Uma Infraestrutura para Experimentos Robóticos Bio-Inspirados em Grades Colaborativas

Lucio Agostinho[1], **Ricardo S. Souza**[1], **Fernando Paolieri**[1], **Eliane G. Guimarães**[2], **Eleri Cardozo**[1]

[1]Faculdade de Engenharia Elétrica e de Computação – FEEC
Universidade Estadual de Campinas – UNICAMP

[2]Centro de Tecnologia da Informação Renato Archer
13083-970 – Campinas - SP

Abstract. *This paper presents an infrastructure for robotic bio-inspired experiments in Computational Grids. This infrastructure supports collaboration for groups of resources that can interact among them in a decentralized model, according to needs of the aggregated resources. An experiment in mobile robotics employing an evolutionary navigation algorithm and a second experiment involving a wireless sensor network are presented in order to illustrate the benefits of the proposed infrastructure.*

Resumo. *Este artigo apresenta uma infraestrutura para experimentos robóticos bio-inspirados em Grades Computacionais. Essa proposta suporta colaboração para grupos de recursos que podem interagir entre si em um modelo descentralizado, de acordo com as necessidades dos recursos agregados. Um experimento em robótica móvel que emprega um algoritmo evolutivo de navegação e um segundo experimento envolvendo uma rede de sensores sem fio são apresentados com o intuito de ilustrar os benefícios da infraestrutura proposta.*

1. Introdução

Uma infraestrutura de Grade Computacional (Grade) oferece um ambiente para compartilhar recursos de armazenagem, processamento de dados e distribuição de tarefas que podem ser processadas por nós geograficamente dispersos. Esses nós compartilham os seus recursos computacionais de forma que tarefas complexas possam ser distribuídas e processadas entre os recursos agregados. A Grade pode ser utilizada tanto para a computação de tarefas complexas que exigem alto poder de processamento quanto para a integração de domínios [Foster and Tuecke 2005] [OGF 2010].

A gerência desses recursos pode ser simplificada com o uso de *toolkits*, que são bibliotecas de software que oferecem serviços para manter o ambiente funcional. Os *toolkits* simplificam o acesso à rede de recursos e possuem interfaces para a comunicação com o ambiente. O *grid middleware* é o software que une a aplicação aos recursos e permite o gerenciamento do sistema [IT Professional 2004]. Muitas aplicações podem compartilhar os recursos oferecidos que, em muitos casos, encontram-se dispersos. A provisão deles depende principalmente de mecanismos de autorização, autenticação, agendamento, visualização de status, contabilidade de uso, entre outros. Os usuários desse ambiente precisam dispor de interfaces compatíveis para interagirem

com o ambiente da Grade. Por questões de segurança a agregação de recursos de domínios diferentes irá depender do acordo estabelecido entre os nós da Grade, o que geralmente é feito com o uso de certificados digitais assinados por Entidades Certificadoras (*Certificate Authority*-CA).

No tráfego de informações inter-domínios deve existir a liberdade de escolha da infraestrutura de *software* mais adequada para cada ambiente. Em virtude disso, a evolução da tecnologia de Grades tornou possível a oferta de serviços sob demanda [Cirne and Neto 2005]. A convergência de tecnologias da área de computação de alto desempenho e de padrões abertos trouxe a necessidade de definir um padrão comum e aberto para aplicações baseadas em Grade [Sotomayor 2005]. Iniciativas do *The Global Grid Forum* [GGF 2010] levaram ao desenvolvimento de Grades segundo o padrão OGSA (*Open Grid Services Architecture*). OGSA especifica um conjunto de interfaces comuns para grande parte dos serviços oferecidos, o *Web Services Resouce Framework* (WSRF) [OASIS 2010], que é uma infraestrutura para a arquitetura OGSA e que estende a capacidade dos *Web Services*, especificando serviços *stateful*. Ainda que o conjunto de ferramentas de *software* seja baseado em padrões abertos, o desenvolvimento de aplicações distribuídas pode ser oneroso. O *Globus Toolkit* [Globus 2010], por exemplo, oferece um conjunto de interfaces para aplicações em Grade, mas o desenvolvimento de serviços complexos e a representação dos recursos agregados ainda exigem a manutenção de um intrincado conjunto de arquivos [Luo et al. 2009].

Por outro lado, a infraestrutura de Grade também suporta diversos tipos de aplicações colaborativas. Como exemplo, a navegação autônoma bio-inspirada de robôs móveis pode se beneficiar do uso da Grade. Esse tipo de experimento exige a avaliação em recursos físicos para adquirir resultados mais precisos. Apesar das facilidades de se utilizar a simulação, experimentos reais fornecem atrativos de grande relevância, uma vez que as circunstâncias de operação no mundo real são difíceis de serem modeladas em cenários virtuais. Considerando o problema específico da navegação autônoma, resultados mais precisos podem ser obtidos em experimentos evolutivos com robôs reais que utilizam Sistemas Classificadores [Cazangi 2008].

Para essa classe de experimentos um agente pode ser definido como um sistema computacional, situado em um dado ambiente, que tem a percepção desse ambiente através de sensores, tem a capacidade de decisão, age de forma autônoma nesse ambiente através de atuadores, e possui capacidades de comunicação de alto-nível com outros agentes e/ou humanos, de forma a desempenhar uma dada função para a qual foi projetado [Reis 2003]. Embora não exista uma definição consensual desse conceito, essa será a definição adotada nesse artigo.

Afirmar que um agente é autônomo significa que ele consegue se auto-regular gerando as próprias regras que regem a sua atuação no ambiente, além de agir por si só [Siegwart and Nourbacksh 2004]. A navegação diz-se bio-inpirada porque é baseada nos mecanismos evolutivos biológicos [Eiben and Smith 2007]. Nesse processo, a evolução é vista como um mecanismo de busca por soluções para resolver um conjunto definido de problemas. Para isso, geralmente utiliza-se uma função de avaliação que define uma superfície de adaptação. O algoritmo bio-inpirado, proposto neste artigo, realiza um processo iterativo de busca, ou seja, de otimização das soluções encontradas em cada passo da evolução das regras do agente. O algoritmo do Sistema Classificador foi proposto para evoluir as opções de movimentação nesse espaço de busca, conduzindo o agente ao longo do ambiente.

