

SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes

Jomi Fred Hübner¹ * and Jaime Simão Sichman¹ **

Laboratório de Técnicas Inteligentes
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, 158
05508-900 São Paulo, SP
{jomi,jaime}@pcs.usp.br
<http://www.lti.pcs.usp.br>

Abstract Saci is a tool that turns programming communication among distributed agents easier. Two kinds of features are provided by Saci: an API for composing, sending, and receiving KQML messages; and tools to get the agents designer rid of some difficulties regarding a distributed environment (agent name service, directory service, remote launching, communication debug, etc.). This paper explains the Saci's fundamentals and use. It also shows that Saci has a good performance by comparing it with others similar tools.

Keywords Agent Communication, KQML, Tools for MAS development

1 Introdução

A implementação de alguns tipos de Sistema Multi-Agentes (SMA) requer que os agentes estejam distribuídos numa rede de computadores e que possam se comunicar neste ambiente [11]. Contudo, a implementação destes sistemas nem sempre é uma tarefa simples: o projetista, em geral preocupado em adicionar o atributo inteligente aos agentes, tem que se preocupar com problemas relacionados com a programação distribuída (portas do protocolo TCP/IP, RMI, CORBA, etc.).

Apesar da grande variedade de ferramentas disponíveis para auxiliar nesta tarefa, várias dificuldades são encontradas na sua utilização, entre elas podem-se citar:

1. baixa interoperabilidade: uma única linguagem de programação é suportada ou a ferramenta pode ser utilizada em um único sistema operacional, etc;
2. baixo desempenho: fato que inviabiliza sua utilização em sistemas onde há muita comunicação entre os agentes;
3. difícil utilização: a instalação, configuração e utilização são complexos e acabam exigindo do programador conhecimento de programação distribuída, protocolos de rede, etc.;

* Financiada pela Universidade Regional de Blumenau e CAPES

** Parcialmente financiado pelo CNPq, bolsa 301041/95-4

4. falta de robustez: o ambiente não funciona adequadamente por muito tempo.

Estas dificuldades motivaram o desenvolvimento do Saci (Simple Agent Communication Infrastructure). O Saci é uma ferramenta que torna transparente para o programador de agentes os aspectos de comunicação distribuída, tendo como principais objetivos ser simples de utilizar e ter um bom desempenho.

Na sequência deste artigo, apresenta-se a arquitetura do ambiente de desenvolvimento Saci, a programação de agentes usando o Saci e o resultado da comparação do Saci com outras ferramentas que têm os propósitos semelhantes.

2 Comunicação entre Agentes

Em um SMA os agentes não são desenvolvidos visando resolver um problema específico; pelo contrário, dado um problema, um grupo de agentes se une para resolvê-lo [1]. Assim, os agentes não precisam necessariamente terem sido desenvolvidos utilizando as mesmas ferramentas e modelos. Este grupo de agentes, contudo, deve concordar no uso de uma linguagem comum. Além disso, como nas sociedades humanas, a comunicação é mais eficiente quando seus usuários seguem certas regras, chamadas de protocolos de comunicação. Algumas especificações de Linguagem de Comunicação de Agentes (LCA) foram propostas com o objetivo de viabilizar a comunicação entre agentes desenvolvidos em projetos diferentes e possibilitar a descrição das regras que regem o fluxo de comunicação entre eles. Dentre as propostas de LCA existentes, duas se destacam: KQML e FIPA ACL.

2.1 Knowledge Query and Manipulation Language (KQML)

KQML é uma especificação de linguagem e protocolo de comunicação entre agentes [7, 8] que tem por objetivo ser um meio comum de troca de informação e conhecimento entre agentes¹. Esta especificação foi amplamente adotada por ter algumas características importantes numa LCA:

- Qualquer linguagem pode ser usada para escrever o conteúdo das mensagens (por exemplo: LISP, Prolog, SQL, Português);
- As informações necessárias para compreender o conteúdo da mensagens estão incluídas na própria comunicação;
- Quando os agentes trocam mensagens KQML, o mecanismo de transporte é transparente, i.e., como a mensagem sai do agente emissor e chega no receptor.
- O formato das mensagens é simples, fácil de ler por pessoas e de ser analisada por um *parser*.

