.1 O cérebro: *breve explicação sobre o funcionamento cerebral*

A vida é uma comédia para os que pensam
e uma tragédia para os que sentem.
Horace Walpole.

O cérebro humano, com um pouco mais de um quilo de células e humores neurais, é três vezes maior que o de seus primos ancestrais, os primatas não humanos. Ao longo de milhões de anos de evolução, o cérebro cresceu de baixo para cima, os centros superiores desenvolvendo-se com elaborações das partes inferiores, mais antigas. Curiosamente o crescimento do cérebro no embrião humano refaz mais ou menos o percurso evolucionário.

A parte mais primitiva do cérebro, partilhada por todas as espécies que têm um sistema nervoso superior a um nível mínimo, é o tronco cerebral em volta da medula espinhal. Esse cérebro raiz regula as funções vitais básicas, como a respiração e o metabolismo dos outros órgãos do corpo, e também controla reações e movimentos estereotipados. Não se pode dizer que o cérebro primitivo pense ou aprenda; ao contrário, ele se constitui num conjunto de reguladores pré-programados que mantêm o funcionamento do corpo como deve reagindo, se necessário, de modo a assegurar a sobrevivência.

Da mais primitiva raiz, o tronco cerebral, surgiram os centros emocionais. Milhões de anos depois, na evolução, desenvolveu-se o cérebro pensante, ou o neocórtex, o grande bulbo de tecidos ondulados que forma as camadas externas.

Com o advento dos mamíferos, vieram novas e decisivas camadas, chave do cérebro atual. Estas, em torno do tronco cerebral, lembravam um pouco um pastel com um pedaço mordido embaixo, no lugar onde se encaixa o tronco cerebral. Uma dessas partes é o sistema límbico ou *limbus* palavra do latim orla. [1].

À medida que evoluía, o sistema límbico foi aperfeiçoando duas poderosas ferramentas de particular interesse neste trabalho: aprendizagem e memória. Esses avanços revolucionários possibilitavam que um animal fosse muito mais esperto nas opções de sobrevivência e aprimorasse suas respostas. Se uma comida causasse doença poderia ser evitada da próxima vez. Decisões de como saber o que comer e o que rejeitar, resolver problemas, elaborar planos etc. tornaram-se viáveis e interessantes.

O neocórtex do *Homo sapiens*, muito maior que o de qualquer outra espécie, acrescentou tudo o que é distintamente humano. O neocórtex é a sede do pensamento; contém os centros que reúnem e compreendem o que os sentidos percebem. Acrescenta a um sentimento o que se pensa dele e permite que se tenha sentimentos sobre idéias, artes, símbolos, imagens etc.

Na evolução, o neocórtex possibilitou um criterioso aprimoramento que, sem dúvida, trouxe enormes vantagens na capacidade de um organismo sobreviver à adversidade, tornando mais provável que sua progênie, por sua vez, passasse adiante os genes contendo esses mesmos circuitos neurais. A vantagem para a sobrevivência deve-se à capacidade do neocórtex de criar estratégias, planejar a longo prazo, replanejar situações em certas condições e outros artifícios mentais. Além disso, os triunfos da arte, civilização e cultura são todos frutos do neocórtex [1].

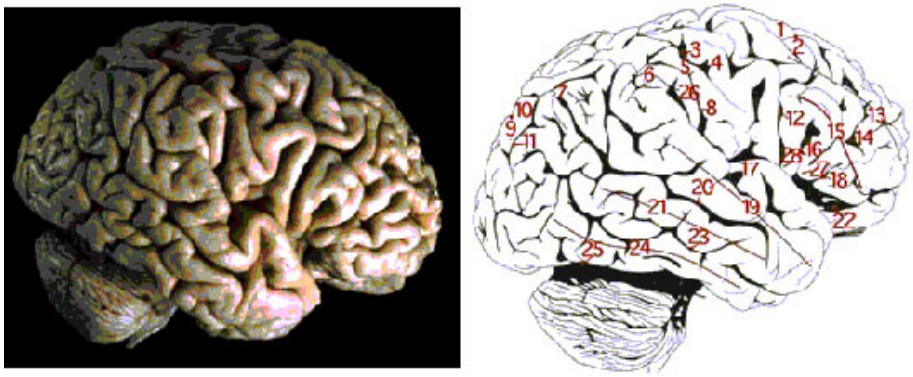
De particular interesse neste trabalho são as arquiteturas do cérebro. Por arquiteturas entende-se as inúmeras maneiras diferentes de o cérebro alocar a gerenciar recursos para resolver um dado problema. Dada a infinita complexidade do cérebro é bastante viável crer que também são inúmeras estas arquiteturas. Foram selecionadas neste projeto apenas duas arquiteturas de maior interesse embora isso não cause tamanha perda de generalidade:

Arquitetura 1: O neocórtex ao receber um problema em particular aloca um processo centralizador que será o coordenador de todo o trabalho. Em seguida, muitos outros processos são acionados e imediatamente ligados ao processo central. Todo o trabalho dos processos menores são submetidos ao processo central de modo que este fique informado dos acontecimentos podendo-os repassar imediatamente, caso necessário, aos outros processos. Os processos menores podem trabalhar cooperativamente com cada processo procurando parte da solução e compartilhando-a com os demais via processo centralizador ou competitivamente com cada processo procurando por si a solução para o problema.

> Exemplo: *Problema apresentado: reconhecer um conjunto de objetos. Este problema é apresentado ao neocórtex e este instancia um processo central que será o coordenador dos processos menores. Em seguida muitos outros processos menores são submetidos ao coordenador. Pelo modo cooperativo os processos menores identificam partes do mesmo objeto e disponibilizam para o coordenador que interpreta os resultados repassando-os aos outros processos menores. Pelo modo competitivo a informação ao chegar no processo coordenador não é repassada para os demais processos.*

Arquitetura 2: O neocórtex ao receber um problema em particular procura alocar o número de processos que julgar necessário à solução. Todos com a mesma prioridade de execução sem um processo controlador ou coordenador central. Os processos podem trabalhar tanto de forma cooperativa realizando o trabalho em grupo e disponibilizando as informações a todos os outros processos diretamente ou competitivamente com cada processo trabalhando por si próprio sem compartilhamento de informações com outros processos.

> Exemplo: *Tome-se o exemplo anterior. Dado que o neocórtex recebeu o problema ele aloca os processos e divide as tarefas. Pelo modo cooperativo os processos vão identificar objeto a objeto disponibilizando a todos os outros processos o que já foi analisado. Pelo modo competitivo estas informações não são enviadas a nenhum outro processo e cada processo tenta resolver o problema por si só.*



1. Superior frontal gyrus 2. Superior frontal sulcus 3. Central sulcus 4. Precentral gyrus 5. Postcentral gyrus 6. Supramarginal gyrus 7. Angular gyrus 8. Postcentral sulcus 9. Parieto-occipital sulcus 10. Superior parietal lobule 11. Intraparietal sulcus 12. Precentral sulcus 13. Middle frontal gyrus 14. Inferior frontal sulcus 15. Inferior frontal gyrus 16. Anterior ascending ramus of lateral sulcus 17. Transverse temporal gyrus 18. Anterior horizontal ramus lateral sulcus 19. Superior temporal gyrus 20. Superior temporal sulcus 21. Middle temporal gyrus 22. Stem lateral sulcus 23. Inferior temporal sulcus 24. Inferior temporal gyrus 25. Preoccipital notch 26. Posterior branch of lateral sulcus 27. Triangular part of inferior frontal gyrus 28. Opercular part of inferior frontal gyrus

1.2. IA: *inteligência artificial, breve história.*

Os computadores não são muito bons no conhecimento do que fazem: toda ação que um computador executa necessita ter sido explicitada, planejada e codificada por um programador anteriormente. Se um programa de computador encontra uma situação não descrita, geralmente, este programa dará erro ocasionando um *crash* no sistema e, em piores casos, até perda de vidas humanas, que deviam ser assistidas pelo dito programa. Para a maioria das pessoas aceita-se que os computadores são obedientes e literalmente servos não imaginativos. Para muitas aplicações isto é útil e, muitas vezes, recomendado. Entretanto, para um número cada vez maior de aplicações, desejam-se sistemas capazes de "tomar decisões" por si próprios de modo a satisfazer seus objetivos. Pensando nisso nasceu a Inteligência Artificial.

O que é inteligência artificial? Embora muitas tentativas de definir precisamente termos complexos e amplamente usados sejam exercícios fúteis é útil dar-se ao menos uma noção sobre o significado deste termo em particular. Inteligência artificial ou IA é o estudo de como fazer os sistemas computacionais resolverem problemas que, em grande parte, as pessoas resolvem melhor [2]. Notoriamente esta é uma definição efêmera devido ao atual nível da Ciência da Computação. Claramente esta definição não engloba os problemas que não podem ser resolvidos nem por computadores nem por qualquer ser humano. Além disso, excetua-se nesta simples definição qualquer abordagem filosófica. Segundo [7] IA é o ramo da ciência da computação dedicado a desenvolver equivalentes computacionais de processos peculiares à cognição humana, como por exemplo, o aprendizado, a compreensão da linguagem natural, reconhecimento de padrões etc.

Domínios de tarefas da IA atualmente [2]:

Tarefas comuns:

- Percepção
  - Visão
  - Fala
- Linguagem natural
  - Compreensão
  - Geração
  - Tradução
- Raciocínio do senso comum
- Controle de robôs

Tarefas formais

- Jogos
  - Xadrez
  - Gamão
  - Damas
  - Go
- Matemática
  - Geometria
  - Lógica
  - Cálculo Integral
  - Demonstração de propriedades de programas

Tarefas de Especialistas

- Engenharia
  - Projeto
  - Descoberta de erros
  - Planejamento de manufatura
- Análise científica
- Diagnóstico médico
- Análise financeira

1.2.1 O que é ser inteligente

Segundo [2] e [6] na vida real observa-se várias situações onde se costuma atribuir um comportamento dito inteligente a entidades coletivas: uma empresa competitiva num determinado nicho de mercado, uma equipe de futebol vencedora, uma colônia de formigas em busca de alimentos. No caso das formigas o comportamento dito inteligente está intimamente ligado ao coletivo. Poucos atribuiriam tal caracterização a uma formiga isoladamente.

Por inteligência entende-se a faculdade ou capacidade de aprender, apreender, compreender ou adaptar-se facilmente [7]. Nenhum estudo mais aprofundado neste quesito foi realizado.

Allan Turing [23] em 1950 propôs um teste, hoje conhecido como *Teste de Turing*, em que se propunha o teste ou medição da capacidade de pensamento de uma máquina. Este teste é muito conceituado ainda hoje. Turing propôs um jogo, por ele chamado de jogo da imitação. Segundo este jogo 3 (três) pessoas participam. Uma pessoa (A) será um homem. Outra pessoa será (B) uma mulher. Finalmente a terceira pessoa será um interrogador sem restrição ao sexo. A comunicação entre as pessoas deve ser por meio que uma não tenha contato com a outra. Pode-se introduzir uma quarta pessoa que sirva apenas para levar as informações a serem trocadas entre os participantes. À terceira pessoa (C) cabe interrogar (A) e (B). Ao final (C) deve ser capaz de dizer que (A) é homem e (B) é mulher. À primeira pessoa (A) cabe sempre mentir para (C) de modo a leva-lo a conclusões erradas. À segunda pessoa (B) cabe sempre ajudar o interrogador (C). O conceito de inteligência aqui está em fazer com que (C) faça a distinção entre os sexos de (A) e (B). Definido o jogo da imitação, Turing propôs a substituição de (A) ou (B) por uma máquina. Assim a máquina seria considerada hábil, inteligente se conseguisse desempenhar bem o seu papel.