Neste artigo é apresentada uma proposta de infraestrutura para robótica colaborativa em Grade com suporte à agregação dinâmica de recursos ao ambiente. O objetivo é oferecer uma infraestrutura interoperável com o ganho de facilitar a oferta de serviços sob demanda. A arquitetura prevê a agregação segura de serviços associados a recursos. Avalia-se a proposta com a experimentação remota em tempo-real da navegação autônoma bio-inspirada, com o uso da versão 5.0.1 do *Globus Toolkit* (GT). Estudos de caso comprovam a viabilidade da arquitetura nos quesitos de segurança, liberdade de uso da infraestrutura de software no domínio, suporte à colaboração em Grade, controle da distribuição de recursos e desempenho da comunicação no processo de telemetria remota em tempo-real.

O artigo esta organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados com a proposta; a Seção 3 descreve a infraestrutura da Grade, detalhando o modelo e a arquitetura; a Seção 4 apresenta o experimento robótico colaborativo de Sistemas Classificadores na Grade; a Seção 5 avalia o algoritmo evolutivo e o desempenho da arquitetura com um experimento que envolve uma rede de sensores. Finalmente, a Seção 6 apresenta as considerações finais.

2. Trabalhos Correlatos

O estudo de Robótica em Grades ainda é pouco explorado na literatura. Em certo ponto, essa área pode ser vista como um tipo de gerência de recursos de vários tipos de agentes robóticos autônomos inteligentes. A Grade pode fornecer recursos quando os robôs não possuem capacidade de processamento suficiente. Outras capacidades são a tolerância a falhas permitindo a computação redundante, além do compartilhamento de recursos robóticos.

A integração segura de recursos entre domínios geograficamente dispersos é um requisito importante que orientou o desenvolvimento da infraestrutura apresentada nesse artigo. Dentre os trabalhos que abordam essa temática está um projeto de arquitetura Robótica de Grade na Internet [Sabatier et al. 2004], que é descrito como uma alternativa para a integração de dois laboratórios de informática entre a França e a Itália. O projeto utiliza o conjunto de bibliotecas GridRPC [Seymour et al. 2002] para o controle remoto distribuído e redundante de um robô autônomo. Outro projeto de arquitetura Robótica em Grade [Breitling et al. 2008] integra uma rede de telescópios robóticos. Essa rede de recursos é capaz de realizar novos tipos de observações que não podem ser realizadas com instrumentos individuais. É empregado o *grid middleware* AstroGrid-D, que é mantido em uma camada de aplicação acima do *Globus Toolkit*, utilizando os comandos do *Grid Resource and Allocation Manager* (GRAM).

Esse artigo foca na integração de recursos robóticos para experimentos colaborativos. A colaboração diz respeito ao trabalho conjunto em uma atividade, especialmente para produzir ou conduzir uma atividade com dois ou mais parceiros [Parker 2010]. *Grades Colaborativas* consistem de muitos participantes que concordam em compartilhar os seus recursos em um ambiente comum. Nessa Grade os recursos são agrupados em federações sob um mesmo domínio administrativo, e essas federações são comumente referenciadas como Organizações Virtuais (*Virtual Organizations* - VOs). Uma VO é primariamente um domínio gerenciável que controla e coordena os serviços e recursos fornecidos por outros domínios, a fim de atingir um objetivo comum [Guillen 2009]. Os membros de uma VO podem ter acesso a uma variedade de recursos agregados ao ambiente por um determinado período [Foster et al. 2002]. Por outro lado,

a Computação *Peer-to-Peer* (P2P), que é um tipo de Computação Distribuída, pode ser confundida com a Computação em Grade. A Computação P2P diferencia-se da Computação em Grade em razão de seus objetivos serem baseados na oferta de serviços especializados, e estritamente relacionados, para milhares de usuários concorrentes [Foster and Iamnitchi 2003].

Diversos projetos propõem o uso de Grades como suporte para a colaboração. Uma abordagem adotada pelo grupo da *Globus Alliance* é o *Java Commodity Grid (CoG) Kit Ad hoc Grid framework* [Laszewski et al. 2010]. Ele é um protótipo que tem o objetivo de simplificar o uso do *Globus Toolkit* e de utilizar essa ferramenta como suporte para a colaboração entre múltiplos domínios. O projeto faz parte do *Java CoG Kit* e pretende ser um mecanismo para acomodar a colaboração na Grade através de ferramentas de gerência e comunicação entre grupos, provisão de serviços, provisão de QoS, descoberta de serviços, invocação de serviços e políticas de autorização de acesso.

Na criação da infraestrutura apresentada nesse artigo, preocupou-se em minimizar o esforço para agregar recursos robóticos, que devem ser mantidos por um *middleware* com funcionalidades reduzidas no domínio do usuário. Um trabalho mais abrangente, mas que ilustra essa preocupação com o suporte de funções no domínio do usuário, é o Grid4All [Krishnaswamy et al. 2008], que utiliza o termo “grid democrático” para definir a sua proposta. O objetivo é envolver instituições de ensino, pequenas empresas, pesquisadores e outros participantes com PCs menos robustos em uma infraestrutura de computação global que permita a colaboração e o trabalho em conjunto. O cenário foca no suporte ao compartilhamento de informações e na colaboração entre os seus membros. O *middleware* Grid4All possibilita o acesso concorrente às informações, a criação de VOs e a adição de recursos à rede. Por meio de interfaces públicas, a alocação de recursos e o espaço de armazenamento pode ser obtido na própria rede. A infraestrutura é baseada em redes *overlay* com técnicas P2P, auto-organização das redes *overlay* com *Distributed Hash Tables* (DHT) e tolerância a altas taxas de entrada e saáda de recursos (*churn*). A proposta também pretende suportar serviços de dados federados.

