Simple Agent Communication Infrastructure

(SACI)

Jomi Fred Hübner

FURB / DSC / GIA

&

Jaime Simão Sichman

USP / POLI / LTI

Agent's Day, Itajaí, 2003

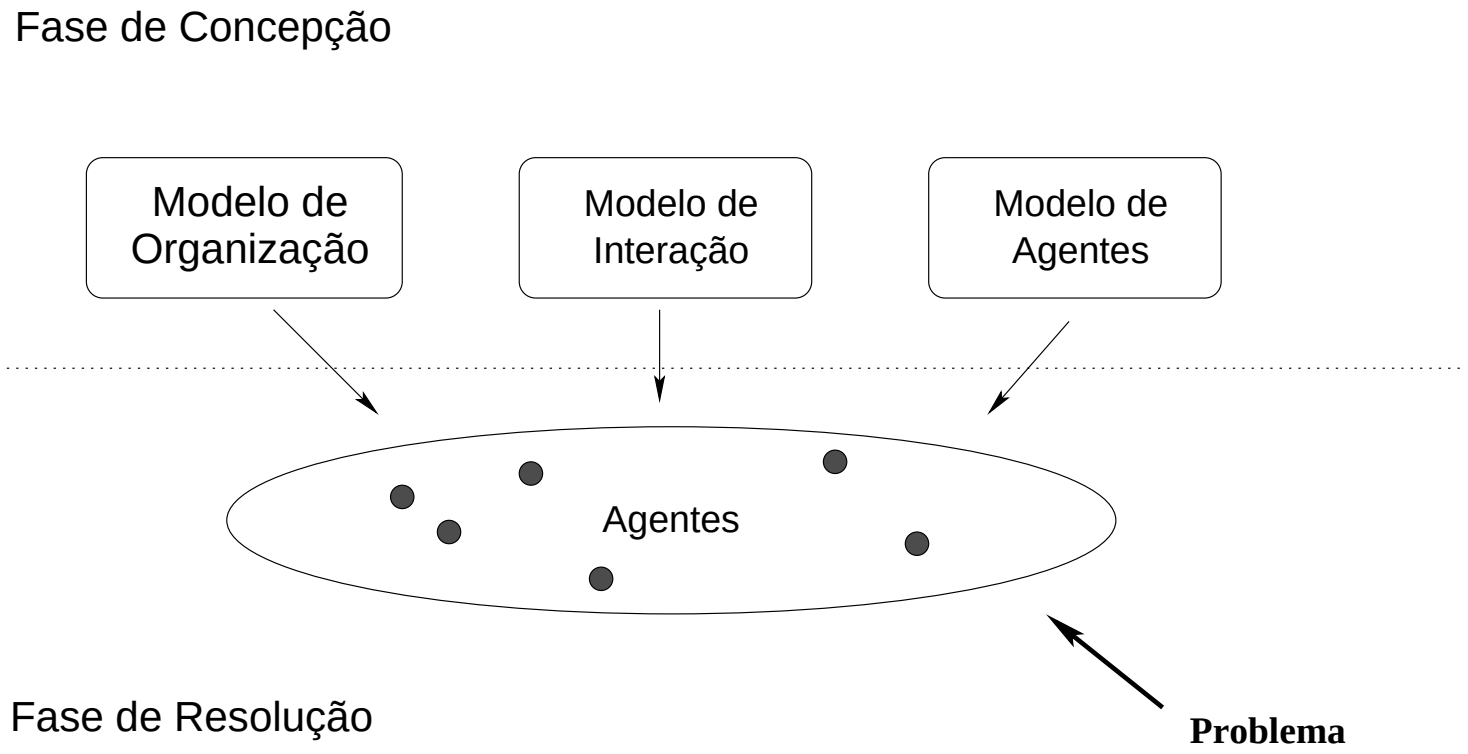
Conteúdo

- Introdução à Sistemas Multiagentes
- Fundamentos do SACI
- Comunicação entre agentes com o SACI
- Utilização dos serviços do SACI