Geralmente, arquiteturas que utilizam KQML adicionam um agente facilitador à sociedade. Este agente sabe quem são os agentes da sociedade e que requisições cada um é capaz de responder. Desta forma o procedimento de apresentação entre os agentes é simplificado.

¹ A troca de conhecimento e não dados é um dos fatores que diferencia uma LCA de uma linguagem padrão para comunicação entre objetos, como CORBA, por exemplo.

2.2 Foundation for Intelligent Physical Agents (FIPA) ACL

A especificação de LCA da FIPA está baseada nos mesmos princípios que o KQML (teoria dos atos de fala) e tem a mesma sintaxe para as mensagens [5]. Esta especificação consiste de um conjunto de tipos de mensagens e a descrição formal de sua pragmática — o efeito das mensagens na mente do emissor e do receptor. Neste aspecto, a FIPA ACL difere do KQML, a semântica dos tipos de mensagens não é exatamente a mesma. Além da LCA, a FIPA também especifica uma notação para descrição de protocolos de comunicação. Esta notação é exemplificada com protocolos para requisição de serviços, rede contratual e outros.

3 Saci

O Saci foi desenvolvido com base na especificação KQML, possuindo as seguintes características principais:

- Os agentes utilizam KQML para se comunicar. Há funções para compor, enviar e receber mensagens KQML.
- Os agentes são identificados por um nome. As mensagens são transportadas utilizando-se somente o nome do receptor, sua localização na rede é transparente.
- Um agente pode conhecer os outros por meio de um serviço de páginas amarelas. Agentes podem registrar seus serviços no facilitador e perguntá-lo sobre que serviços são oferecidos por quais agentes.
- Os agentes podem ser implementados como applets e terem sua interface em uma *home-page*.
- Os agentes podem ser iniciados remotamente.
- Os agentes podem ser monitorados. Os eventos sociais (entrada na sociedade, saída, recebimento ou envio de mensagens) podem ser visualizados e armazenados para análise futura.²

3.1 Especificação

Há consenso entre os pesquisadores de SMA no fato de que os agentes existem num ambiente e interagem com ele [4, 6, 11]. O Saci provê a parte *social* deste ambiente: um meio dos agentes *conhecerem* os demais e se *comunicarem*. Uma forma de um agente conhecer outro é por meio de uma consulta a uma relação com os nomes dos agentes da sociedade. Entretanto, em algumas aplicações um agente necessita conhecer mais do que simplesmente a identificação dos agentes, é necessário conhecer as habilidades dos demais agentes, o que eles podem fazer.

² Estas últimas propriedades do Saci não serão abordadas neste artigo, mas um manual detalhado do Saci está disponível em <http://www.lti.pcs.usp.br/saci>. Onde também podem ser obtidas informações sobre a implementação do Saci.

As informações necessárias para o conhecimento mútuo entre os agentes formam a estrutura da sociedade. Formalmente, um estado da estrutura da sociedade Saci é definida pela tupla:

$$Soc = \langle \mathcal{A}, \mathcal{S}, l, \delta \rangle$$

tal que

$$\begin{aligned} \mathcal{A} &= \{ \alpha \mid \alpha \text{ é a identificação de um agente que pertence à sociedade} \}, \\ \mathcal{S} &= \{ \sigma \mid \sigma \text{ é uma habilidade disponível na sociedade} \}, \\ l &\text{ é a linguagem da sociedade, e} \\ \delta &: \mathcal{A} \rightarrow \mathbb{P}(\mathcal{S}) \text{ é uma função parcial que mapeia as habilidades de um} \\ &\text{agente, tal que} \\ \delta(\alpha) &= \{ \sigma \mid \sigma \text{ é uma habilidade de } \alpha \}. \end{aligned}$$