Outro requisito definido para a infraestrutura apresentada neste artigo é o de utilizar o ambiente para executar um conjunto de serviços distribuídos, mas mantidos dentro do domínio interno da Grade. O usuário deve ser capaz de criar as suas aplicações, combinando os serviços já oferecidos pelo ambiente. Esse requisito é similar ao do projeto PlanetLab [Peterson et al. 2006] quanto à oferta de serviços distribuídos. O PlanetLab é uma plataforma global para desenvolver e avaliar serviços de rede. O projeto fornece uma rede *overlay* que permite a avaliação e a implementação de serviços distribuídos. A arquitetura atende a pesquisadores que desenvolvem serviços de rede e a clientes que desejam utilizar tais serviços. A execução contínua de aplicações é possível porque uma fatia (*slice*) da rede é atribuída para a execução da aplicação. O mecanismo de virtualização distribuído requer a multiplexação dos serviços concorrentes. A gerência da rede é dividida em serviços de gerenciamento independentes, cada um executando em sua própria fatia da rede [Ziviani and Duarte 2005].

A abordagem de Grades Colaborativas é apresentada em outros projetos como o OurGrid [Andrade et al. 2003] e LaCOLLA [Vilajosana et al. 2007]. A Computação em Grade é utilizada em muitos outros projetos de Computação Distribuída relacionados à Biologia e Medicina, Ciências da Terra, Matemática, Física e Astronomia, Arte, Inteligência Artificial e outros [GRID Infoware 2010].

3. Infraestrutura para Colaboração em Grade

Nossa proposta da Grade Colaborativa foi baseada em um modelo capaz de oferecer serviços sob demanda, otimizar o desenvolvimento de novas aplicações e agregar recursos de diferentes domínios à infraestrutura de serviços já oferecida pelo GT. Como mostra a Fig. 1, um conjunto de módulos foi definido para simplificar a agregação de recursos, mantendo a segurança e a escalabilidade. Esse modelo baseia-se na especificação do CoG Kit [Laszewski et al. 2010] ao definir um conjunto de serviços que executam em uma camada de aplicação superior ao GT. O modelo orientou a criação do *Simple Interface with Grid Agent (SIG Agent)* que é uma interface no domínio do usuário que abstrai o desenvolvimento de aplicações de Grade.

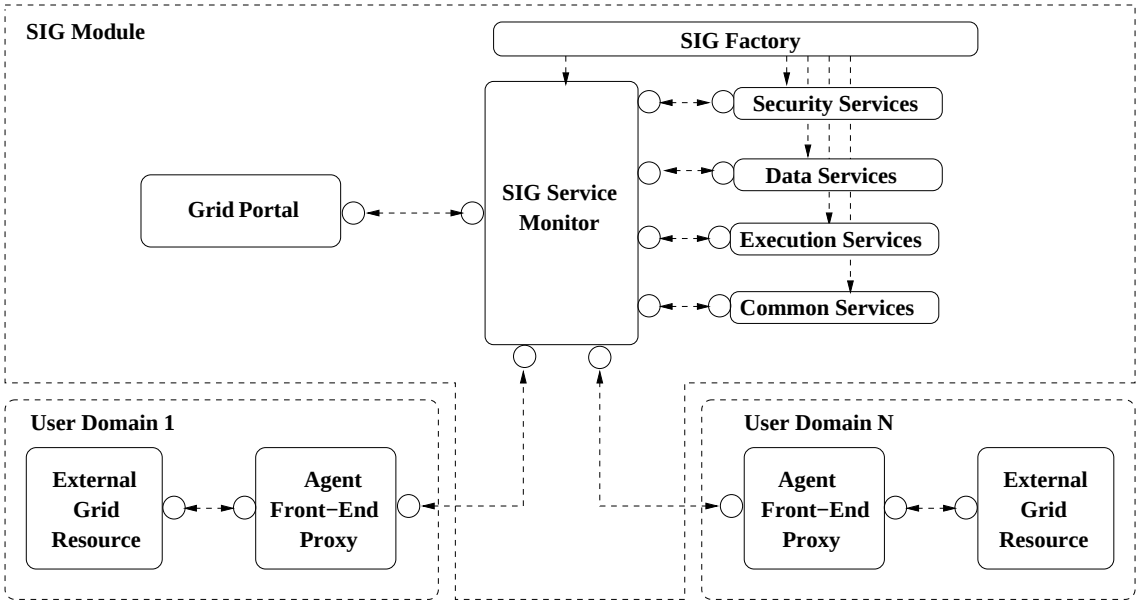


Fig. 1. Modelo para Integração de Recursos em Grade.

A proposta é baseada no modelo do Java CoG Kit mas, diferente dele, fornece uma alternativa mais cômoda para o desenvolvimento de aplicações. Para criar um novo serviço para a Grade, o usuário cria a sua aplicação e a insere na fábrica. O *SIG Agent*, no domínio do cliente, não precisa ser modificado, podendo ser utilizado tanto para acessar os serviços da fábrica quanto para acessar os outros serviços criados pelo usuário. A aplicação do usuário utiliza o *SIG Agent* como uma interface para interagir como conjunto de serviços mapeados para as funcionalidades básicas do GT, tais como: execução de *jobs* (*globusrun*), transferência de arquivos (*gsiftp*), monitoramento de serviços ativos (implementado como um serviço da fábrica), início de proxy (*grid-proxy-init*), finalização de proxy (*grid-proxy-destroy*), entre outros, e para os novos serviços criados pelo usuário. No modelo apresentado foram definidos os seguintes componentes:

- *External Grid Resource*: é o módulo que mantém o conjunto de serviços que controlam o recurso fora do domínio da Grade;
- *Agent Front-End Proxy*: define os parâmetros de acesso ao recurso. Esse módulo atua como uma interface entre a aplicação de controle do recurso e o *proxy* da Grade. Também é responsável por abstrair os comandos do *grid*

middleware nativo do domínio e repassá-los ao serviço solicitado da Grade. Esse módulo permite que recursos sejam adicionados ou removidos sem interferir no funcionamento dos outros recursos;