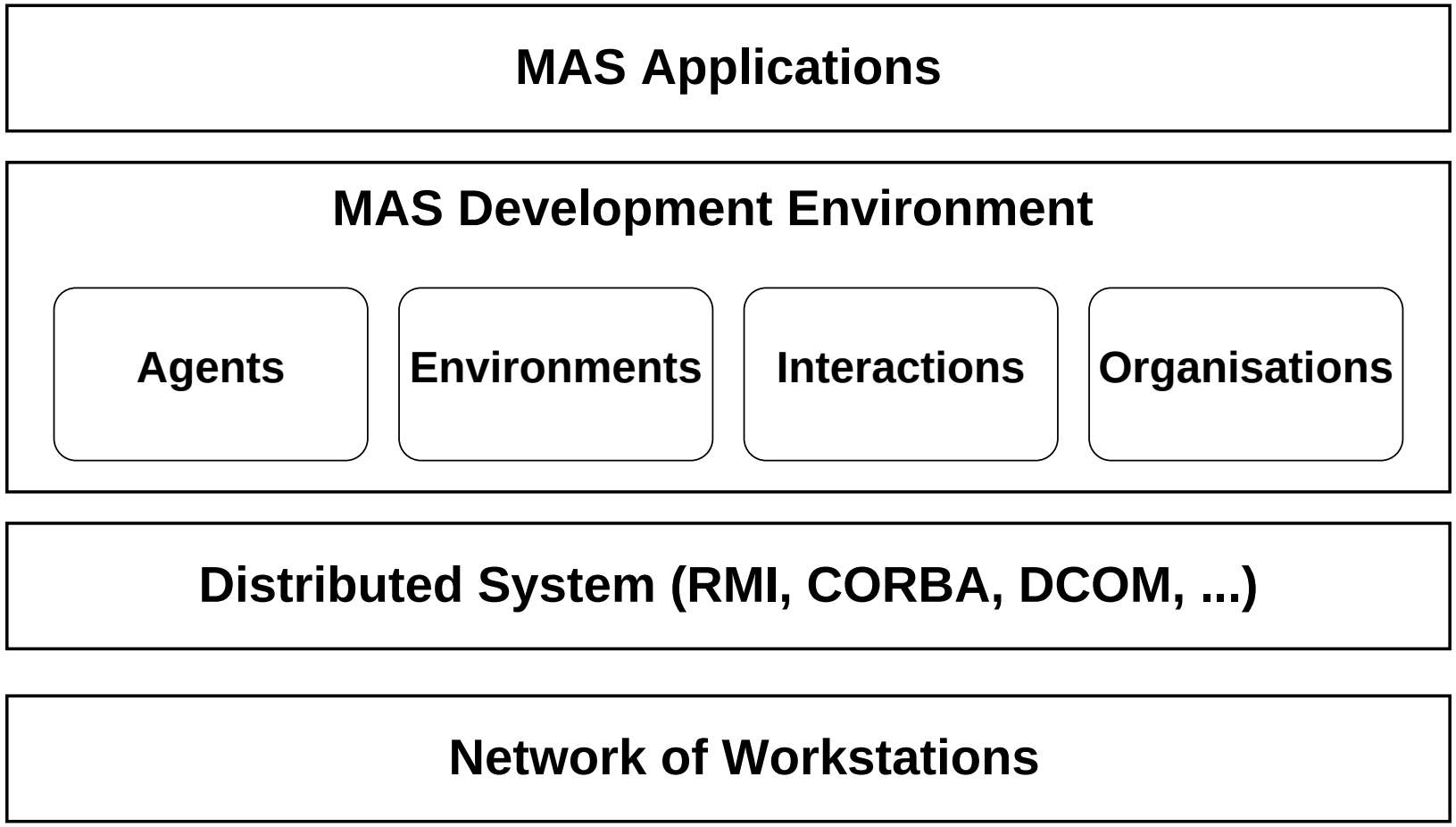
Introdução

Contexto

Sistemas Multiagentes: grupo de agentes que interagem com um objetivo [Weiß, 1999]



Desenvolvimento de SMA:



[Demazeau, 1995]

O que uma ferramenta para interação deveria ter:

- Envio e recebimento de mensagens utilizando uma linguagem de comunicação
- Uso de protocolos de comunicação especificados pelo usuário
- Acompanhamento do funcionamento do sistema
- Facilidades: páginas amarelas, execução remota, interface, ...

Motivação

Uma ferramenta com as seguintes características (além das enumeradas no slide anterior)

- interoperabilidade (linguagens, SO, ...)
- desempenho
- robustez
- fácil utilização para o programador e para o usuário (facilitando o ensino de SMA)
- transparência dos aspectos de distribuição
- open-source

Saci

O SACI é uma ferramenta que torna a programação da **comunicação** entre agentes distribuídos mais fácil, em conformidade com um padrão, rápida e robusta.