Várias objeções são feitas a esta proposta de Turing. Dentre estas objeções cita-se:

i) *Objeção teológica*: "pensar é uma função da alma imortal humana. Deus deu uma alma imortal a todo homem e toda mulher, mais a nenhum outro animal ou máquina. Logo, nenhum animal ou máquina pode pensar". Ao que Turing rebate: animais se assemelham mais com os homens que com seres inanimados, como as máquinas sem contar que historicamente os argumentos teológicos freqüentemente mostraram-se insatisfatórios.

ii) *Objeção matemática*: segundo o teorema de Gödel [22] qualquer sistema lógico suficientemente poderoso pode-se formular enunciados que não são passíveis de prova ou argumentação contrária dentro do sistema, a menos que possivelmente, o próprio sistema seja inconsistente. Assim no jogo da imitação, haveria respostas que a máquina não poderia dar, ou daria uma resposta errada. Segundo Turing realmente há limitações inerentes a qualquer máquina. Entretanto, se uma máquina erra não quer dizer que ela não possa ser

inteligente. Afinal, segundo a maioria, os humanos são ditos inteligentes, mesmo que, sujeito a erros.

iii) *Objeção de Lady Lovelace* [3]: "a máquina analítica, o computador, não tem nenhuma pretensão de criar o que quer que seja. Pode fazer *tudo quanto se saiba ordenhar-lhe que faça.* A maquina jamais pode pregar-nos peças".Entretanto Turing diz o contrário. Se nós estamos esperando que um determinado estado de voltagem, por exemplo, esteja em um nível, e, na realidade, ele estiver em outro então, na realidade, a máquina nos pregou uma peça contradizendo a afirmação de Lady Lovelace.

Muitas outras objeções foram feitas à proposição de Turing. Assim o que a esta proposição nos mostra, é que não se deve tentar entender a possível *inteligência* em uma máquina comparando com o que se entende por inteligência humana. O termo inteligência é muito abrangente e pode ser entendido de inúmeras formas diferentes. Cabe ao ser humano tentar identificar estas formas e classificá-las em coerentes ou não.

Desta forma, o sucesso de uma máquina autônoma dependeria única e exclusivamente de sua capacidade de lidar com uma variedade de eventos inesperados no ambiente em que opera. Estas máquinas teriam maior capacidade de aprender tarefas de alto nível cognitivo que não são facilmente manipuláveis e continuariam a se adaptar e realizar mais tarefas gradativamente.

1.2.2 Mapeamento de problemas da IA

Os principais problemas da IA estão relacionados com o seguinte dilema:

a) Um sistema de IA precisa conter muito conhecimento para poder lidar com outras coisas além de problemas triviais, em "domínios de brinquedo".
b) Mas, com o aumento da quantidade de conhecimento, fica mais difícil acessar as coisas apropriadas quando necessário, e, portanto, mais conhecimentos têm de ser incluídos para ajudar. Conseqüentemente haverá mais conhecimentos a serem gerenciados e, portanto, mais conhecimentos precisam ser acrescentados e assim sucessivamente [2].

Pode-se dizer que a inteligência artificial ainda está na sua infância. É difícil saber exatamente sob qual perspectiva as coisas devem ser vistas. Não se pode resistir à célebre citação de Lady Lovelace há mais de 100 (cem) anos quando falava sobre a máquina analítica de Babbage:

> *Ao considerar qualquer assunto, freqüentemente existe a tendência de, primeiro superestimar aquilo que para nós já é notável ou interessante, e, segundo, por um tipo de reação natural, subestimar o verdadeiro estado do caso, quando realmente descobrimos que nossas noções superaram aquelas que eram realmente sustentáveis.* [3]

Para se ter uma idéia de quão complexa se torna uma modelagem de um problema segundo a abordagem da inteligência artificial diz-se que são necessários, por exemplo, pelo menos 67 passos para trocar uma lâmpada segundo as técnicas de IA enquanto que esta tarefa seria realizada por um indivíduo normal certamente em menos de 10 passos. [2]

1.2.3 Inteligência artificial distribuída (IAD) e sistemas multiagentes (SMA)

A inteligência artificial distribuída, IAD, um ramo da inteligência artificial, tornou-se nos últimos anos um domínio de pesquisas muito promissor [4]. Contrapondo-se à Inteligência Artificial clássica cuja ênfase é colocada na representação de conhecimentos e métodos de inferências a IAD baseia seu modelo de inteligência no comportamento social sendo a ênfase transposta para as ações e interações entre agentes [5].

Metaforicamente a IA clássica tem natureza psicológica enquanto a IAD tem natureza sociológica / etológica.

1.2.3.1 Conceitos básicos em IAD

Os principais conceitos apresentados a seguir aparecem em mais detalhes em [8].

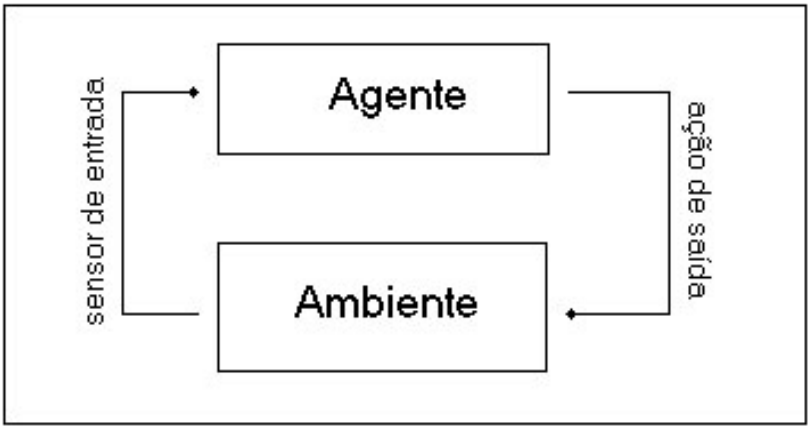
Dado um determinado sistema, denomina-se agente cada uma de suas entidades ativas. O conjunto de agentes constitui uma sociedade. O ambiente é formado pelas entidades passivas. A um agente cabe "raciocinar" sobre o ambiente, sobre outros agentes bem como decidir racionalmente objetivos a seguir e ações a tomar [9].

De acordo com [7] ativo é aquele que exerce a ação, que age. Claramente o termo agente está intimamente ligado a um conjunto de regras e bases de conhecimento além de um mecanismo de controle para sua ativação.

Em IAD um agente se assemelha a um processo do cérebro. Deste modo uma sociedade em busca da solução de um problema seria o cérebro com seus inúmeros processos também resolvendo um problema.

Iteração: são as trocas de informação que podem ocorrer entre agentes. Como será visto adiante esta comunicação pode ser direta, explicitamente por meios de comunicação pré-definidos ou indireta por meio do ambiente, por exemplo.

Basicamente um agente se comporta no ambiente segundo a figura abaixo:



Comportamento de um agente em um ambiente qualquer [13]

Organização: são as restrições aplicadas aos agentes pertencentes a uma sociedade. Existem inúmeras alternativas de representação para organização entre agentes dentre as quais pode-se citar: estruturas de autoridade, táticas de controle, modelos de tipos pares (conhecimento, ação) etc. [4]. Aprofundando o conceito de agente têm-se a definição de [10]: *um agente é uma entidade à qual nós podemos associar uma identidade única e que é capaz de executar um processamento de cálculo. Um agente pode ser considerado com um meio que produz um certo número de ações, a partir de seu conhecimento e dos mecanismos internos que lhe são próprios.*

A IAD se divide em duas subáreas, a *Resolução distribuída de problemas (RDP) e os Sistemas multiagentes (SMA).* Nas próximas seções há mais detalhes a respeito. Textos completos podem ser obtidos em [11].

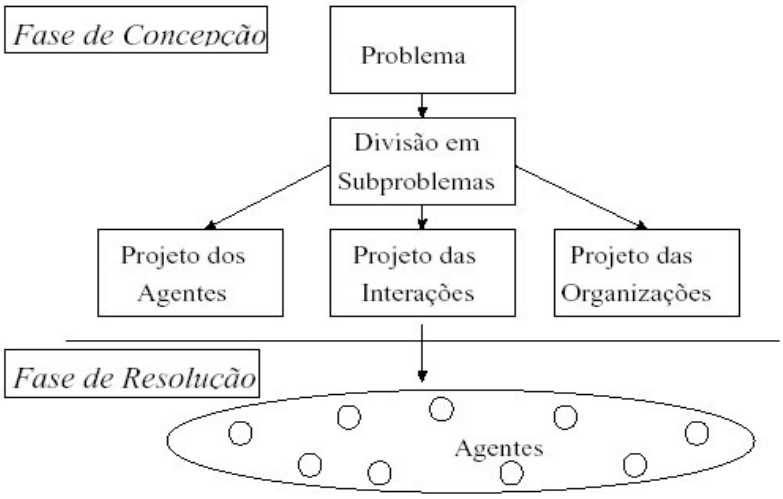
1.2.3.2 Resolução distribuída de problemas (RDP) [4]

Quando se tem a abordagem RDP, observa-se que, a partir de um problema inicial a estratégia de solução apresenta as seguintes características:

- O problema é resolvido por um conjunto de agentes concebidos para solucionar este problema em particular.
- Uma organização é concebida para restringir o comportamento destes agentes. A organização é definida durante a fase de concepção do sistema.
- A interação entre os agentes relaciona-se ao modelo de solução e ao problema a resolver. Este interação pode ser direta, por troca de mensagens, ou indireta, por compartilhamento de dados comuns.
- A execução dos agentes ocorre em modo concorrente visando acelerar o processo.

- Os agentes podem cooperar, dividindo o problema em subproblemas menores ou podem aplicar estratégias diferentes de solução para uma mesma tarefa caracterizando a competição.
- O sistema todo pode ser implementado de forma centralizada, com um agente centralizador controlando outros agentes, ou distribuída, sem um agente controlador.

Em RDP a concepção dos agentes, a sua organização e suas interações são realizadas quando há um problema a solucionar. Não há preocupação com a reusabilidade dos agentes.



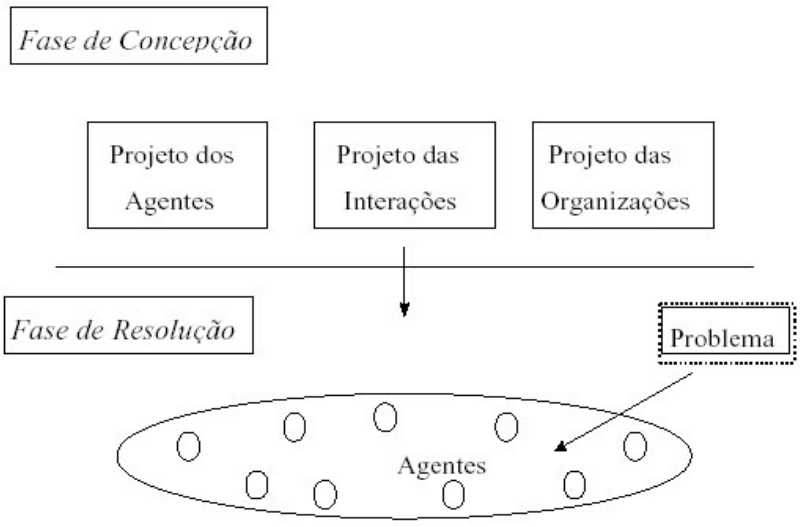
A abordagem RDP segundo [4]

1.2.3.3 Sistemas Multiagentes

Em SMA os agentes têm uma existência própria, independente de outros agentes. Não existe um problema definido *a priori* que o sistema deva resolver.