- *SIG Factory*: permite a oferta de serviços sob demanda. Os serviços de acesso a novos recursos na Grade podem ser instanciados em tempo de execução. Esse módulo instancia os serviços que resolvem a abstração do evento disparado no domínio do usuário. Cada serviço (*Security, Data, Execution, Common*) possui uma interface com os serviços de software do *grid middleware*. Por ser voltado para o GT, esse modelo é em parte similar à proposta deste *grid middleware* porque agrega serviços que devem implementar a interface de acesso às funcionalidades originais do sistema. Os serviços gerados a partir do *SIG Factory* foram:
  a) Módulo *Security Services*: mantém os serviços relacionados ao acesso seguro, tais como validação de *proxy*, emissão de certificados, serviços de delegação, entre outros;
  b) Módulo *Data Services*: contém os serviços que realizam operações de persistência junto ao *grid middleware*;
  c) Módulo *Execution Services*: contém os serviços que se comunicam com as funções de transferência de dados, submissão e escalonamento de tarefas;
  d) Módulo *Common Services*: mantém os serviços que executam serviços específicos de outros ambientes;
  e) *SIG Service Monitor*: define um conjunto de interfaces para os serviços de informação e de monitoramento dos serviços intanciados pelo *SIG Factory*. Esse módulo também monitora os recursos agregados ao domínio. Apesar do MDS (*Monitoring and Discovery Service*) no *Message Service Component* do GT ser capaz de monitorar e descobrir serviços e recursos, o MDS possui dificuldade para descrever recursos não-computacionais [Luo et al. 2009]. O *SIG ServiceMonitor* reconhece que nem toda fonte de informação suporta interfaces *WSRF/WSNotification* e utiliza serviços próprios para resolver essa limitação, mas mantém uma interface com os serviços originais do GT para complementar os serviços originais já oferecidos;
- *Grid Portal*: interface de gerência que permite o acesso seguro do usuário aos recursos agregados.

3.1. Arquitetura para Colaboração em Grade

Os experimentos robóticos são altamente dependentes do desempenho da rede e da precisão dos sistemas de posicionamento e navegação. Por se tratar da interação com agentes móveis, a perda acentuada de pacotes na rede pode levar à acumulação de erros de posição (erros de odometria). Como consequência, o agente será incapaz de atingir o alvo real porque o seu deslocamento no ambiente o levou para uma posição no ambiente diferente da que ele “acredita” ser a posição correta. Isso implica que a infraestrutura deve fornecer um tempo de interação aceitável para atender às requisições do agente.

A arquitetura proposta segue o modelo anteriormente descrito, e é ilustrada na Fig. 2. Para minimizar o esforço para a agregação e interação com os recursos, o *SIG Agent* atua como *middleware* no domínio do usuário e oferece o mapeamento adequado

dos serviços da Grade. Para ampliar a interação com a infraestrutura, o *Grid Proxy* mantém um container de *Web Services* de forma a permitir o acesso de diferentes domínios. A aplicação do usuário pode estar fora dos limites da rede interna da infraestrutura. Por isso, cada usuário precisa ter uma conta na Grade, com os respectivos certificados em seu diretório *home* do *host* de *proxy* (*Grid Proxy*). Os serviços do *SIG Agent* oferecem o mapeamento para o início do *proxy* (*globus-proxy-init*) a partir dos certificados presentes no *Grid Proxy*, e serviços para interação com as demais funcionalidades da Grade. Na arquitetura, os serviços da Grade são mantidos na *SIG Factory*.

O *External Grid Resource* representa o recurso robótico da arquitetura. O *Agent Front-End Proxy* é a interface de acesso, representada pelo *SIG Agent*. Esse componente identifica o recurso, trata o envio e recepção de mensagens e possui a implementação da lógica de acesso ao *Grid Proxy* e ao recurso do domínio do usuário. Dessa forma, o recurso pode ser agregado ao ambiente independente de sua localização, desde que seja fornecido o endereço de *proxy* para o seu domínio. Essa arquitetura suporta o uso de *Web Services* no *Grid Proxy* e no *Grid Portal*. Essa característica é importante porque simplifica o processo de encaminhamento de mensagens entre *firewalls* de domínios distintos, uma vez que são trocadas mensagens XML com o uso do protocolo SOAP (*Simple Object Access Protocol*). O *Grid Proxy* mantém a *SIG Factory* para a instanciação dos serviços de segurança, dados, execução e demais funcionalidades. A arquitetura prevê o uso do GT em todos os nós da rede interna. O *Grid Portal* exibe as informações sobre os recursos que interagem no domínio.

Na arquitetura os *hosts* da Grade mantˆem o relacionamento de confianc¸a com o uso de certificados digitais assinados pela mesma CA. O *Grid Proxy* é o *host* que mantém a fábrica de serviços que abstraem a comunicação com as funções do GT. Portanto, o *Grid Proxy* participa de uma hierarquia na qual tanto o tráfego de ingresso quanto de egresso é filtrado antes de entrar e depois de sair do domínio da Grade, ou seja, apenas são admitidas submissões de *jobs* e o uso de serviços da fábrica por usuários que tenham certificados válidos, dentro do prazo definido na geração do *proxy* para o usuário. Os *nós internos* apenas encaminham as requisições e realizam o roteamento de mensagens, se necessário. O *Grid Portal* é o *host* que mantém a comunicação com os *hosts* de borda do domínio e realiza as funções de gerência e monitoramento de recursos dentro do domínio. Os recursos se comunicam por meio de serviços da fábrica. Esses dados trafegam em mensagens SOAP compactadas que seguem o estilo arquitetural *Representational State Transfer* (REST) [Fielding 2000].