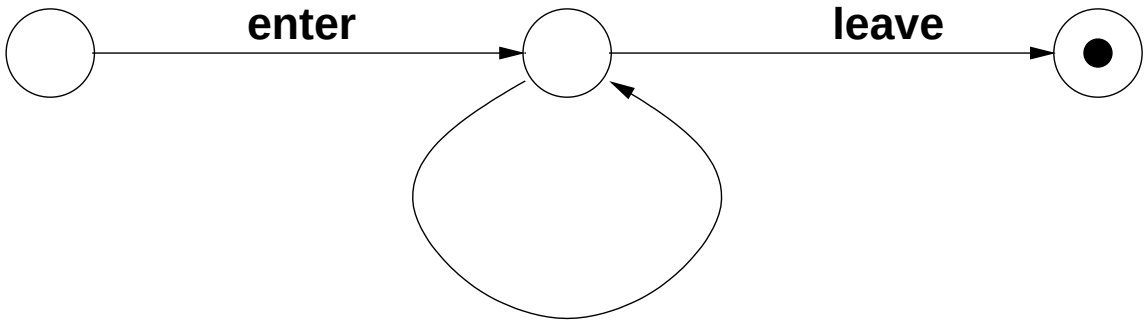
- Possui uma API para compor (parser), enviar (as/sincronamente) e receber (mail box) mensagens KQML
- Um conjunto de facilidades:
  - ⋆ páginas brancas (nomes e localização dos agentes)
  - ⋆ págimas amarelas (serviços disponibilizados pelos agentes)
  - ⋆ controle e execução remota de agentes
  - ⋆ monitoramento da sociedade (eventos sociais podem ser obtidos)
  - ⋆ mobilidade de agentes

Fundamentos

Agentes

Para o Saci, os **agentes** possuem as seguintes propriedades:

- estão agrupados em sociedades
- possuem uma identificação única
- interagem com os demais agentes utilizando uma linguagem comum
- oferecem serviços aos demais agentes da sua sociedade
- tem o seguinte ciclo de vida:



Sociedade

A estrutura de uma **sociedade** é especificada pela tupla

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

tal que

$\mathcal{A} = \{\alpha \mid \alpha$ é uma identificação de agente$\}$,
$\mathcal{S} = \{\sigma \mid \sigma$ é uma habilidade disponível$\}$,
l é a linguagem de comunicação da sociedade
$\delta : \mathcal{A} \nrightarrow \mathbb{P}(\mathcal{S})$ mapeamento das habilidades dos agenes, tal que
$\delta(\alpha) = \{\sigma \mid \sigma$ é uma habilidade de $\alpha\}$.

por exemplo:

$$\begin{aligned} \text{Iti} = \langle & \{\text{Jomi, Jaime, Julio, Jose}\}, \\ & \{\text{Java, C, Prolog, Teach}\}, \text{Portuguese}, \\ & \{\text{Jomi} \mapsto \{\text{Java}\}, \text{Jaime} \mapsto \{\text{C,Teach}\}, \text{Julio} \mapsto \{\text{Java}\}\}\rangle \end{aligned}$$

Eventos Sociais

- Um agente **entra** na sociedade

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

- Um agente **anuncia** uma habilidade

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

$$\delta'(x) = \begin{cases} \delta(x) & \text{if } x \neq \alpha \\ \delta(x) \cup \{\sigma\} & \text{otherwise} \end{cases}$$

- Os agentes **enviam**/**recebem** mensagens

- Um agente **deixa** a sociedade

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

$$\delta'(x) = \begin{cases} \delta(x) & \text{if } x \in \mathcal{A}' \\ \{\} & \text{otherwise} \end{cases}$$