Por exemplo:

$$\begin{aligned} lti &= \langle \{ \text{Jomi, Jaime, Julio, José} \}, \\ &\quad \{ \text{Java, C, Prolog, Teach} \}, \\ &\quad \text{Português}, \\ &\quad \{ \text{Jomi} \mapsto \{ \text{Java, Prolog} \}, \text{Jaime} \mapsto \{ \text{C, Teach} \}, \text{Julio} \mapsto \{ \text{Java} \} \} \rangle \end{aligned}$$

A estrutura da sociedade pode mudar no decorrer do tempo por causa de eventos sociais:

$$Soc_i \Rightarrow Soc_{i+1} \mid \text{algum evento social aconteceu no momento } i.$$

Os eventos sociais que podem acontecer refletem o ciclo de vida de um agente:

Entrada na sociedade: o agente ganha um lugar na sociedade adicionando sua identificação na sociedade. A entrada de um agente α altera a estrutura da sociedade da seguinte forma:

$$\langle \mathcal{A}, \mathcal{S}, l, \delta \rangle_i \Rightarrow \langle \mathcal{A}', \mathcal{S}, l, \delta \rangle_{i+1} \mid \mathcal{A}' = \mathcal{A} \cup \{ \alpha \}$$

Anúncio de habilidades: opcionalmente, um agente pode anunciar suas habilidades para a sociedade. Se um agente α anuncia uma habilidade σ , a estrutura da sociedade muda da seguinte forma:

$$\begin{aligned} \langle \mathcal{A}, \mathcal{S}, l, \delta \rangle_i &\Rightarrow \langle \mathcal{A}, \mathcal{S}', l, \delta' \rangle_{i+1} \mid \mathcal{S}' = \mathcal{S} \cup \{ \sigma \} \\ \delta'(x) &= \begin{cases} \delta(x) & \text{if } x \neq \alpha \\ \delta(\alpha) \cup \{ \sigma \} & \text{if } x = \alpha \end{cases} \end{aligned}$$

Envio e recebimento de mensagens para outros agentes na mesma sociedade.

Saída da sociedade: o agente α que sai da sociedade perde sua identidade nesta sociedade:

$$\begin{aligned} \langle \mathcal{A}, \mathcal{S}, l, \delta \rangle_i &\Rightarrow \langle \mathcal{A}', \mathcal{S}', l, \delta' \rangle_{i+1} \mid \mathcal{A}' = \mathcal{A} - \{ \alpha \} \\ \mathcal{S}' &= \{ \sigma \mid \sigma \in \delta'(x) \} \\ \delta'(x) &= \begin{cases} \delta(x) & \text{if } x \in \mathcal{A}' \\ \{ \} & \text{otherwise} \end{cases} \end{aligned}$$

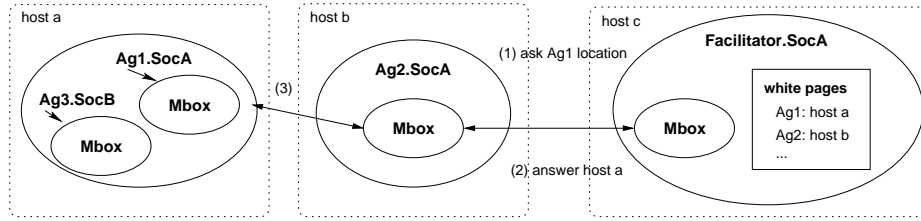


Figura 1. Serviço de páginas brancas. O agente *Ag2* da sociedade *SocA* deseja comunicar-se com o agente *Ag1* que pertence a mesma sociedade e que o *Ag2* conhece o nome. Inicialmente, o *MBox* do *Ag2* precisa saber a localização do *Ag1*, portanto, pergunta ao facilitador da sociedade *SocA* tal localização (seta (1)), que lhe responde *host a* (seta (2)). Tendo a localização, o *Ag2* inicia a comunicação com o agente *Ag1* (seta (3)). O agente que está no *host a* pertence a duas sociedades *SocA* e *SocB*, tendo um *Mbox* para cada sociedade.