Em SMA:

- Os agentes são concebidos independentemente de um problema em particular.
- Busca-se desenvolver protocolos de interação genérico que possam ser reutilizados em aplicações similares.
- Não existe uma noção de controle global do sistema.
- Os agentes instanciam dinamicamente as organizações e interações quando um problema lhes é apresentado.



A abordagem SMA segundo [4]

1.2.3.3.1 Sistemas multiagentes reativos [4]

A abordagem reativa foi introduzida por [16] no domínio da robótica. Um ambiente simples de robôs reativos foi desenvolvido por [12] na resolução de outro problema clássico da literatura de IA.

Agentes reativos são agentes que não possuem representação de seu ambiente. As informações relativas ao seu comportamento estão no ambiente e suas reações dependem unicamente de sua percepção deste ambiente. Eles não possuem registro de suas ações passadas e nem podem antecipar (planejar) o futuro. Vivem no *aqui e agora*. Embora esta simplicidade os torne *indefesos* isoladamente, quando em grandes grupos adquirem força o suficiente para realizar tarefas complexas. Daí a pergunta: Há necessidade de agentes complexos para realização de tarefas complexas? Não necessariamente. Tome-se o exemplo uma colônia de formigas. Uma formiga apenas não significa nada. Não pode fazer nada. Mas em colônias organizadas elas procuram alimentos, defendem a colônia, cuidam dos ovos e larvas etc. Ou seja, indivíduos simples que, agrupados, podem resolver problemas complexos. O mesmo ocorre com agentes inteligentes reativos. Quando em grande número, como já dito, são capazes de proezas.

Resumindo, segundo [13], em sistemas multiagentes reativos:

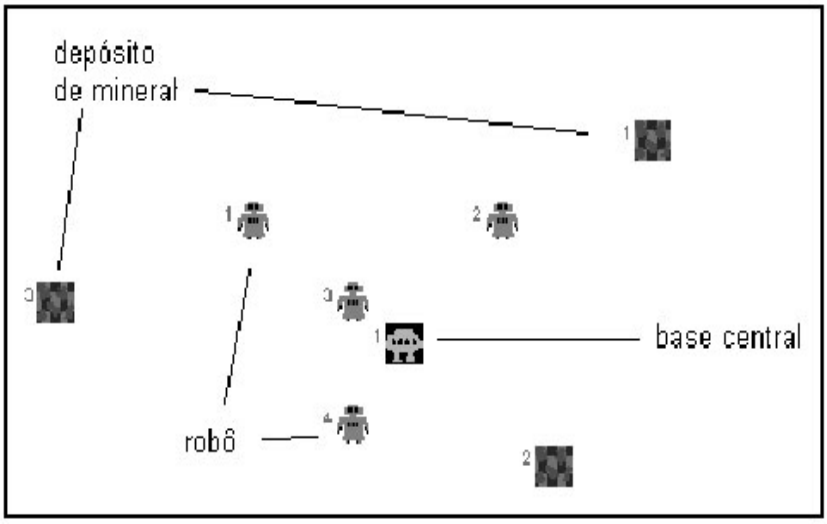
- Não há representação do ambiente – o comportamento se baseia no que é percebido a cada instante.
- Não há memória das ações.
- Organização é etológica – a forma de organização dos agentes reativos é similar a dos animais, em oposição à organização social dos sistemas cognitivos.
- Há grande número de membros.

Modelos de sistemas multiagentes reativos: Modelo da funcionalidade emergente e Modelo Paco.

1.2.3.3.1.1 Modelo da funcionalidade emergente

O modelo da funcionalidade emergente é apresentado em [17]. Este modelo utiliza idéias da arquitetura de subsunção *(subsunction architecture* [16]*)* que estabelece prioridades entre a execução dos comportamentos atribuídos aos agentes.

A idéia básica da arquitetura de subsunção é a decomposição do controle em camadas ou módulos que correspondem a comportamentos elementares. Pode-se considerar a estrutura da sociedade de agentes como a sua organização, e a realização das tarefas, por parte dos agentes, como a sua funcionalidade. Um exemplo clássico do modelo da funcionalidade emergente é o dos robôs mineradores [17]. Este problema consiste em um conjunto de robôs cujo objetivo é encontrar e levar para uma base central amostras de minerais. A localização das amostras e o ambiente são desconhecidos, mas tipicamente as amostras encontram-se agrupadas em pequenos "depósitos".



Elementos do problema dos robôs coletores de mineral

O caminho de volta à base é feito através da suposição de que a base sempre emita um sinal (gradiente) decrescente com a distância que pode ser detectado pelos robôs. Para resolver o problema [17] propôs apenas cinco comportamentos elementares:

1. Evitar obstáculos.
2. Se perceber um mineral e não estiver carregado (já com mineral), pegá-lo.
3. Se perceber a base central e estiver carregado, descarregar.
4. Se estiver carregado, seguir o sinal da base central (na direção do maior gradiente).
5. Realizar movimento randômico.

As simulações feitas por [17] mostram que os robôs realizam a tarefa de levar todos os minerais para a base central.

O modelo de prioridades de tarefas foi utilizado em larga escala nas simulações deste trabalho como pode ser conferido mais adiante.

1.2.3.3.1.2 Modelo da Eco-resolução

O modelo da Eco-resolução [18] define um problema como uma população de agentes cujo conjunto de comportamentos tende a atingir um estado estável, que é chamado de solução do problema.

Cada agente dispõe de um conjunto de comportamentos elementares predefinidos que o levam à procura de um estado de satisfação. Na procura por satisfação os agentes podem ser incomodados por outros agentes. Neste caso, eles agridem os que os estão incomodando, os quais são obrigados a fugir. Na fuga os agentes podem ser levados a agredir outros agentes que eventualmente os atrapalhem. Esta operação prossegue até que os agentes que estão atrapalhando se desloquem efetivamente.

Um eco-agente pode ser caracterizados por:

a) um objetivo, atingir um estado de satisfação
b) um estado interno: satisfeito, em busca de satisfação ou em fuga.
c) ações elementares.
d) uma função de percepção de quem o está incomodando.

De forma geral um eco-agente tem dois comportamentos gerais: vontade de estar no estado de satisfação e obrigação de fugir. Um exemplo do funcionamento do modelo da eco-resolução é o quebra-cabeça de 8 que pode ser encontrado em [4].

As idéias do modelo da eco-resolução foram utilizados nas simulações deste trabalho no sentido de permitir que um agente ao encontrar outro sempre se desvie e no sentido de elaboração das condições de parada dos sistema.

1.2.3.3.2 Sistemas multiagentes cognitivos

Os sistemas multiagentes cognitivos são baseados em modelos organizacionais humanos, como grupos, hierarquias e mercados [4]. As suas principais características [4] e [19] são:

- mantêm uma representação explícita de seu ambiente e de outros agentes da sociedade.
- podem manter um histórico das interações e ações passadas.
- a comunicação entre os agentes é feita pelo modo direto, através do envio e recebimento de mensagens.
- seu mecanismo de controle é deliberativo, ou seja, os agentes raciocinam e decidem sobre quais objetivos devem alcançar, que planos seguir etc.
- seu modelo de organização, como já dito, é baseado em modelo sociológicos.
- uma sociedade contém tipicamente poucos agentes.

Neste trabalho não foram utilizadas técnicas de sistemas multiagentes cognitivos.

1.2.3.4 Considerações sobre RDP e SMA

RDP e SMA são complementares. Na maioria das vezes é praticamente impossível discernir se um dado sistema foi construído segundo a abordagem RDP ou SMA.

Os sistemas multiagentes são classificados como reativos ou cognitivos. Ambos são apresentados nas próximas duas seções.

1.2.3.5 Problemas abordados em IAD

Atualmente os principais problemas abordados em IAD são:

- Descrição, decomposição e alocação de tarefas: como descrever e decompor uma tarefa complexa e como executar estas sub tarefas;
- Interação, linguagem e comunicação: como definir uma linguagem de comunicação, como exprimir as interações entre os agentes etc.
- Coordenação, controle e comportamento coerente: como projetar o controle de um sistema.
- Conflitos e incerteza: como detectar e resolver conflitos e como lidar com dados incertos
- Linguagens e ambientes de programação: computacionalmente que linguagem é mais propícia ao desenvolvimento de uma aplicação em especial.

1.3 O protocolo genérico de comunicação KQML

A Linguagem de Comunicação de Agentes (*Agent Communication Language – ACL*) desenvolvida pelo projeto *Knowledge Sharing Effort*, que ´e financiado pela agência americana ARPA foi a primeira tecnologia para comunicação entre agentes de software bem difundida a incluir alguns dos conceitos complexos de comunicação de alto nível provenientes da literatura de IAD. Ela foi concebida como um padrão para interações entre agentes objetivando permitir interoperabilidade de sistemas de múltiplos agentes; ACL tem também o objetivo de permitir a interoperabilidade com sistemas legados através do conceito de *mediação*

Para facilitar a implementação da ACL foi criada a KQML, *Knowledge Query and Manipulation Language*, uma linguagem para adicionar contexto (intencionalidade) às mensagens. (KQML).

KQML é baseada na teoria dos atos de fala; a linguagem é formada de um conjunto extensível de *performativas* que determinam as interações entre os agentes. As "performativas" KQML se referem, de fato, às *forças ilocucionárias* e explicitam a intenção do agente remetente da mensagem. Por exemplo, uma mensagem que é uma afirmação (`tell`), tem o objetivo de alterar as *crenças* do agente destinatário, enquanto que uma mensagem que é uma solicitação (`achieve`), almeja mudar os *objetivos* do agente destinatário.

KQML inclui também uma arquitetura para plataformas de comunicação entre agentes. Esta arquitetura está baseada na presença de *facilitadores* que são agentes que tem conhecimento sobre quais agentes estão acessíveis e quais são as habilidades de cada um. Isto permite que sociedades de agentes funcionem de forma *aberta*, ou seja, que agentes entrem e saiam da sociedade. Se um agente quiser entrar em uma sociedade, basta avisar o facilitador a respeito de suas habilidades e qual o seu endereço físico na rede. Através de consultas ao facilitador, todo agente pode localizar outros agentes com os quais seja interessante interagir.

Finalmente, KQML induz a um conjunto de protocolos. Estes protocolos são seqüências típicas de mensagens que são normalmente trocadas entre agentes a partir de uma interação iniciada por algum agente pelo envio de algum tipo específico de mensagens. A linguagem KQML foi concebida com o objetivo de se tornar um padrão de comunicação entre agentes.