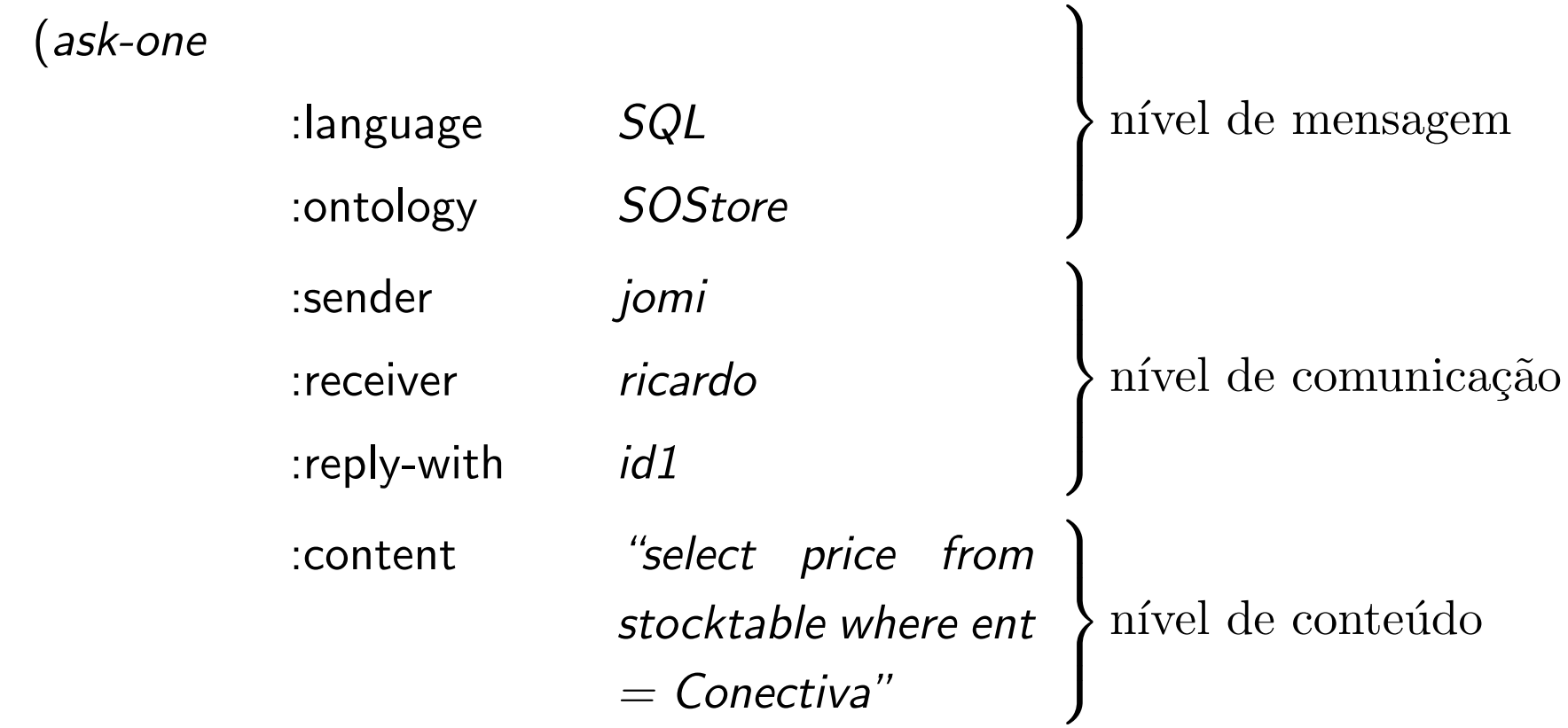
$$\mathcal{S}' = \{\sigma \mid \sigma \in \delta'(x)\}$$

KQML (Knowledge Query and Manipulation Language)

KQML é uma especificação de linguagem de comunicação entre agentes [Labrou and Finin, 1997].

- Qualquer linguagem pode ser usada para escrever o conteúdo da mensagem
- A informação necessária para a interpretação da mensagem está na própria mensagem
- Os agentes podem ignorar o mecanismos de transporte (TCP/IP, RMI, IIOP, ...)
- O formato é simples, fácil de ler a verificar

Exemplo de mensagem KQML:



```
(tell
      :language       prolog
      :ontology       SOStore
      :receiver       jomi
      :sender         ricardo
      :in-reply-to    id1
      :reply-with     id45
      :content        "[price(10.0)]")
```