A implementação dessa arquitetura utiliza o GT em *hosts* Linux Dell Quad Core, robôs móveis Pioneer P3-DX e sensores Intel Imote2 com Sistema Operacional TinyOS [TinyOS 2010]. O *SIG Agent* também suporta parcialmente a interação com o *softphone* Ekiga [Ekiga 2010] para estabelecer chamadas SIP (*Session Initiation Protocol*) no domínio do usuário. O *Grid Proxy* executa um servidor Apache Tomcat e um container Apache Axis2 para a comunicação com o *SIG Agent*.

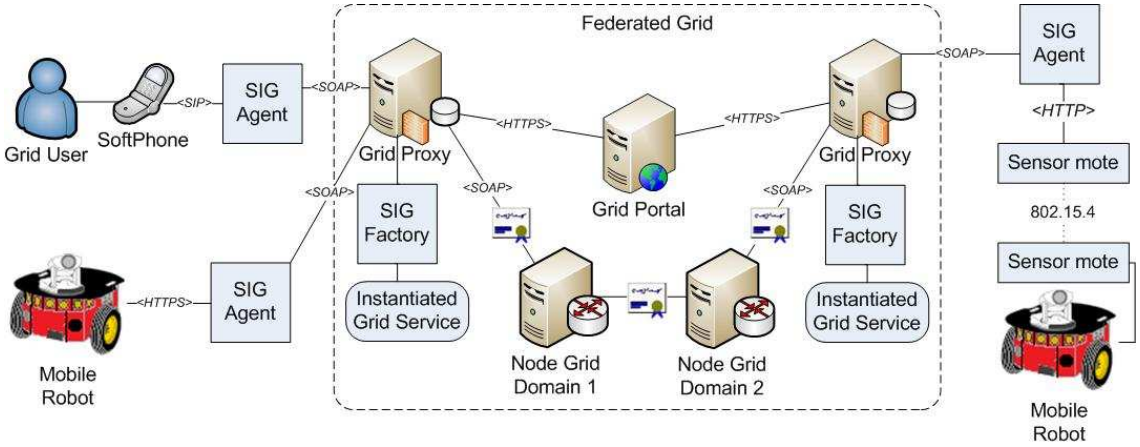


Fig. 2. Arquitetura para Colaboração em Grade.

4. Sistemas Classificadores em Grade

Como exemplo de experimento colaborativo, procurou-se utilizar a Grade como uma infraestrutura capaz de suportar a interação de múltiplos agentes autônomos controlados por Sistemas Classificadores, ativados com serviços da fábrica, que realizam chamadas de sistema às funções mapeadas do GT. Esse estudo considera o uso da Grade como infraestrutura de suporte para experimentos colaborativos.

Sistemas Classificadores foram originalmente propostos por Holland [Holland 1998]. Um Sistema Classificador de Aprendizagem (*Learning Classifier System* - LCS) é um sistema de propósito geral que adota uma metodologia evolucionária para sintetizar mecanismos de inferência adaptativos capazes de operar em condições variadas durante a sua execução [Geyer-Schulz 1995] [Eiben and Smith 2007]. Uma população de regras do tipo *condição-ação-predição*, chamadas classificadores, é mantida para representar a solução atual do problema. Cada classificador mantém uma parte do problema, com a *condição* relacionada às entradas, *ação* relacionada à decisão, e *predição* que estima a relevância do classificador em termos da solução.

O problema da navegação autônoma nos experimentos realizados envolve o alcance de um alvo pré-definido no espaço de busca discretizado. As informações sobre a localização do robô, do alvo e dos parâmetros para a ativação do robô são informados no *job*. No algoritmo descrito a seguir, o método *createRules* gera uma matriz R com classificadores binários aleatórios. Cada linha na matriz R é um classificador com uma estrutura de vetor e o campo *fitness* iniciando com o valor inteiro 100. O método *acquireInfoEnv* calcula a distância que o robô se encontra do alvo, traduz a entrada dos sensores em binário e insere o resultado no vetor C. O método *evaluate* calcula os campos de C que possuem o mesmo valor em cada linha da matriz de classificadores. Para cada combinação positiva, o campo de energia da regra é incrementado, com a conseqüência de gerar classificadores mais específicos para tratar a situação do ambiente. No método *competition* um processo de sinalização é realizado para selecionar os classificadores mais adequados às entradas do ambiente. O campo *status* no classificador recebe o resultado da competição. Inicialmente um processo de previsão é realizado para verificar se o próximo movimento irá orientar o robô móvel em direção

Algorithm 1 Classifier System

Require: population ≥ 1

1: $R \Leftarrow createRules(\#rules)$
2: $C \Leftarrow acquireInfoEnv(sensors)$
3: $e \leftarrow 1$
4: **while** $e \leq epoch$ **do**
5: $g \leftarrow 1$
6: **while** $g \leq generations$ **do**
7: $fit \Leftarrow evaluate(C, R)$
8: $R \Leftarrow competition(R)$
9: $R \Leftarrow action(R)$
10: $R \Leftarrow taxTimeLife(R)$
11: $g \leftarrow g + 1$
12: **end while**
13: $R2 \Leftarrow select(R)$
14: $R2 \Leftarrow reproduces(R2)$
15: $R2 \Leftarrow variate(R2)$
16: $R \Leftarrow renewRules(R2, R)$
17: $e \leftarrow e + 1$
18: **end while**

ao alvo. No experimento, o *fitness* é a relação entre a posição atual do robô e o objetivo a ser alcançado. O *fitness* é calculado como segue:

$$fit = ((1/dis\tan ce) + K) * noise \quad (1)$$

Na Eq.(1), como um problema de minimização, o *fitness* é o inverso da distância Euclidiana ao alvo; *K* é uma constante utilizada para balancear o processo de seleção do tipo *Roulette Wheel* [Eiben and Smith 2007] dos classificadores; *noise* é uma constante estimada fornecida para balancear os erros de deslocamento do robô no ambiente real. Portanto, o *fitness* representa a qualidade do classificador de acordo com as entradas recentes obtidas do ambiente. Um classificador com *fitness* elevado terá maiores chances na fase de competição em razão de estar relacionado a uma menor distância em relação ao alvo. Na fase de competição, cada classificador faz um lance (*Bid*) que depende da especificidade do classificador, energia atual e *fitness*, como descrito a seguir:

$$Bid_t = (spec_t / numSensors) + energy_t + fit_t \quad (2)$$

$$spec_t = (N - total\_\#) / N \quad (3)$$

$$Tax = Bid\_tax * Bid_t \quad (4)$$

onde *t* representa o instante em que os valores foram coletados. Nas Eq.(2) e Eq.(3), o *spec* é uma proporção entre o número total de bits menos os caracteres do tipo *'do not care'* (representados por '#' e que podem representar os bits 0 e 1), em relação ao número total de bits. Na Eq.(2) *numSensors* é o número de sensores do robô; a proporção entre *spec* e *numSensors* é interessante porque mantém o valor de *Bid* bem

distribuído para qualquer número de sensores; o valor *energy* representa a 'força' atual do classificador, ou seja, ao contrário do *fitness* que é válido apenas no movimento atual, o valor de *energy* é mantido nas gerações subsequentes, até que uma *época* seja completada. Na fase de competição uma taxa de participação decrementa o valor de *energy* dos classificadores. Essa atenuação é proporcional ao *Bid* oferecido, com a intenção de evitar valores elevados de *energy*, reduzindo as chances de outros classificadores ganharem nas próximas competições. Na Eq.(4) o valor de *Tax* é a taxa de participação; a contante *Bid tax* é aplicada no valor atual do *Bid* no instante *t*. Depois da fase de competição, a ação no classificador é realizada. De acordo com o resultado dessa ação, a regra é recompensada ou punida. A fórmula geral de cálculo de *energy* a cada *geração* é descrita como segue:

$$E_{t+1} = (1 - life\_tax) * E \ + R_t - Bid\_tax \quad (5)$$

Na Eq.(5), *E* é o valor de *energy* no instante *t*; *life tax* é a taxa de vida que é decrementada do valor de *energy* em cada *geração*; *R* é o valor da recompensa ou punição resultando da ação no instante *t-1*. De acordo com a ação no ambiente, será obtido $R \geq 0$ em uma ação positiva (aproximação ou alcance do alvo), ou $R \leq 0$ caso contrário. Finalmente, *Bid tax* é a constante aplicada ao valor atual de *Bid* no instante *t*. Note que o valor de *Bid* é utilizado para indicar o classificador vencedor na fase de competição. Em cada *geração*, o método *select* recupera o classificador com energia mais elevada, utilizando uma seleção do tipo *Roulette Wheel*, selecionando metade da população de classificadores e armazenando na matriz R2. O método *reproduces* realiza o *crossover* pontual nas regras; *variates* realiza o processo de mutação com inversão pontual de bits, com a intenção de aumentar a diversidade de regras. Finalmente, *renewRules* troca os classificadores com energia mais baixa pelos novos classificadores gerados. Portanto, a competição ocorre entre os descendentes, e um descendente que possui um valor elevado de *fitness* substitui a regra que obteve baixo valor de *energy* em R. Essa é uma estratégia evolucionária do tipo (μ, λ)-EE [Eiben and Smith 2007].

5. Avaliação e Resultados

O Sistema Classificador utiliza os serviços da Grade para realizar operações de telemetria com os sonares do robô (*getSonarReadings*), consulta de posição (*getPosition*), controle de orientação (*setHeading*) e deslocamento (*go*). O algoritmo controla o deslocamento do robô com o envio periódico de requisições, utilizando os serviços da Grade. Essa comunicação fim-a-fim é realizada como mostra o diagrama da Fig. 3. No cálculo do RTT (*Round Trip Time*) , obteve-se um RTT total médio de 950ms (milissegundos) nessa comunicação fim-a-fim.

A convergência do Sistema Classificador foi avaliada com o envio de requisições para o simulador MobileSim [MobileRobots 2010], com uma população de 5 robôs móveis. Na Fig. 4 as linhas espessas representam os caminhos percorridos: (1) o alvo definido em coordenadas cartesianas X=-12000, Y=-12000, escala em milímetros, foi definido para a população; (2) o algoritmo é iniciado e os robôs evitam colidir uns com os outros, mas se deslocam em direção ao alvo; (3) ocorre a convergência do algoritmo e os robôs terminam as suas atividades. A convergência do algoritmo mostra