O formato básico de uma mensagem KQML é o seguinte:

```
performativa (  :sender <word>
                :receiver <word>
                :language <word>
                :ontology <word>
                :content <expression> )
```

A seguir, serão descritas as principais performativas da linguagem KQML. Estas performativas são divididas em diversas categorias. Nas descrições abaixo, *S* denota o agente que envia a mensagem *(sender)*, e *R* denota o agente que recebe a mensagem *(receiver)*.

Performativas de informação:

- **`tell`**: *S* informa *R* que a sentença (ou conteúdo) desta mensagem é verdadeiro para *S*, ou seja, a sentença está na base de conhecimentos de *S*.
- **`deny`**: *S* informa *R* que a negação do conteúdo da mensagem é verdadeira para *S*.
- **`untell`**: o conteúdo da mensagem não está na base de conhecimentos de *S*.

Performativas de consulta:

- **`ask-if`**: *S* quer saber se o conteúdo desta mensagem é verdadeiro para *R*.
- **`ask-all`**: *S* deseja todas as instâncias da mensagem que são verdadeiras para *S*.

Respostas básicas:

- **`error`**: *S* indica a *R* que não compreendeu a mensagem recebida anteriormente.
- **`Sorry`**: *S* diz a *R* que compreende sua mensagem, mas não pode prover uma respota.

Performativas de bases de dados:

- **`insert`**: *S* pede para *R* acrescentar o conteúdo da mensagem na base de conhecimentos de *R*.
- **`delete`**: *S* pede para *R* deletar o conteúdo da mensagem da base de conhecimentos de *R*.

Definição de capacidades:

- **`advertise`**: *S* quer que *R* saiba que *S* pode e processará mensagens do tipo que está no conteúdo desta.

Performativas de efetuação:

- **`achieve`**: *S* solicita que *R* tente fazer o conteúdo da mensagem vir a ser verdadeiro.
- **`unachieve`**: *S* quer reverter a ação de um *achieve* enviado previamente.

Performativas de rede:

- **`register`**: *S* anuncia para *R* sua presença e nome simbólico associado com seu endereço físico.
- **`unregister`**: cancela um register feito anteriormente.
- **`transport-address`**: *S* anuncia um novo endereço físico na rede.
- **`forward`**: *S* quer que *R* passe a mensagem para todos agentes que *R* conhece.

Performativas de facilitação:

- **`broker-one`**: *S* pede a *R* para achar uma resposta para a executiva que está no conteúdo desta mensagem.
- **`recommend-one`**: *S* pede a *R* para sugerir um agente que possa processar seu conteúdo.

Neste trabalho desenvolveu-se uma adaptação do protocolo KQML. Foi feito um módulo KQML para comunicação entre os agentes de limpeza. Isto pode ser observado no modelo 6 (seis) *RandKQML* descrito no item 4.6.

2. Objetivos deste trabalho

Esta pesquisa procurou propiciar ao autor um contato com as técnicas de implementação de inteligência artificial em máquinas, permitindo uma iniciação à metodologia do desenvolvimento científico. Provendo capacidades de descobrir a importância da pesquisa, das referências bibliográficas, do projeto, da implementação e da validação dos resultados atingidos. Procurou-se dar ênfase ao estudo das técnicas de arquitetura para inteligência artificial distribuída, desenvolvimento de arquiteturas para inteligência artificial distribuída, desenvolvimento de protótipos que implementassem a/as arquiteturas propostas.

Finalmente, nesse trabalho, buscou-se simular algumas das arquiteturas do cérebro. Para isto foi feito um estudo simplificado do funcionamento desta máquina fascinante bem como um estudo aprofundado das técnicas de inteligência artificial investigando o que há de mais concreto na interpretação artificial dos processos naturais do cérebro.

Todos os conceitos julgados necessários ao entendimento deste trabalho são brevemente abordados ao longo do texto. Com certeza, materiais mais completos podem ser encontrados nas referências.

3. Metodologia do trabalho

Considerando que os conceitos básicos já foram introduzidos, doravante só serão explicados termos ainda não apresentados.

Dado as arquiteturas que o cérebro utiliza para resolver os problemas que lhe são apresentados escolheu-se duas em específico como já dito na introdução deste trabalho. Para haver a simulação escolheu-se um problema clássico da literatura de inteligência artificial: *dirty catch* [13]. Este problema consiste em se instanciar robôs em um ambiente e fazer com que estes percorram o ambiente em busca de sujeiras recolhendo-as caso as encontre.

Este é um problema trivial quando apresentado a um ser humano. Todavia quando "mapeado" em computador pode tornar-se complexo.

Tendo escolhido o problema escolheu-se o ambiente de desenvolvimento das regras de conhecimento. Como este trabalho é didático foi escolhido o ambiente JESS (*Java Expert System Shell*) [14]. Conseqüentemente a linguagem de programação selecionada foi o JAVA [15].

Observação:

i) *Inicialmente na proposta do trabalho o ambiente de desenvolvimento de regras proposto foi o JATLITE [20]. Posteriormente verificou-se que tal pacote tivera seu desenvolvimento descontinuado. Deste modo o JATLITE foi deixado de lado. Outra razão para a substituição deveu-se ao fato de o JESS [14] disponibilizar seu código fonte junto à distribuição bem como possuir um bom suporte técnico por parte de seus desenvolvedores.*

ii) *Para que o pacote JESS suportasse processamento paralelo e distribuído foram feitas pequenas modificações no código fonte do pacote que é de livre distribuição. Para o relatório final pretende-se colocar as classes que foram modificadas.*

Após a escolha das ferramentas foram desenvolvidas 6 (seis) simulações. Uma usando primitivas de resolução apenas simulando a resolução do problema com um agente tal como o cérebro utilizaria apenas um processo. Três simulam a arquitetura centralizada de processos e uma simula a arquitetura distribuída. Finalmente a sexta simulação propõe um protocolo genérico de comunicação dos agentes: KQML.

Em todas as simulações o termo agente foi estendido para robô. Por robô entende-se a união de uma base de conhecimentos e seus atuadores, responsáveis por suas ações no ambiente como a locomoção.

Na próxima seção são apresentadas as 6 (seis) simulações. Mais informações sobre estas podem ser encontradas nos anexos.

Os passos seguidos foram os seguintes:

i) Coleta bibliográfica.
ii) Estudo de arquiteturas de Inteligência Artificial Distribuída.
iii) Especificação de requisitos para o desenvolvimento da arquitetura.
iv) Análise detalhada para o desenvolvimento da arquitetura.
v) Projeto da arquitetura utilizando-se dos diagramas de classe e de seqüência.
vi) Depuração e melhoria no projeto da IAD.
vii) Implementação da arquitetura.
viii) Teste de Validação da arquitetura.
ix) Testes de Verificação da arquitetura.
x) Melhoria do modelo e da implementação.
xi) Construção da base de conhecimento.
xii) Simulação da arquitetura e a comunicação entre os Agentes.
xiii) Integração entre os Agentes e as inferências produzidas.
xiv) Testes finais do protótipo gerado.
xv) Documentação.

Cada tópico está explicado nos anexos onde foi explicitado o andamento da pesquisa.

4. Resultados e discussão

Apenas para simplificação o ambiente em todas as simulações foi representado como uma matriz quadrada *NxN* onde *n* denota o número de linhas ou colunas.

Em todas as simulações tudo é tratado discretamente, ou melhor, por pontos inteiros. Uma mini representação do ambiente encontra-se abaixo:

0:0	0:1	0:2
1:0	1:1	1:2
2:0	2:1	2:2

0:1 quer dizer Linha 0, Coluna 1

Se o robô está totalmente no centro (em destaque) diz-se que ele ocupa a posição 1:1, ou seja, linha 1, coluna 1.

Para representação dos 5 (cinco) robôs foi adotado o modelo da funcionalidade emergente utilizando as idéias da arquitetura de subsunção. Também foram abstraídos alguns princípios da modelagem segundo a eco-resolução.

Nas simulações em que aparecem mais de um robô no ambiente é necessário dizer que cada robô tem sua consciência própria, i.e., tem sua própria base de conhecimentos e princípios de *raciocínio*. Deste modo todas as decisões de um robô não afetam decisões dos companheiros. Para que um robô saiba de alguma decisão por parte de um companheiro é necessário que ambos se comuniquem através de seus módulos de comunicação quando implementados.

4.1 Robô *alone* não deliberativo (RAND)

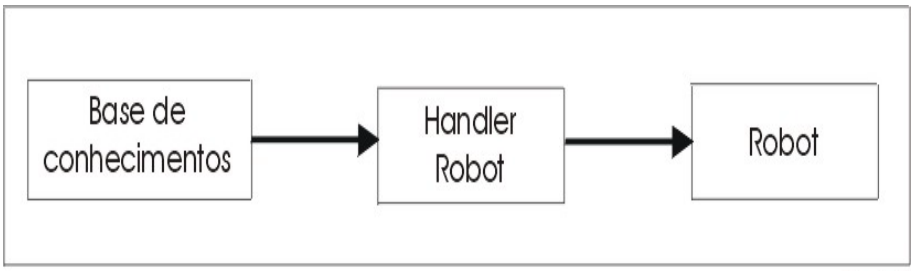
O nome robô *alone* foi dado devido ao fato de o robô atuar sozinho (do inglês *alone)* neste ambiente. O termo "não deliberativo" significa que não há raciocínio lógico na execução das tarefas apresentadas ao robô. Este robô é determinístico porque sempre tem o mesmo caminho na procura da solução do problema.

O RAND foi desenvolvido utilizando 6 (seis) classes escritas na linguagem Java mais a base de conhecimento escrita em Jess. Esta base de conhecimento foi escrita segundo regras simples.

As regras são basicamente as seguintes:

1) Regra *existe sujeira*: se a posição onde o robô estiver possuir sujeira limpar.
2) Regra *não existe sujeira*: caso a posição onde o robô estiver não possuir sujeira seguir em frente.
3) Regra *andar*: dividida em 4 regras menores
    a. Seguir norte
    b. Seguir sul
    c. Seguir leste
    d. Seguir oeste
4) Regra *virar*: divida em 4 regras menores:
    a. Virar norte
    b. Virar sul
    c. Virar leste
    d. Virar oeste
5) Regra *determinar final*: ativar quando o robô percorrer todo o ambiente.

Tudo o que é decidido pela base de conhecimentos do robô é passado aos seus atuadores. Isto é feito através da classe *HandlerRobot* que faz o intercâmbio com a classe *Robot* local onde ficam os atuadores do robô.



Robô

Dado, por exemplo, que a regra *seguir sul* foi ativada o *HandlerRobot* captura esta ativação e aciona os atuadores da classe *Robot* que são responsáveis pela efetivação do movimento. Para que se possa assistir aos eventos criou-se a classe *GUIBot* que juntamente com a classe *Painel* desenham a interface gráfica para a aplicação de simulação. A classe *ItemHandler* é responsável por capturar ações do usuário. E, finalmente, a classe *ConfigRobot* permite a configuração do ambiente tais como as posições onde haverão sujeiras.

Dado que o robô andou ele avisa a *GUIBot* sua ação que por conseguinte desenha na tela o movimento.