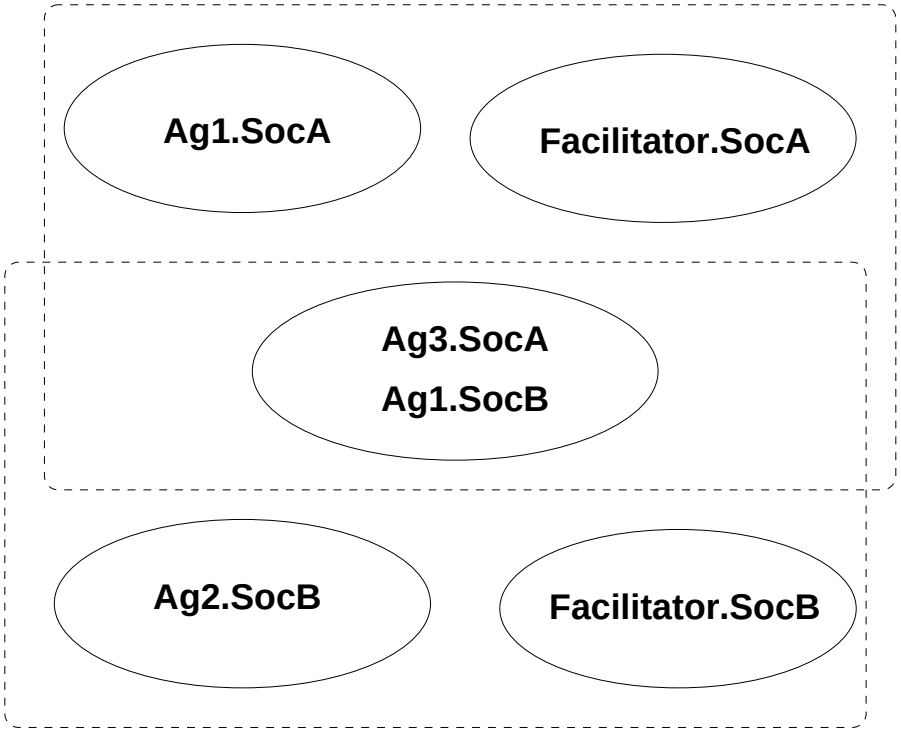
Embora a KQML tenha um conjunto pré-definido de performativas e palavras-chave, este conjunto não é nem mínimo nem fechado. Contudo, os agentes que usam uma das performativas reservadas devem usá-las da forma padrão.