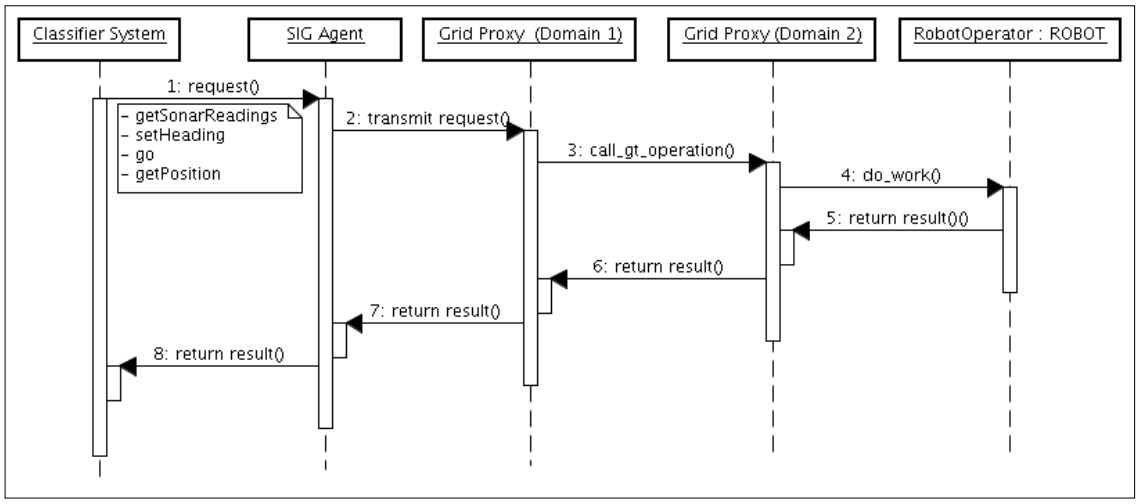
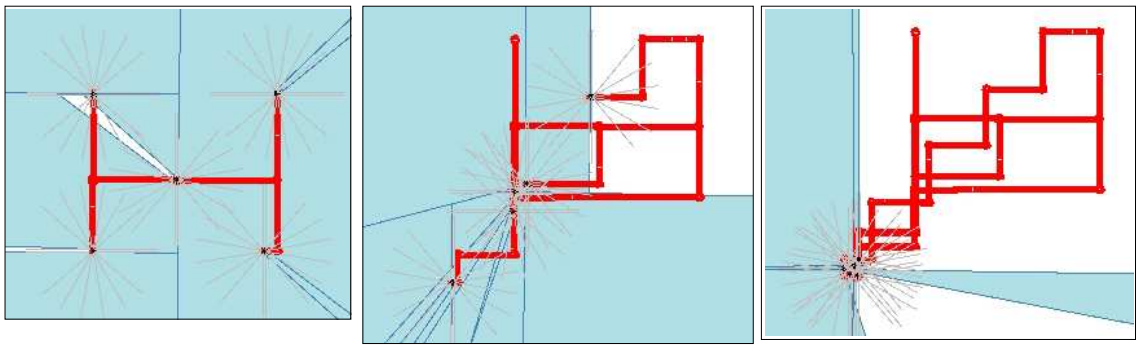


Fig. 3. Interação entre o Sistema Classificador, a Grade e o Robô.



(1) **(2)** **(3)**

Fig. 4. Convergência do Sistema Classificador.

que a infraestrutura é capaz de manter a interação de um conjunto de agentes autônomos, sem afetar o resultado da execução do algoritmo.

Outra classe importante de recursos da Grade são pequenos dispositivos sensores interligados em rede, como ilustrado na Fig. 2. Apresentamos a seguir um experimento envolvendo o acesso por meio da Grade a sensores Imote2. Um dos sensores tem a função de *gateway* interligando a Grade aos demais sensores da rede, um dos quais embarcado no robô. Nesse experimento foram realizados testes de RTT com um cliente diretamente conectado a um *Grid Proxy* e que consulta o recurso Mote que está conectado a outro *Grid Proxy*, de acordo com a Fig. 2. A Fig. 5 mostra o RTT total da arquitetura (RTT1): acesso ao *Grid Proxy*, transferência de dados na rede interna, acesso ao Mote, e retorno da resposta para o cliente. A comparação é feita com o RTT do acesso interno ao Mote (RTT2), a partir do *Grid Proxy* ao qual ele está conectado. Os resultados obtidos mostram que o desempenho da arquitetura é suficiente para o uso de aplicações que exigem reduzida latência para a execução de suas atividades. Essa característica é importante para diversos algoritmos, como os de computação evolutiva, que exigem a interação contínua com os serviços de correção de odometria para a navegação autônoma.

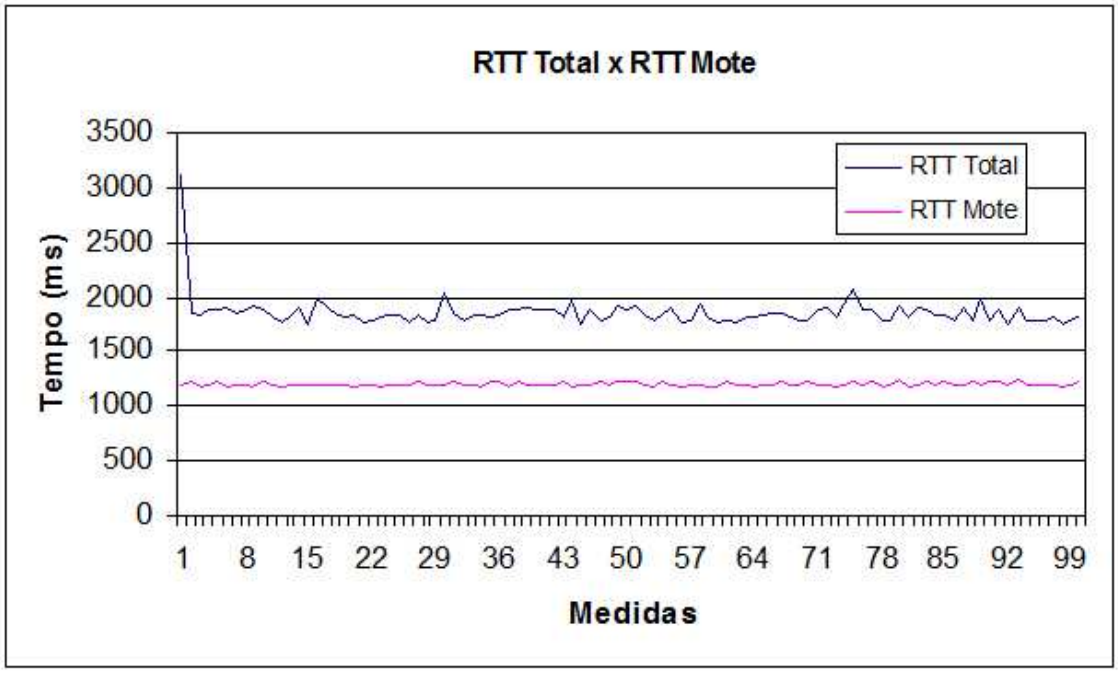


Fig. 5. Avaliação do Desempenho da Arquitetura.