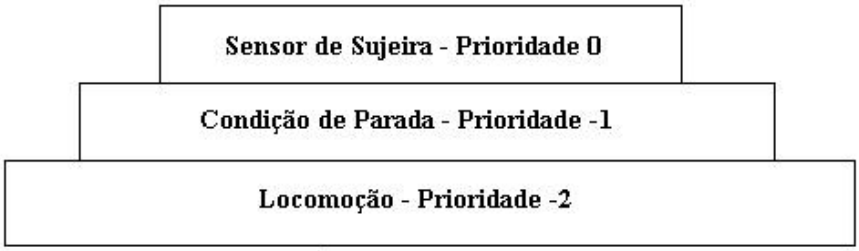
O determinismo do RAND pode ser verificado no algoritmo adotado para sua locomoção. Eis o algoritmo:

- Iniciar o robô na posição 0:0, ou seja, linha 0, coluna 0.
- Seguir em frente se não houver obstáculos.
- Caso haja algum obstáculo vire 90º e continue.
- Marque cada posição visitada.
- Caso haja sujeira na posição atual limpe-a.
- Caso esteja no centro do ambiente finalize.

O resultado da execução deste algoritmo é:

Algoritmo de locomoção do RAND
Espiral para o centro da matriz de ambiente

Para que o RAND funcione sem conflitos as regras precisam seguir algumas prioridades de execução. Deste modo as regras de movimentação *andar* e *virar* têm prioridade -2. A regra *determinar final* tem prioridade -1. A regra *existe sujeira* tem prioridade 0. Quanto maior o número maior a prioridade. Neste caso, a regra *existe sujeira* tem a maior prioridade de execução.



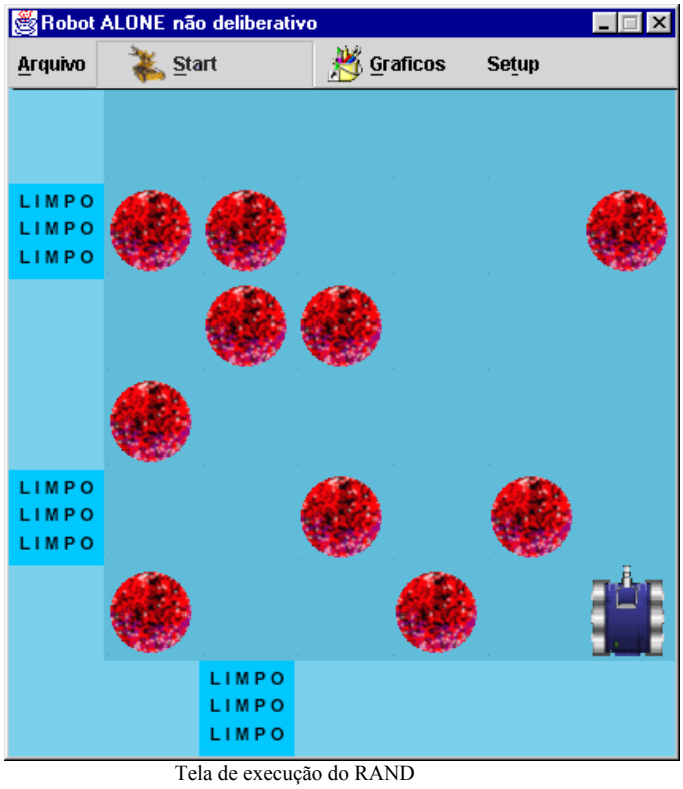
Pirâmide de prioridades

Testes realizados nas seguintes máquinas:

Descrição do hardware	**Sistema Operacional**
Pentium Celeron 350 MHz, 128 MB RAM.	*Windows NT Server*
AMD K6-II 550 MHz, 256 MB RAM	*Windows 98 SE*
AMD K6-II 550 MHz, 256 MB RAM	*Conectiva Linux 7.0*
Pentium II, 500 MHz, 128 MB RAM.	*Windows NT Workstation*
Pentium II, 500 MHz, 128 MB RAM.	*Red Hat Linux 7.1*

Todos os testes foram isolados. Não houve comunicação entre as máquinas

Ficou provado o funcionamento do algoritmo testado atuando como era esperado em teoria. As sujeiras foram todas recolhidas pelo RAND. Isto se deve ao fato de seu movimento ser determinístico e, principalmente, devido à visita de todas as posições da matriz de ambiente.



Tela de execução do RAND

4.2 Robôs não conscientes (RNCs)

Nesta simulação objetivou-se mostrar os principais problemas encontrados ao se tentar resolver um problema com mais de um agente. Em termos de comportamento cerebral seriam os problemas que aparecem quando o cérebro opta por resolver um problema através de vários processos ao invés de resolver o problema apenas com um processo. O termo *robôs não conscientes* denota que estes robôs não realizam raciocínio lógico algum. A principal diferença dos RNCs para os RANDs é aqueles realizam movimentos randômicos.

As regras foram modeladas da seguinte maneira:

a) Regra *sortear-movimentação*: sorteia uma posição a seguir
b) Regra *obstáculo*: verifica se na posição ocupada pelo robô há sujeira
c) Regra *andar* dividida em 4 (quatro) regras menores:
    a. *Andar norte*
    b. *Andar sul*
    c. *Andar leste*
    d. *Andar oeste*
d) Regra *algum limite*: verifica se há algum limite físico para o robô que o impeça de prosseguir.
e) Regra *determinar final*: verifica se algum limite foi atingido 10 (dez) vezes. Caso isso ocorra o robô pára.

Note que os RNCs só param quando "batem" dez vezes, ou seja, o mecanismo de parada é bastante rudimentar.

Os RNCs apresentam vários problemas interessantes. Dentre esses problemas pode-se citar à infração à lei da física de que dois corpos não ocupam o mesmo lugar no espaço. Devido ao fato de não possuírem nenhum sensor para identificar a presença de outros robôs um robô pode bater com outro em uma determinada posição. Os RNCs apresentam apenas sensores de limites físicos ou seja, limites do ambiente em que estão. Sendo assim é possível, pela rudimentar condição de parada, que um RNC fique "batendo cabeça" em algum limite seguidas vezes até tomar outro rumo de locomoção.

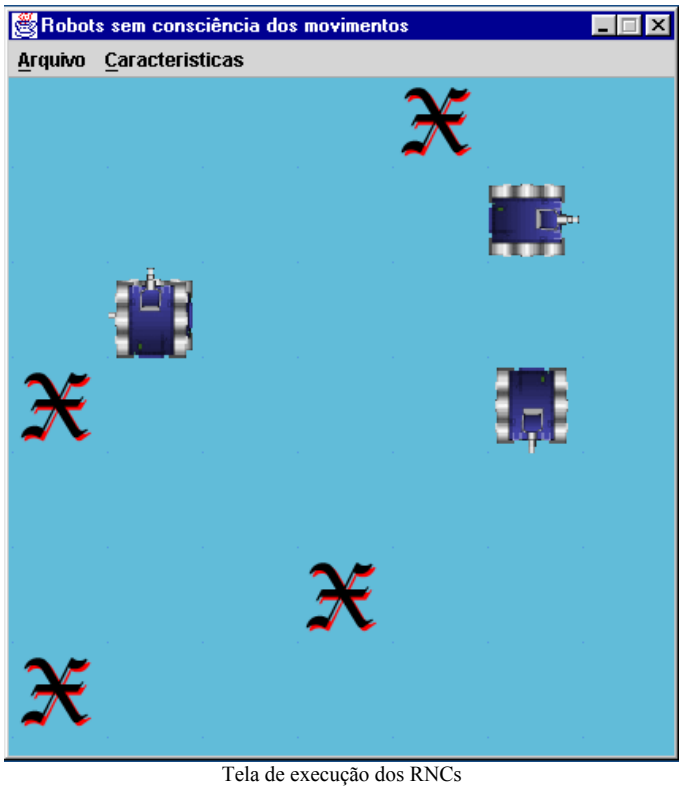
A estruturação das classes é bastante simples. São ao todo 8 (oito) classes. A classe *HandlerRobot* que cuida das decisões da base de conhecimento passando-as para a classe *Robot* que é onde estão os atuadores de cada RNC. Cada classe *Robot* é gerenciada por uma única classe *AgenteCentralizador* que sabe tudo o que há no ambiente. As classes *ConfigAgente e HandlerConfigAgente* são responsável pela caracterização do ambiente tais como número de RNCs, respectivas posições iniciais e posições de obstáculos. A classe *ContainerEstados* é apenas uma estrutura de dados para que o *AgenteCentralizador* possa guardar as informações de cada RNC no ambiente. A classe *HandlerAgente* gerencia as ações do usuário com a interface gráfica que é gerenciada pela classe *AreaGrafica.*

Testes realizados nas seguintes máquinas:

Descrição do hardware	**Sistema Operacional**
Pentium Celeron 350 MHz, 128 MB RAM.	*Windows NT Server*
AMD K6-II 550 MHz, 256 MB RAM.	*Windows 98 SE*
AMD K6-II 550 MHz, 256 MB RAM.	*Conectiva Linux 7.0*
Pentium II, 500 MHz, 128 MB RAM.	*Windows NT Workstation*
Pentium II, 500 MHz, 128 MB RAM.	*Red Hat Linux 7.1*

Todos os testes foram isolados. Não houve comunicação entre as máquinas

Conseguiu-se que os RNCs provassem mesmo sua dita falta de consciência. Sempre batiam nos limites físicos do ambiente não tendo conhecimento suficiente para tomar outros caminhos. Quanto ao recolhimento ou vencimento de posições onde havia sujeiras isto se deu de forma razoável visto que os RNCs não visitam todas as posições bem como não se comunicam com companheiros. Foi verificada, como previsto em teoria também, a superposição de RNCs em determinadas situações devido à falta de sensores de movimento e localização.



Tela de execução dos RNCs

4.3 Robôs semiconscientes (RSCs)

Nesta simulação objetivou-se dotar os robôs de uma semiconsciência na execução das tarefas. Comparando com os processos do cérebro seria quando o cérebro dotasse os seus processos de alguma consciência ainda permitindo certos erros na resolução de um problema. O que torna os robôs ditos semiconscientes é o fato destes não repetirem locais por onde já passaram, i.e, eles passam a ter uma espécie de memória de movimento. Se já fizeram certo percurso não o fazem novamente. Com esta medida os RSCs diminuem em muito o número de colisões com outros robôs mas ainda estão suspensos a cometê-las. As colisões com outros robôs ainda acontecem devido à ausência de comunicação com outros robôs no sentido de todos saberem onde alguém já visitou. Caso cada robô dissesse aos companheiros que visitou um certo local obviamente seus companheiros não visitariam este local novamente. As classes desta simulação são ao todo 8 (oito). São basicamente as mesmas classes dos RNCs salvo algumas modificações para o acréscimo de memória. A principal modificação está na base de conhecimentos, agora escrita tal como:

a) Regra *configurar estado inicial*: prepara a "consciência" do robô para o início da execução no sentido de zerar qualquer dado em sua memória, caso exista.
b) Regra *obstáculo*: verifica se na posição atual há sujeira, se houver o robô a limpa.
c) Regra *sortear:* faz o sorteio da próxima posição a ser ocupada pelo robô. Esta regra torna o movimento dos RSCs randômico.
d) Regra *presente:* verifica se o robô já esteve presente num determinado local. Caso a resposta seja positiva uma nova ativação da regra *sortear* é realizada.
e) Regra *andar*: efetua o movimento de locomoção dado uma resposta da regra *sortear* conjugado com a regra *presente.*

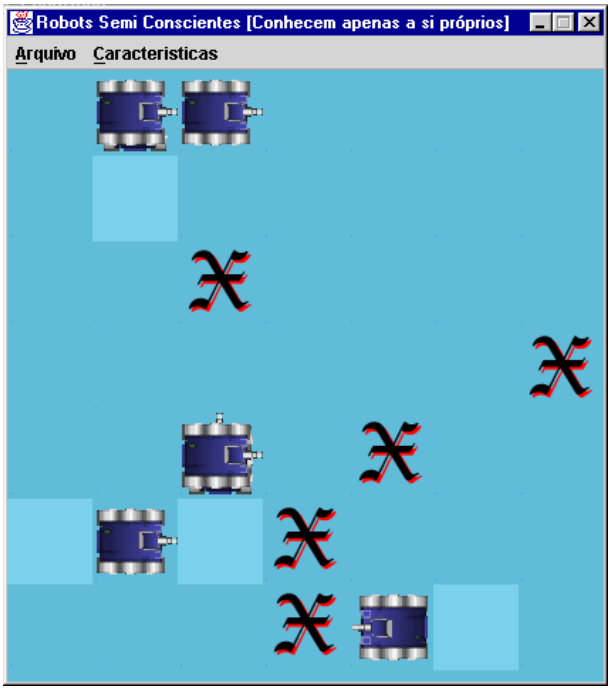
Note-se que a elaboração das regras é muito simples. Logo, a força do sistema de busca de sujeira em todas as simulações está na quantidade de robôs e não na capacidade destes de elaborar planos, visto que, nenhuma simulação prevê tal capacidade.