Arquitetura de funcionamento do SACI

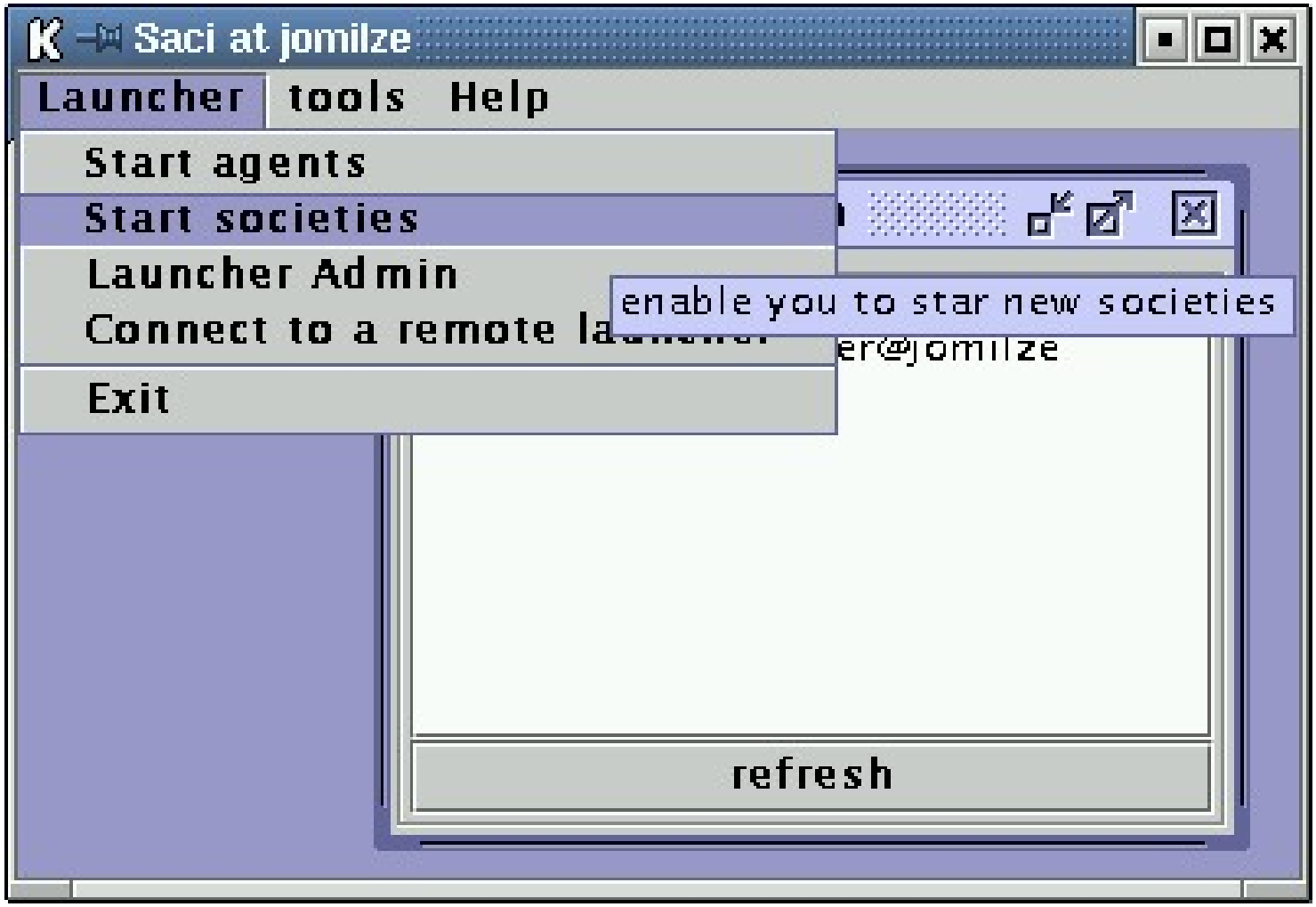
Entrada e **saída** da sociedade

Cada sociedade possui um **agente facilitador** que mantém sua estrutura no decorrer da sua história (sequência de eventos sociais).

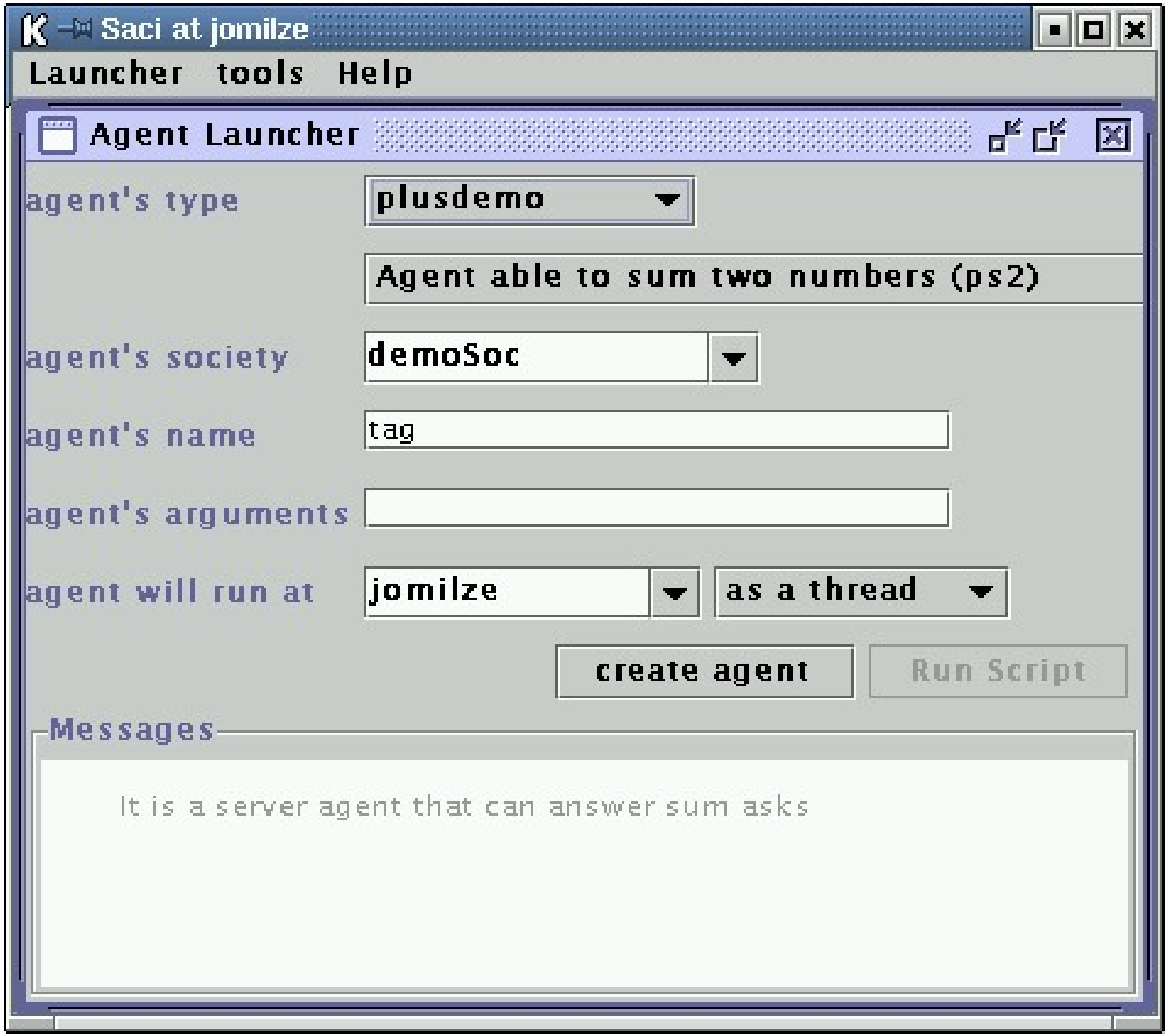
- Para entrar em uma sociedade um agente tem que registrar seu nome no facilitador e, antes de sair, deve avisar o facilitador.
- Os serviços que um agente deseja disponibilizar à sociedade devem ser anunciados ao facilitador.