3.2 Arquitetura

Entrada e saída de sociedades. Como sugerido pela arquitetura KQML, cada sociedade tem um agente facilitador que mantém sua estrutura: a identidade, localização³ e serviços oferecidos pelos agentes da sociedade. Quando um agente deseja entrar numa sociedade ele tem que contatar o facilitador e registrar um nome. O facilitador irá verificar a unicidade deste nome e associá-lo à localização do agente. Da mesma forma, quando deseja sair da sociedade, deve avisar o facilitador desta sociedade.

Envio e recebimento de mensagens. Baseado na Application Program Interface (API) proposta pela ferramenta MASENV [2], o Saci possui um componente denominado *MBox* que serve de interface entre o agente e a sociedade. Sua finalidade é tornar transparente o envio e o recebimento de mensagens. Este componente possui funções que encapsulam a composição de mensagens KQML, o envio síncrono e assíncrono de mensagens, o recebimento de mensagens, o anúncio e a consulta de habilidades e o *broadcast* de mensagens (cf. Fig. 1).

Anúncio de habilidades. Para ser melhor conhecidos na sociedade, os agentes podem anunciar suas habilidades ao facilitador de forma análoga ao serviço de páginas amarelas. Assim, quando um agente precisa de um serviço e não conhece o nome de um agente capaz de realizá-lo, pode requisitar ao facilitador uma lista de agentes com tal habilidade (a Fig. 2 ilustra um exemplo dessa interação). Na verdade, este é apenas um modelo disponível no Saci para a apresentação, sendo que outras formas podem ser utilizadas, como o *broadcasting* das habilidades.

³ Na arquitetura do Saci, alguns elementos foram adicionados ao modelo apresentado na Sec. 3.1, por exemplo, a localização do agente na rede.

3.3 Utilização

Para demonstrar algumas das funções oferecidas pelo Saci para a programação da comunicação entre os agentes, segue um programa que implementa um agente que tem a habilidade de somar e disponibiliza esta funcionalidade à sua sociedade:

```
1 import saci.*;
2 public class PlusServer extends Agent {
3     public static void main(String [] args) {
4         Agent a = new PlusServer ();
5         Config c = new Config ();
6         c.set(" facilitator .host", " nantes");
7         if (a.enterSoc(" Ag2", c)) {
8             a.initAg ();
9             a.run ();
10        } }
11    public void initAg () {
12        try {
13            mbox.advertise(" ask-one", null, "math", "X+Y");
14        } } catch (Exception e) { ... }
15    public void run () {
16        while (true) {
17            Message m = mbox.polling ();
18            if (m.get(" performative").equals(" ask-one")) {
19                Message r = new Message(" tell");
20                r.put(" receiver", m.get(" sender"));
21                r.put(" in-reply-to", m.get(" reply-with"));
22                r.put(" content", sum(m.get(" content")));
23                mbox.sendMsg(r);
24            } } }
25    String sum(Object formula) { ... }
26 }
```

O ciclo de vida do agente está codificado nas linhas 7 a 9: o agente entra na sociedade sugerindo uma identificação *Ag2*, anuncia suas habilidades (através do método *initAg*) e inicia sua execução (recebimento e envio de mensagens no método *run*). O método *initAg*, na linha 13, envia um *advertise* ao facilitador indicando que o agente pode receber mensagens com a performativa *ask-one*, com conteúdo escrito em qualquer linguagem — o *null* indica isto —, na ontologia *math* e com formato *X+Y*. Note que é através do objeto *mbox* que as funcionalidades do Saci são utilizadas, por exemplo: na linha 13 é feito o anúncio nas páginas amarelas do facilitador, na linha 17 o agente espera até que uma mensagem chegue em seu mail box, nas linhas 19 a 22 uma mensagem KQML é construída, e na linha 23 esta mensagem é enviada assincronamente.