Testes realizados nas seguintes máquinas:

Descrição do hardware	**Sistema Operacional**
Pentium Celeron 350 MHz, 128 MB RAM.	*Windows NT Server*
AMD K6-II 550 MHz, 256 MB RAM.	*Windows 98 SE*
AMD K6-II 550 MHz, 256 MB RAM.	*Conectiva Linux 7.0*
Pentium II, 500 MHz, 128 MB RAM.	*Windows NT Workstation*
Pentium II, 500 MHz, 128 MB RAM.	*Red Hat Linux 7.1*

Todos os testes foram isolados. Não houve comunicação entre as máquinas

Foi verificado, como era esperado, que os RSCs não visitam a mesma posição da matriz de ambiente mais de uma vez. Isto lhes dá certa consciência. Além disso foi verificado que as superposições caíram drasticamente devido a esta providência. Comprovou-se também o movimento aleatório implementado na consciência dos robôs.



Tela de execução dos RSCs

4.4 Robôs conscientes (RCs)

Nesta simulação objetivou-se resolver o problema *dirty catch* segundo o cérebro o resolveria utilizando a arquitetura de vários processos em execução submetidos a um processo coordenador.

O termo *Robôs conscientes* é devido ao fato destes conseguirem um certo dinamismo em sua atuação no ambiente. O algoritmo de locomoção de cada robô consiste em sortear uma entre quatro possibilidades de movimento. Logo os RCs locomovem-se randomicamente. Devido ao seu sensor de sujeira as posições com sujeira têm maior probabilidade de escolha.

Assim que um robô escolhe uma posição para seu movimento ele avisa seus atuadores para que o movimento seja efetivamente realizado. Ao realizar o movimento o robô avisa um agente central, que tem o controle de todas as operações, que o movimento foi realizado. Ao receber o aviso o agente centralizador avisa a todos os outros robôs do ambiente sobre tal ação de modo que estes não repitam uma ação já realizada por outro robô. Deste modo uma mesma posição não será visitada mais que uma vez e conseqüentemente a verificação de sujeira na mesma também ocorrerá apenas uma única vez. A implementação da base de conhecimentos consiste nas seguintes regras:

1) Regra *obstáculo*: verifica se há sujeira na posição ocupada pelo robô.
2) Regra *sortear*: faz o sorteio da próxima posição do movimento.
3) Regra *andar*: dado o sorteio efetua a movimentação.
4) Regra *limpar verificações*: limpa as variáveis para novas verificações no ambiente.

Nota-se que a inclusão de consciência nos robôs simplifica sua base de conhecimentos diminuindo o número de regras. Na verdade o que houve foi uma expansão do poder das regras, ou seja, estas poucas regras fazem o trabalho de muitas pequenas outras. Ao nível de classes de implementação foram desenvolvidas 8 (oito). A classe *HandlerRobot* liga os acontecimentos da base de conhecimentos à classe *Robot.* A classe *AgenteCentralizador* faz o papel do coordenador dos robôs. Recebe todas as informações, desde informações de movimentos reais e atuação no ambiente até as que apenas solicitam alterações na interface gráfica. As classes *HandlerAgente* e *ConfigAgente* fazem a configuração do ambiente onde serão colocados os robôs. Estas configurações consistem no número de robôs no ambiente, suas respectivas posições de início e as posições de localização de sujeiras. A classe *ContainerEstados* é uma estrutura de dados que serve para o *AgenteCentralizador* guardar a situação de cada robô no ambiente. A classe *ÁreaGrafica* contém as informações para desenho da interface gráfica da aplicação. Finalmente a classe *Robot* contém os atuadores do robô, i.e, as funções para efetivar uma ação dada uma resolução da base de conhecimentos.

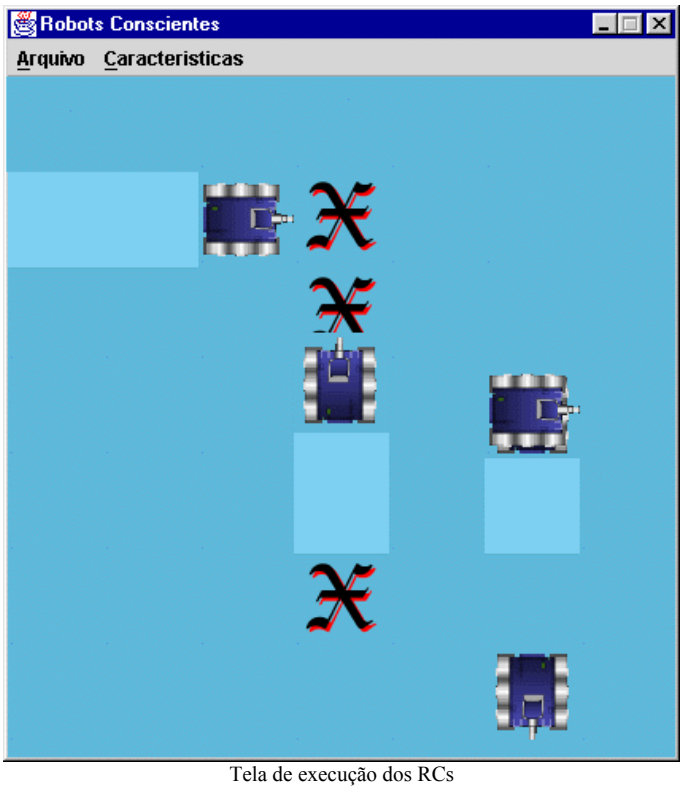
Os RCs param sua execução quando não conseguem mais efetuar nenhum movimento válido, i.e., estão em algum beco sem-saída ou já visitaram toda sua vizinhança. Observa-se, às vezes, que nem todas as regiões com *sujeira* foram limpas. Isto se deve ao fato de que o movimento dos RCs é randômico e que eles não vão a locais já visitados. Desta forma é bem possível que um RC se feche em um canto sem mesmo imaginar que existem muitos outros locais a serem investigados.

Testes realizados nas seguintes máquinas:

Descrição do hardware	**Sistema Operacional**
Pentium Celeron 350 MHz, 128 MB RAM.	*Windows NT Server*
AMD K6-II 550 MHz, 256 MB RAM.	*Windows 98 SE*
AMD K6-II 550 MHz, 256 MB RAM.	*Conectiva Linux 7.0*
Pentium II, 500 MHz, 128 MB RAM.	*Windows NT Workstation*
Pentium II, 500 MHz, 128 MB RAM.	*Red Hat Linux 7.1*

Todos os testes foram isolados. Não houve comunicação entre as máquinas

Verificou-se que os robôs têm bastante consciência. Tem um alto grau de comunicabilidade com seus companheiros através de interações com o agente centralizador. Não há sobreposição de robôs. Os RCs têm movimento aleatório e como era de se esperar em algumas situações não recolhem todas as sujeiras do ambiente pois nem sempre vão em todas as posições. Foi verificado também, que, devido ao sensor de sujeira instalado nos RCs sempre que estes estão passando por uma região próxima de posições de sujeira estes se movimentam em direção à sujeira, aumentando sua eficiência mesmo que o movimento destes robôs seja predominantemente aleatório.



Tela de execução dos RCs

4.5 Robôs conscientes distribuídos (RCDs)

Até o momento, em todas as simulações a comunicação entre os robôs, caso houvesse, era realizada via agente centralizador. Nesta simulação há uma mudança de paradigma. A figura do agente centralizador que tudo vê e tudo ouve deixa de existir. Cada robô é responsável por sua ação bem como seu módulo de comunicação com seus companheiros de tarefa. Esta simulação trata da arquitetura distribuída do cérebro.

A estrutura da base de conhecimentos não mudou muito em relação aos RCs. Ei-la:

a) Regra *configurar-estado-inicial:* prepara o robô para início de execução.
b) Regra *obstáculo:* verifica se a posição atual tem sujeira, se afirmativo, limpa a região.
c) Regra *sortear:* faz o sorteio da próxima posição a ser ocupada. Responsável pelo movimento randômico dos RCDs.
d) Regra *presente:* verifica se o robô já esteve em certa posição
e) Regra *andar:* faz o movimento dado a resposta da regra *presente,* se o robô já esteve em certa posição o movimento em direção a esta posição é mudado.
f) Regra *verificar possibilidades:* verifica uma espécie de memória das mensagens recebidas dos robôs companheiros de modo que este RCD não visite uma posição já visita por outro RCD.

A estrutura de comunicação e troca de mensagens é bastante simples. Utilizou-se a "invocação remota de métodos", um recurso da linguagem JAVA que permite a comunicação entre módulos de programas escritos nesta linguagem em máquinas diferentes. Desta forma, é possível a execução de vários robôs em várias máquinas distintas em uma rede para resolver o problema proposta. Toda a troca de mensagens é feita por TCP/IP logo nenhuma complicação foi encontrada neste quesito. Para escrever os módulos de comunicação acrescentou-se algumas classes ao esqueleto de classes que já havia vindo desde a primeira simulação. A estruturação das classes está como segue: As classes *areaGrafica, MyJFrame* e *configGraf* são responsáveis pela interface gráfica da aplicação. A classe *robotImpl* é o robô propriamente dito, contém os atuadores do robô, bem como seu módulo de comunicação com os outros robôs do ambiente. A classe *handlerRobot* faz o elo entre a base de conhecimentos do robô e a classe *robotImpl.* O módulo de comunicação da classe *robotImpl* é descrito pela classe *robot.*

Quando cada robô se instancia ele se registra na classe *GgrafRobotImpl* que é o gerenciador gráfico ou o ambiente propriamente dito. Esta classe é responsável por saber quem está no ambiente para poder disponibilizar os endereços *IPs* aos novos robôs. Esta classe recebe também, através da classe *handlerGGraf* as ações do usuário da aplicação. Para que a classe *GgrafRobotImpl* esteja disponível todo o tempo a

qualquer RCD que queira se registrar e trabalhar na busca de solução do problema ela também tem um módulo de comunicação provido pela classe *GgrafRobot*.