Ferramentas para criação de agentes e facilitadores




Society Launcher
society's name
simp
The host where society's facilitator will run
jomilze.lar
Organizational model
Flat
Moise+
no parameters
create society
Messages
Creating flat society simp@jomilze.lar...ok


Saci at jomilze
Launcher tools Help
Agent Launcher
agent's type
plusdemo
Agent able to sum two numbers (ps2)
agent's society
demoSoc
agent's name
tag
agent's arguments
agent will run at
jomilze
as a thread
create agent
Run Script
Messages
It is a server agent that can answer sum asks

Exercício de **criação** de agentes

- Instalar e executar o SACI

  http://www.lti.pcs.usp.br/saci

- Criar uma sociedade chamada "curso" (menu launcher/start societies)
- Criar outra sociedade chamada "teste"
- Criar dois agente, com nomes "soma2a" e "soma2b", na sociedade "curso" (menu launcher/start agents; agent type=plusDemo/agent able to sum...)
- Criar dois agente, com nomes "soma2a" e "teste2b", na sociedade "teste"

- Criar um agente, com nome “consulta”, na sociedade “curso” (menu launcher/start agents; agent type=plusDemo/interface agent that...)

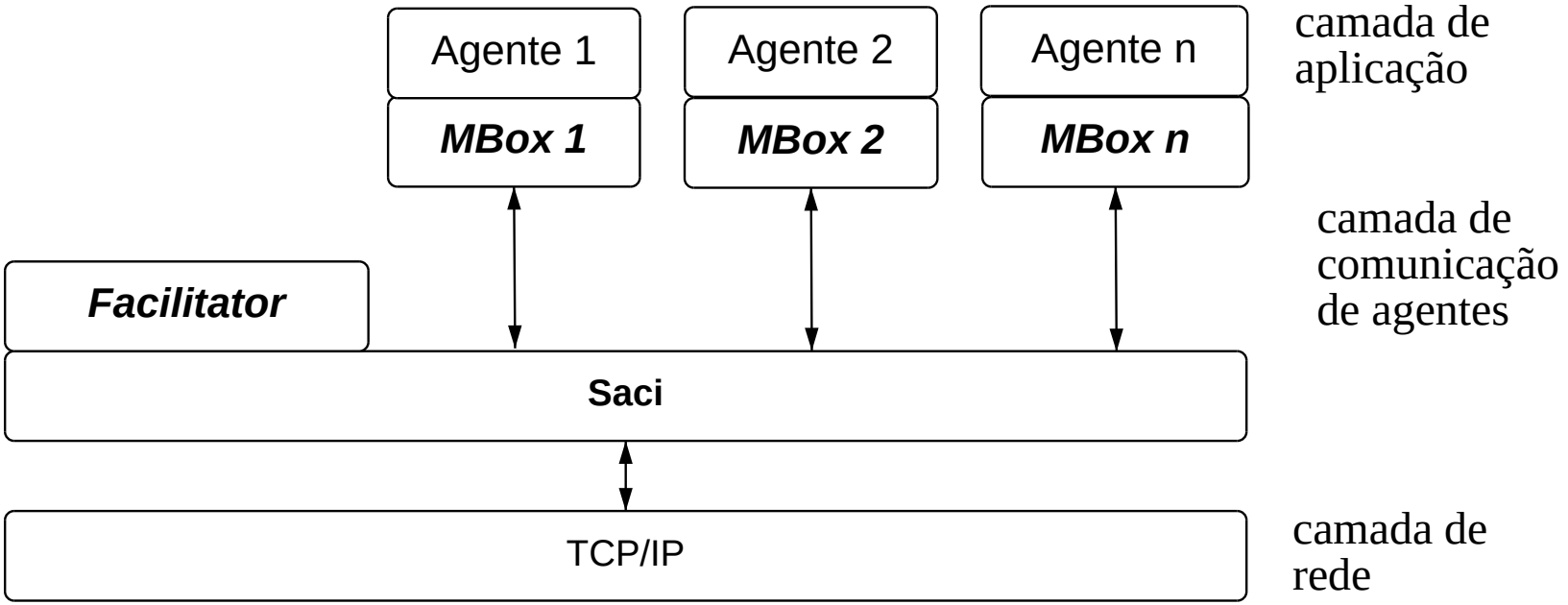
  Quais os agentes que respondem ao pedido “get sum” deste agente?