Um agente que solicita o serviço do agente desenvolvido acima poderia ter o seguinte método *run*:

```
1 public void run(String exp) {
2     try {
3         Vector receptores =
```

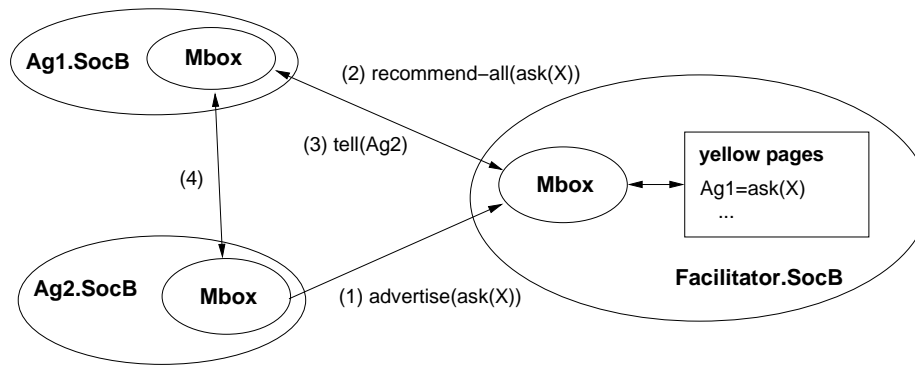


Figura 2. Serviço de páginas amarelas. Tendo o agente *Ag2* anunciado a habilidade x (seta (1)), se o agente *Ag1* pedir ao facilitador uma lista de agentes com a habilidade x (seta (2)), receberá *Ag2* como resposta (seta (3)) e poderá iniciar a comunicação com este agente (seta (4)).

```

4      mbox.consultYP("ask-one", null, "math", "X+Y");
5      if (receptores.size() == 0) {
6          System.out.println("No agent to answer me!");
7      } else {
8          for (int i=0; i < receptores.size(); i++) {
9              String ag = (String) receptores.elementAt(i);
10             Message m = new Message("ask-one");
11             m.put("content", "\"" + exp + "\"");
12             m.put("receiver", ag);
13             Message r = mbox.ask(m);
14             System.out.println(r.get("sender")+
15                 " answers is " + r.get("content"));
16         } }
17     } catch (Exception e) { ... }
18 }

```

O método `consultYP` (linha 4) encapsula a consulta ao facilitador pelos agentes que tem a capacidade de somar. Para cada um destes agentes é enviada uma mensagem (composta nas linhas 10 a 12) solicitando a realização de uma soma (linha 13). O método utilizado para esta solicitação (`ask(m)`) envia uma mensagem de performativa `ask` e espera a resposta para esta mensagem.

4 Comparação com Outras Ferramentas

A fim de avaliar o Saci, foi realizada uma comparação com outras ferramentas que apresentam características semelhantes, a saber: ter distribuição gratuita, a comunicação entre os agentes é feita segundo uma das especificações vistas na Sec. 2, ter serviços de páginas brancas e amarelas e utilizar Java como linguagem

de programação. As seguintes ferramentas foram selecionadas: JKQML versão 5.1a, desenvolvida sob o projeto alphaWorks da IBM [10]; Jackal versão 3.1, desenvolvido na UMBC [3]; e FIPA-OS versão 1.1, desenvolvido pela Nortel Networks [9]. As duas primeiras seguem a especificação KQML e a última a FIPA ACL.

Os critérios adotados para a comparação estão agrupados em dois grupos distintos: características e desempenho das ferramentas. Quanto às características, os critérios são os seguintes:

- c1. Os agentes poderem ser executados como applets em home-pages.
- c2. Possuir um mecanismo de monitoramento que mostre as mensagens trocadas entre os agentes e a estrutura da sociedade.
- c3. Simplicidade de utilização. Neste item foram utilizadas duas métricas: (a) o número de palavras do código fonte dos agentes desenvolvidos para o teste; e (b) o número de parâmetros de configuração que tiveram que ser alterados para a realização dos testes.
- c4. Possibilidade de definir os protocolos de comunicação a serem seguidos pelos agentes.