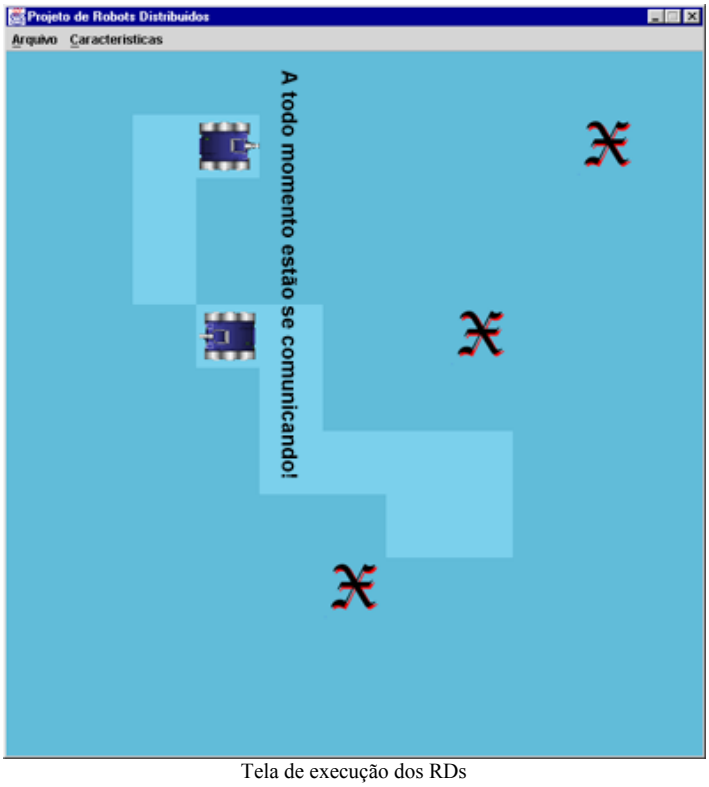
Nenhum problema de sobreposição de robôs encontrado nesta simulação. Dado que o movimento dos RCDs é randômico e que estes não visitam locais já visitados por qualquer outro RCD ou por ele mesmo nem sempre todos os obstáculos *sujeira* são recolhidos. Isto serve para demonstrar que o cérebro humano é mesmo uma caixa de surpresas. Dado um problema tão simples quanto o *dirty catch* e artificialmente não é tão simples resolvê-lo por completo.

Testes realizados nas seguintes máquinas:

Descrição do hardware	**Sistema Operacional**
i)Pentium Celeron 350 MHz, 128 MB RAM	*Windows NT Server*
ii) AMD K6-II 550 MHz, 256 MB RAM.	*Windows 98 SE*
iii) AMD K6-II 550 MHz, 256 MB RAM.	*Conectiva Linux 7.0*
iv) Pentium II, 500 MHz, 128 MB RAM.	*Windows NT Workstation*
v) Pentium II, 500 MHz, 128 MB RAM.	*Red Hat Linux 7.1*

Observação: *Como este foi um modelo distribuído de agentes a simulação foi em grupos de computadores. Foi verificado a execução de vários agentes simultaneamente e se comunicando estando sendo executados máquinas do tipo (iv) e (v). Além disso, foram executados testes com o casamento de máquinas do tipo (i) com máquinas do tipo (ii) e máquinas do tipo (iii). O número de máquinas nos testes foi variado.*

Conseguiu-se testar a comunicabilidade entre os RDs via protocolo TCP/IP. Os RDs são tão eficientes quanto os RCs no quesito busca de sujeiras e movimentação. A diferença aparece na ausência do agente centralizador de modo que, agora, cada unidade robô é independente e é responsável tanto pela sua atuação no ambiente quanto pela comunicação com os companheiros de jornada.



Tela de execução dos RDs

4.6 Robôs *KQML* (RandKQML)

As bases de implementação são as mesmas da simulação 1. Apenas os protocolos de comunicação foram substituídos. Nesta nova abordagem os métodos não são chamados diretamente. Há 3 (três) novas classes: KQMLParser, KQMLCliente, KQMLServidor.

KQMLParser é responsável por receber uma mensagem do Cliente, no caso, o robô e decompô-la de forma que esta possa ser entendida pelo agente centralizador, que, por sua vez, executará os comandos existentes nesta mensagem agora *tokenizada*, ou seja, quebrada em vários comandos.

KQMLCliente é responsável por elaborar mensagens a serem enviadas ao servidor de mensagens.

KQMLServidor é responsável por receber as mensagens de KQMLCliente e trata-las com KQMLParser.

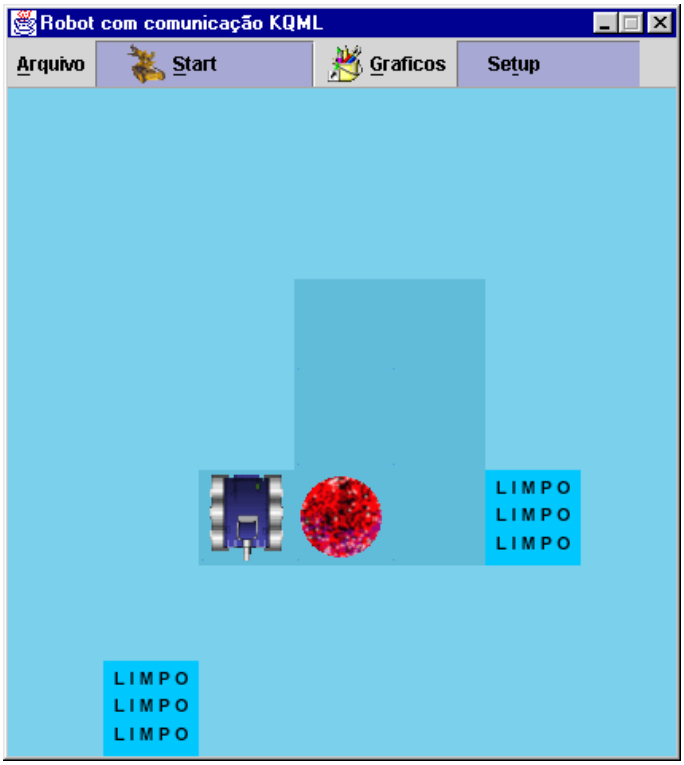
Testes realizados nas seguintes máquinas:

Descrição do hardware	**Sistema Operacional**
i)Pentium Celeron 350 MHz, 128 MB RAM	*Windows NT Server*
ii) AMD K6-II 550 MHz, 256 MB RAM.	*Windows 98 SE*
iii) AMD K6-II 550 MHz, 256 MB RAM.	*Conectiva Linux 7.0*

Observação:

i) Seria bastante interessante se esta simulação ampliasse os módulos KQML para todos as assertivas dispostas no item 1.3.

ii) Os nomes das assertivas foram mudados nesta simulação e simplificados de modo a estarem mais próximos à simulação. Isto não causa danos à generalidade da idéia do protocolo KQML, visto que, ele é livre e pode ser implementado de acordo com as necessidades do desenvolvedor.



Tela de execução do RandKQML

5. Conclusão

Com este trabalho foi possível ao autor deste trabalho ter uma noção do método científico. Proporcionou um contato com a comunidade científica de Inteligência Artificial pelo Brasil e pelo mundo.

Foi possível um grande contato com as mais novas técnicas de implementação e tratamento de problemas em Inteligência Artificial. Aprendeu-se como as máquinas se comportam atualmente, quais os problemas enfrentados etc.

Através do estudo das técnicas de arquiteturas para Inteligência Artificial Distribuída possibilitou-se a ligação entre o que foi estudo sobre o funcionamento cerebral com a possibilidade de simulação utilizando a Inteligência Artificial.

Com a implementação das várias técnicas de arquiteturas de IAD (Inteligência Artificial Distribuída) realizadas foi possível além do conhecimento destas fazer uma divulgação do que foi estudado pelo departamento de computação através de seminários e página na *web* [21].

O mais interessante é que a área pesquisada é muito complexa e pode ser estudada em vários nichos diferentes. Nos anexos encontra-se uma proposta de continuação deste projeto a quem se interessar. São idéias interessantes que podem ser pesquisadas ao longo do tempo.

Finalmente, ainda há muito o que se descobrir na inteligência artificial. Após as simulações desenvolvidas foi possível perceber o quanto se precisa pensar para resolver problemas triviais à mente humana. Falar de inteligência artificial é muito difícil. Quando se resolve falar já se está restringindo o termo inteligência. As máquinas não são capazes de pensar. Pensar como os humanos, diga-se de passagem. Mas, de certa forma, já são capazes de pensar como máquinas. Dado que várias máquinas tenham a capacidade de se comunicar e resolver problemas elas já não estão pensando de alguma forma? É muito difícil predizer até onde tudo poderá chegar, quem sabe nem estaremos aqui quando uma máquina realmente pensar tal como nós seres humanos. Para falar a verdade este autor não gostaria de aqui estar para presenciar tal acontecimento. Caso algo assim, algum dia, venha a se concretizar haverá outros problemas preocupantes: um possível embate ente criatura e criador, máquinas e humanos. Quem será o vencedor? Isto é uma incógnita.

6. Bibliografia

[1] GOLEMAN, Daniel. *Inteligência Emocional: A teoria revolucionária que define o que é ser inteligente*. Rio de Janeiro, Editora Objetiva, 1995.

[2] RICH, Elaine. Knight, Kevin. *Inteligência Artificial*. São Paulo, Makron Books, 1993.

[3] LOVELACE, A. "Notes upon L. F. Menabre's sketch of the Analytical Engine invented by Charles Babbage". In: *Charles Babbage and His Calculating Engines*. Dover, New York, ed. P. Morrison e E. Morrison, 1961.

[4] SICHMAN, Jaime Simão. Alvares, Luis Otávio. *Introdução aos Sistemas Multiagentes.* Anais da Jornada de Atualização em Informática – JAI 97. São Paulo, 1997, editora da Sociedade Brasileira de Computação, SBC.

[5] SICHMAN, Jaime Simão. Demazeau, Y. e Boissier, O. *When can knowledgebased systems be called agents?*. In: *Anais do IX SBIA*, p. 172-185. Rio de Janeiro, 1992.

[6] CHANDRASEKARAN, B. Natural and social system metaphors for distributed problem Solving: introduction to the issue. In: *IEEE Transactions on Systems, Man and Cybernetics*, v. 11, n. 1, p. 1-5, Jan. 1981.

[7] FERREIRA, Aurélio Buarque de Holanda. *Dicionário da língua portuguesa*. Rio de Janeiro, Editora Nova Fronteira, 1993

[8] DEMAZEAU, Y. From interactions to collective behaviour agent-based systems. In: *Proc. of the 1st. European Conference on Cognitive Science*, St. Malo, France, 1995.

[9] PLEIAD. Vers une taxinomie du vocabulaire pour les systèmes multi-agents. In: *Actes de la 1ère. Journée Systèmes Multi-Agents du PRC-GDR Intelligence Artificielle*, Nancy, France, 1992.