A Tab. 1 mostra os resultados obtidos para uma população de 100 amostras para o RTT1 e RTT2. São apresentados também o desvio-padrão (DP) e o intervalo de confiança (IC) para um nível de significância de 0,05. Os resultados mostram que o desempenho da arquitetura permite a realização de experimentos remotos em um tempo aceitável e inferior, em média, a 1 segundo (diferença entre RTT1 e RTT2). Essas avaliações são interessantes porque permitem definir nos experimentos práticos o tempo de espera entre as requisições a recursos de baixo processamento, sem a perda acentuada de pacotes. No caso do recurso Mote, o tempo de espera entre as requisições do cliente deve ser superior à média do RTT1. Essa mesma métrica pode ser utilizada sem perda de generalidade para o acesso a outros recursos, sem considerar a sobrecarga do uso compartilhado da rede.

Média do RTT1 (ms)	DP	IC	Média do RTT2 (ms)	DP	IC
1858,14	143,309	28,088	1206,66	14,644	2,870

Tab. 1. Comparação entre RTT1 e RTT2.

6. Considerações Finais e Trabalhos Futuros

A avaliação inicial da arquitetura mostra que uma grande diversidade de experimentos pode ser realizada no ambiente. O suporte à agregação dinâmica de recursos simplifica o uso desse ambiente respeitando os quesitos de segurança. A arquitetura é uma alternativa para que experimentos colaborativos possam ser realizados com a abstração da lógica de configuração do domínio. Como trabalho futuros pretende-se estender o conjunto de experimentos e utilizar uma interface de interação colaborativa junto a um *workflow*, para submeter tarefas que exigem o processamento paralelo.

Referências

Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). *"OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing"*. Number ISBN 978-3-540-20405-3. Springer Berlin / Heidelberg.

Breitling, F., Granzer, T., and Enke, H. (2008). "*Grid Integration of Robotic Telescopes*". *Astronomische Nachrichten*.

Cazangi, R. R. (2008). *"Síntese de Controladores Autônomos em Robótica Móvel por meio de Computação Bio-inspirada"*. PhD thesis, Universidade Estadual de Campinas.

Cirne,W. and Neto, E. S. (2005). "*Grids Computacionais: Transformando a Computação através de Serviços sob Demanda*". *XXIII SBRC*.

Eiben, A. E. and Smith, J. E. (2007). *"Introduction to Evolutionary Computing"*. Springer, Natural Computing Series.

Ekiga (2010). Ekiga - free your speech. URL at `http://ekiga.org`.

Fielding, R. T. (2000). *"Architectural Styles and the Design of Network-based Software Architectures"*. PhD thesis, University of California.

Foster, I. and Iamnitchi, A. (2003). "*On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing*". Pages 118–128.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). "*The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*".

Foster, I. and Tuecke, S. (2005). "*The Different Faces of IT as Service*". Enterprise Distributed Computing - ACM Queue, 3(6).

Geyer-Schulz, A. (1995). "*Holland Classifier Systems*". SIGAPL APL Quote Quad Journal, 25(4):43–55. ISSN 0163-6006.

GGF (2010). Open Grid Forum. URL at `http://www.ggf.org`.

Globus (2010). The Globus Alliance. URL at `http://www.globus.org`.

GRID Infoware (2010). Grid Computing Info Centre (Grid Infoware). URL at `http://www.gridcomputing.com`.

Guillen, X. V. (2009). *"Distributed Resource Allocation for Contributory Systems"*. PhD thesis, Universitat Oberta de Catalunya.

Holland, J. H. (1998). "*Emergence: from chaos to order*". Addison Wesley, Inc.

IT Professional (2004). "*Grid Computing 101: What's All the Fuss About?*". IT Professional Magazine.

Krishnaswamy, R., Navarro, L., and Vlassov, V. (2008). "*A democratic grid: Collaboration, sharing and computing for everyone*". In eChallenges'08 Collaboration and the Knowledge Economy: Issues, Applications, Case Studies, Stockholm. IOS Press.

Laszewski, G. v., Hategan, M., and D., K. (2010). "*Java CoG Kit*". URL at `http://wiki.cogkit.org/wiki/Java CoG Kit`.

Luo, S., Peng, X., Fan, S., and Zhang, P. (2009). "*Study on Computing Grid Distributed Middleware and Its Application"*. International Forum on Informtion Technology and Applications.

MobileRobots (2010). Mobile Robots Inc. URL at `http://www.mobilerobots.com`.

OASIS (2010). Advancing Open Standards for the Information Society. URL at `http://www.oasis-open.org`.

OGF (2010). Open Grid Forum. URL at `http://www.ogf.org/UnderstandingGrids/resource_articles.php`.

Parker, P. M. (2010). "*Webster's Online Dictionary"*. URL at `http://www.websters-online-dictionary.org`.

Peterson, L., Anderson, T., Culler, D., and Roscoe, T. (2006). "*Experiences Building PlanetLab*". Proceedings of the Seventh Symposium on Operating System Design and Implementation (OSDI). URL at `http://www.planet-lab.org/about`.

Reis, L. P. (2003). "*Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico"*. PhD thesis, FEUP.

Sabatier, F., De Vivo, A., and Vialle, S. (2004). "*Grid Programming for Distributed Remote Robot Control*". Pages 358 – 363.

Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C., and Casanova, H. (2002). *"Overview of GridRPC: A Remote Procedure Call API for Grid Computing"*, volume 2536/2002. Springer Berlin / Heidelberg.

Siegwart, R. and Nourbacksh, I. R. (2004). *"Introduction to AutonomousMobile Robots"*. The MIT Press.

Sotomayor, B. (2005). *"The GT 4 Programmer's Tutorial"*. Apache Public License.

TinyOS (2010). Tinyos Community Forum. URL at `http://www.tinyos.net`.

Vilajosana, X., Daradoumis, T., Navarro, L., and Manuel, J. (2007). "*LaCOLLA: Middleware for Self-Sufficient Online Collaboration*". IEEE Internet Computing, 11:56–64.

Ziviani, A. and Duarte, O. C. M. B. (2005). "*Metrologia na Internet*". *XXIII SBRC*.