Quando ao desempenho, foi medido o número médio de mensagens trocadas por segundo entre dois agentes. Estes dois agentes funcionam de forma muito semelhante aos descritos na Sec. 3.3, i.e., um agente tem papel de servidor: anuncia seu serviço, espera requisições (performativas do tipo ask) e as responde. O outro agente tem papel de cliente: encontra um servidor através das páginas amarelas, e mede o tempo necessário para realizar n requisições. Os testes foram realizados em três circunstâncias distintas:

- t1. Os dois agentes executando na mesma Máquina Virtual Java (JVM) como threads.
- t2. Os dois agentes executando em JVMs diferentes mas no mesmo computador.
- t3. Os dois agentes executando em computadores diferentes.

4.1 Resultados

A Tab. 1 apresenta o resultado da comparação considerando as características de cada ferramenta. O JKQML não possui exatamente a noção de protocolos, mas permite que a comunicação entre os agentes seja regulada por meio de políticas de conversação entre os agentes. Entretanto, o manual não diz como utilizar esta capacidade. Quanto ao critério 3, evidentemente não se pode avaliar a simplicidade de utilização de uma ferramenta somente pelo tamanho do código

| Ferramenta | Critérios | | | | |
|------------|-----------|-----|-------|-------|-----|
| | c1 | c2 | c3(a) | c3(b) | c4 |
| Saci | sim | sim | 421 | 0 | não |
| Jackal | não | não | 382 | 6 | sim |
| JKQML | não | não | 1176 | 1 | sim |
| FIPA-OS | não | sim | 950 | 1 | não |

Tabela 1. Resultado da comparação entre as ferramentas quanto às características apresentadas

fonte e o número de parâmetros de configuração. Entretanto, os números levantados corroboram a avaliação subjetiva dos autores, i.e., as ferramentas apresentam a seguinte ordem de simplicidade de uso (da mais simples para a mais complexa): Saci, FIPA-OS, JKQML, Jackal.

Os testes de desempenho foram realizados no seguinte ambiente: o agente com papel de servidor foi executado em um Pentium 333 Mhz, 64 Mb memória e Linux 2.2; no caso do teste t3, o agente cliente foi executado em um Pentium 233 Mhz, 128 Mb de memória e Linux 2.2.5; a rede local é de 10 Mbps; em ambas as máquinas foi utilizado Java Development Kit (JDK) 1.2.2 com compilador Just In-Time (JIT) habilitado; os programas foram compilados com otimização de código.

A Tab. 2 apresenta o número médio de mensagens trocadas por segundo na execução de 2000 requisições pelo agente cliente. No primeiro teste, os dois agentes na mesma JVM, o Saci teve um desempenho muito melhor devido a uma otimização em seu mecanismo de transferência: quando tanto o emissor quanto o receptor estão na mesma JVM o envio da mensagem é feito por chamada de método, não passando pelo mecanismo de transferência da camada de rede. Nos demais testes, supõe-se que as outras ferramentas tiveram um desempenho inferior porque implementam restrições de protocolos.

Uma característica importante observada nas outras ferramentas foi a degradação do desempenho com o passar do tempo. Por exemplo, até a requisição 50 o Jackal enviava 8,44 mensagens/seg., esse número vai diminuindo até que chega em 4,73 mensagens/seg. No caso do JKQML este problema é ainda maior (cf. Fig. 3). Isso pode estar ocorrendo pelo fato da ferramenta manter informações das conversas que estão acontecendo. Entretanto, os manuais não comentam como liberar a memória utilizada para manter esta informação.

| Ferramentas | Testes | | |
|-------------|----------|--------|--------|
| | t1 | t2 | t3 |
| Saci | 2.412,54 | 197,24 | 137,49 |
| Jackal | 6,03 | 6,64 | 4,73 |
| JKQML | 1,46 | 2,55 | 3,56 |
| FIPA-OS | 17,95 | 25,13 | 18,86 |

Tabela 2. Resultado da comparação entre as ferramentas quanto ao desempenho

5 Conclusão

O ambiente Saci mostrou-se uma solução viável para a implementação da comunicação e apresentação num SMA. Os problemas apresentados na introdução são minimizados: a portabilidade é garantida por meio da linguagem Java, os testes mostraram que o desempenho do Saci é muito bom, o Saci encapsula vários aspectos de programação distribuída (dispensando o usuário de realizar configurações), e o desempenho no decorrer do tempo não diminui, pelo contrário, aumenta. Comparando com outras ferramentas, o Saci apresenta um desempenho nitidamente melhor. Entretanto, para realizar uma comparação mais precisa e avaliar as razões dos resultados, é necessário implementar o conceito de conversação no Saci (o que será feito nas próximas versões).