[10] GASSER, L. *Boundaries, identity and aggregation: plurality issues in multiagent systems*. Werner, E. e Demazeau, Y. eds. *Decentralized AI 3*, Amsterdam, Elsevier, 1992.

[11] SICHMAN, Jaime Simão. *Du raisonnement social chez les agents: une approche fondée sur la théorie de la dépendance*. Thèse de Doctorat de l'INPG, Grenoble, France, 1995.

[12] FROZA, R. *SIMULA – um ambiente para o desenvolvimento de sistemas multiagentes reativos.* Dissertação de Mestrado, CPGCC/UFRGS, Porto Alegre, 1997.

[13] WEISS, Gerhard. *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT Press, 1999.

[14] SANDIA LABORATORIES. *Jess, the Expert System Shell for the Java Platform.* 10 de agosto de 2001, http://herzberg.ca.sandia.gov/jess/

[15] THE SUN MICROSYSTEMS. *The Java Tutorial*. 10 de agosto de 2001, http://java.sun.com/docs/books/tutorial/

[16] BROOKS, R. A robust layered control system for a mobile robot. In*: IEEE Journal of Robotics and Automation*, v. 2, n. 1, p. 14-23, march 1986.

[17] STEELS, Luc. Cooperation between Distributed Agents through Self-Organization. In: *Decentralized A.I.* Demazeau, Y. e Muller, J-P. (Eds), North-Holland, Amsterdam, 1990.

[18] FERBER, Jacques. Jacopin, Eric. *The Framework of Eco-Problem Solving*. In: Demazeau, Yves; Muller, Jean-Pierre (Eds.). *Decentralized AI 2*, Amsterdam: North-Holland, 1991.

[19] FERBER, J. e Gasser, L. *Intelligence artificielle distribuée*. In: XI International Workshop on Expert Systems & their Applications, Avignon, France, 1991, Cours n. 9.

[20] STANFORD UNIVERSITY. *JATLite Introduction*. 20 de setembro de 2001, http://java.stanford.edu/

[21] ROCHA, Anderson de Rezende. *Opiniões, artigos e publicações.* 15 de abril de 2002, http://www.comp.ufla.br/~undersun

[22] DUMMETT, Michael, *O Significado Filosófico do Teorema de Gödel*, in *O Teorema de Gödel e a Hipótese do Contínuo.* Lisboa, 1977, Organização de M. S. Lourenço, Fundação Calouste Gulbenkian.

[23] TURING, Allan M. *Computing Machinery and Intelligence*. Journal of the mind association, vol LIX, nº 236. Oxford University Press, 1950.

7. Anexos I

Durante o desenvolvimento das cinco simulações propostas vários problemas apareceram. A seguir estão alguns deles, bem como propostas de solução posteriormente implementadas nos modelos finais.

a) **RAND**: Para desenvolver o RAND o principal problema foi conhecer a linguagem de representação de bases de conhecimentos JESS. Feito isso apareceram problemas de mais alto nível. Dentre alguns estão:
   a. o robô não respeitava a decisão da base de conhecimentos.
   b. saía dos limites do ambiente, não andava por todo o ambiente.
   c. não havia um algoritmo eficiente de busca em matriz para a locomoção do robô.

   Para que o robô passasse a obedecer todas as decisões da base de conhecimentos foram tomadas as seguintes providências:
   a. implementou-se um módulo apenas para tratar estas decisões: *handlerRobot*, assim antes de enviar uma mensagem ao atuador do robô a mensagem era tratada e interpretada segundo a linguagem JAVA.
   b. Para que o robô respeitasse os limites do ambiente melhorou-se as regras de movimentação para que estas sempre verificassem se o robô não estava nos limites físicos do ambiente.
   c. Quanto ao algoritmo de locomoção ou busca pela matriz ou ambiente escolheu-se um algoritmo simples: o da busca em espiral, visto que, não era o objetivo aqui testar algoritmos de busca em matrizes.

b) **RNCs**: já que estes robôs não são capazes de raciocinar o mais problemático aqui foi desenvolver uma condição de parada para os mesmos. Isto só foi possível elaborando uma regra especialmente para isso. Como houve uma mudança de paradigma neste ponto deixando de procurar a solução do problema com apenas um robô e passando para vários o problema foi fazer com que cada robô agisse de forma individual, ou seja, tivesse sua própria base de conhecimentos e além de tudo, trabalhasse em paralelo com os demais companheiros. Para que o RNC tivesse individualidade foi preciso apenas modelar sua base de conhecimentos para isso. Para que executasse trabalho em paralelo com os companheiros foi preciso usar um recurso da linguagem JAVA conhecido como *Threads* [15].

c) **RSCs**: A representação da memória das posições visitadas foi o principal problema enfrentado. Dado que cada robô tem sua própria base de conhecimentos como isso poderia ser representado de forma eficiente? Isto foi resolvido criando essa espécie de memória na própria base de conhecimentos do robô. Assim quando um RSC passa por uma posição específica a sua base de conhecimentos trata de memorizar esta posição para evitar que esta seja novamente visitada pelo mesmo RSC.

d) **RCs**: Neste ponto como todos os robôs são ditos *conscientes* nenhum problema é tolerável. Qualquer problema verificado, e, não resolvido, nas simulações anteriores teve de ser resolvido. Dentre estes problemas está o caso das sobreposições de robôs e da comunicação falha entre estes. Para resolver o primeiro problema o agente centralizador passou a saber todas as posições visitadas por qualquer outro RC, assim, se um RC for se movimentar a uma determinada posição este solicita informações ao agente centralizador verificando a possibilidade do movimento. A melhoria da comunicação entre os robôs foi realizada desenvolvendo-se um módulo de comunicação em cada um de modo que estes possam se comunicar apenas com o agente centralizador. Assim quando um RC solicita uma informação ele estará se comunicando com o agente centralizador e com ninguém mais.

e) **RCDs**: Fazer com que um problema seja resolvido de forma paralela não é tão difícil. Mas, fazer tal problema ser resolvido também de forma distribuída pode requerer alguns conhecimentos extras. Foi o que aconteceu nesta simulação. Para que a busca da solução do problema

apresentado ficasse mais interessante simulando também o comportamento distribuído dos processos do cérebro lançou-se mão da computação distribuída. Isto foi possível através do protocolo de comunicação TCP/IP já famoso na *Internet* e do recurso *RMI (Invocação remota de métodos)* da linguagem Java. Um dos problemas verificados aqui foi a falta de um memorizador dos endereços *IPs* dos RCDs registrados no ambiente. Para resolver isto foi desenvolvida a classe *GgrafRobotImpl*. Esta classe também resolveu o problema da atualização da interface gráfica quando qualquer modificação fosse realizada. Assim quando um RCD resolve andar ele avisa todos os companheiros de ambiente que fará tal operação e avisa à interface gráfica via *GgrafRobotImpl* para desenhar o movimento na tela.

f) **RandKQML**: Não houve maiores problemas nesta simulação visto que praticamente houve apenas reutilização de software. O desenvolvimento do protocolo KQML (simplificado) apresentou algumas inconsistências no início em relação "ao semáforos" do sistema operacional, ou seja, estava mandando informações (mensagens) em ordem errada. Resolvido isso (através da sincronização dos processos) não apareceu maiores incidentes.

Anexos II

Andamento e divisão das tarefas durante a pesquisa:

1 - Coleta bibliográfica:

Num primeiro momento foi realizado uma procura de material bibliográfico sobre o cérebro e sua arquiteturas. Foi realizada também uma busca de informações sobre a linguagem JAVA e sobre a linguagem de representação de conhecimentos JESS, visto que, a linguagem JATLITE fora abandonada por motivos já explicados neste documento. Posteriormente, foi feita uma exaustiva busca por materiais de inteligência artificial e de agentes inteligentes para a implementação das idéias propostas.

2 - Estudo de arquiteturas de Inteligência Artificial Distribuída.

Foram estudados os tipos básicos de arquiteturas para escolha das arquiteturas mais apropriadas para o desenvolvimento do protótipo proposto.

3 – Especificação de requisitos para o desenvolvimento da arquitetura.

Uma vez definido a arquitetura a ser utilizada, passou-se para a fase de análise especificando-se os requisitos que deviam ser atendidos em nível de projetos. Ou seja, o que a arquitetura deveria prover, o que ela poderia prover e o que ela não proveria, procurando sempre uma maior aproximação da arquitetura cerebral a ser simulada.

4 – Análise detalhada para o desenvolvimento da arquitetura.

Determinado o escopo do sistema, foram projetadas a estrutura de dados e as rotinas que o sistema deveria conter, de tal forma a atender a especificação de requisitos.

5 - Depuração e melhoria no projeto da IAD.

Vários erros encontrados foram sendo corrigidos com o tempo. As melhorias ainda estão sendo implementadas até hoje.

6 – Implementação da arquitetura.

Nesta etapa criou-se "extra projeto" um *site* para disponibilizar tudo o que estava sendo produzido e pesquisado. Tudo o que foi produzido foi e está sendo disponibilizado na Internet através do site *http://www.comp.ufla.br/~undersun* de modo que todo estudante possa ter acesso às informações coletas e às simulações implementadas. Qualquer pessoa que quiser fazer modificações nas classes desenvolvidas poderá faze-las sem prévia autorização do desenvolvedor, visto que, o projeto foi desenvolvido justamente para estimular o pensamento crítico em relação ao tema proposto.

7 – Integração entre os Agentes e as inferências produzidas.

Nesta etapa partiu-se para e elaboração de vários agentes trabalhando no mesmo ambiente em busca da solução do problema. Estes Agentes foram criados a partir da base de conhecimento gerada. Os Agentes deveriam ser capazes de rodar em máquinas distintas e trocar informações e conhecimento de modo cooperativo para se chegar a um objetivo em comum.

8 - Testes final do protótipo gerado.

Todos os protótipos gerados foram devidamente testados e produziram os resultados esperados.

9 - Documentação.

À medida do andamento do projeto tudo foi documentado nos próprios códigos fonte para que qualquer interessado não tenha maiores dificuldades em utilizar e/ou modificar os códigos produzidos.

Anexos III

Algumas propostas de continuação deste projeto consistem em:

a) Implementar os agentes de forma que quando estiverem sendo executados em uma rede de computadores e algum dos agentes apresentar problemas tais como iminência de desligamento do terminal, onde este se encontra, este imediatamente faça um *backup* (cópia de segurança) de sua base de conhecimentos e instancie um *clone* de si em uma outra máquina que esteja funcionando normalmente. Logo após, este agente deve inserir seus conhecimentos neste *clone*. Deste modo os agentes tornar-se-iam independentes das máquinas onde estivessem sendo executados não tendo mais relações temporais tão íntimas com os terminais.
b) Conectar o robô a uma plataforma de navegação autônoma para realizar testes práticos em ambiente real buscando simular um trator arando o campo cujo motorista é um agente do tipo *Driver Agent.*
c) Agregar um módulo de processamento de imagens e visão computacional ao agente para que o mesmo possa ter mais parâmetros para tomar uma decisão acertada.

Anexos IV

Todas as simulações desenvolvidas estão no CD em anexo. Juntamente, encontram-se este relatório em sua versão eletrônica, formato .PDF, os slides da apresentação do projeto no *Congresso de Iniciação Científica da UFLA*, **CICESAL,** e o pacote de gerenciamento de conhecimentos JESS, o qual, é necessário ao funcionamento de quaisquer das simulações.