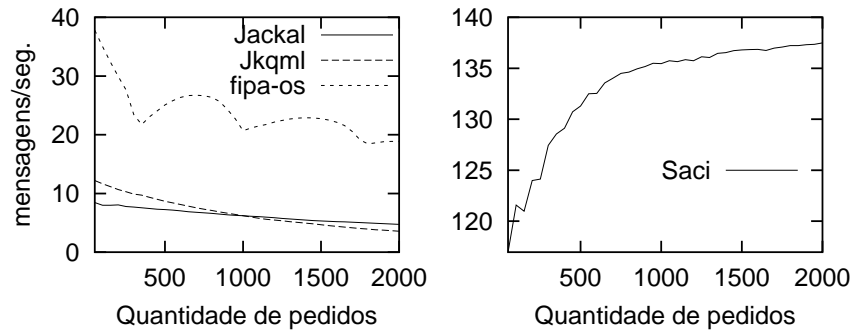


Figura 3. Desempenho de dois agentes em máquinas diferente

Referências

- [1] ALVARES, L. O., SICHMAN, J. S. Introdução aos sistemas multiagentes. In: MEDEIROS, C. M. B. (Ed.) *Jornada de Atualização em Informática*. Brasília: SBC, agosto 1997. v. 16, Cap. 1, p. 1ss.
- [2] CARDOZO, E., SICHMAN, J. S., DEMAZEAU, Y. Using the active object model to implement multi-agent systems. In: International Conference on Tools with Artificial Intelligence, 5., 1993, Boston, USA. *Proceedings...* c1993.
- [3] COST, R. S., FININ, T., LABROU, Y., LUAN, X., PENG, Y., SOBOROFF, I., MAYFIELD, J., BOUGHANNAM, A. Jackal: A java-based tool for agent development. In: Workshop on Tools for Developing Agents, 1998, Madison, Wisconsin. *Working notes...* c1998. (<http://jackal.cs.umbc.edu/Jackal/>).
- [4] DEMAZEAU, Y., MÜLLER, J.-P. (Eds.). *Decentralized artificial intelligence*. Amsterdam: Elsevier, 1990.
- [5] FIPA, Geneva. *FIPA 97 specification, part2: Agent communication language*, 1997. (<http://www.fipa.org>).
- [6] FRANKLIN, S., GRAESSER, A. Is it an agent or just a program? a taxonomy for autonomous agents. In: International Workshop on Agent Theories, Architectures, and Languages, 3., August 12-13, 1996, Budapest, Hungary. *Proceedings...* Editors MÜLLER, J. P., WOOLDRIDGE, M., JENNINGS, N. R. Lecture Notes in Computer Science, Vol. 1193. Springer, c1997. p. 21-35.
- [7] LABROU, Y., FININ, T. *A proposal for a new KQML specification*. UMBC, Baltimore, 1997.
- [8] LABROU, Y., FININ, T., PENG, Y. Agent communication languages: the current landscape. *IEEE Intelligent Systems*, v. 14, n. 2, p. 45-52, March/April 1999.
- [9] NETWORKS, N. FIPA-OS, 2000. (<http://www.nortelnetworks.com/products/announcements/fipa/>).
- [10] TSUCHITAMI, H. Java KQML. (<http://www.alphaworks.ibm.com/formula/JKQML>).
- [11] WEISS, G. (Ed.). *Multiagent systems: A modern approach to distributed artificial intelligence*. London: MIT Press, 1999.