- Criar um agente, com nome “consulta”, na sociedade “**teste**”

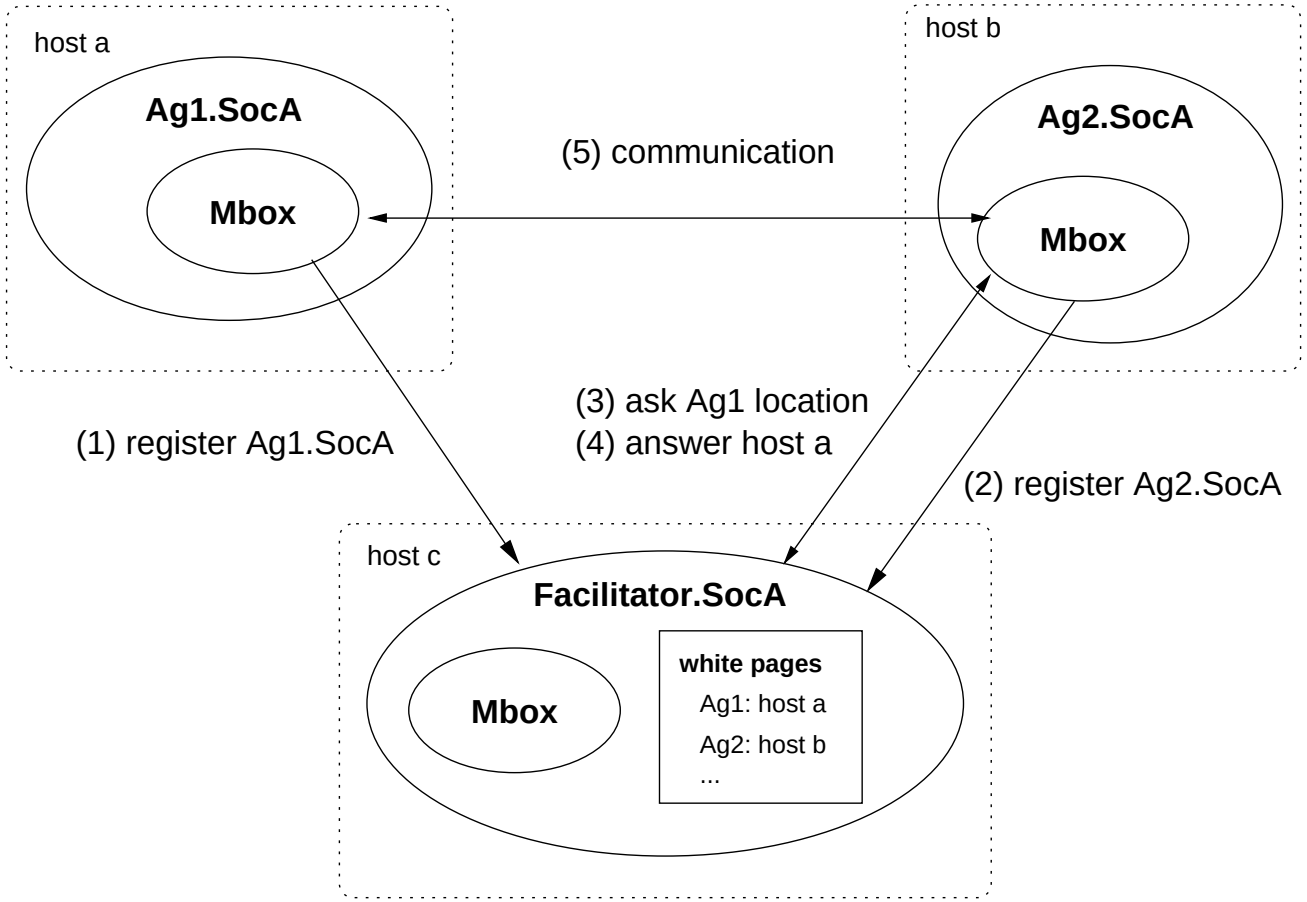
  Quais os agentes que respondem ao pedido “get sum” deste agente?

- Verificar no monitor (menu tools/monitor) se as sociedades e os agentes estão criados corretamente.

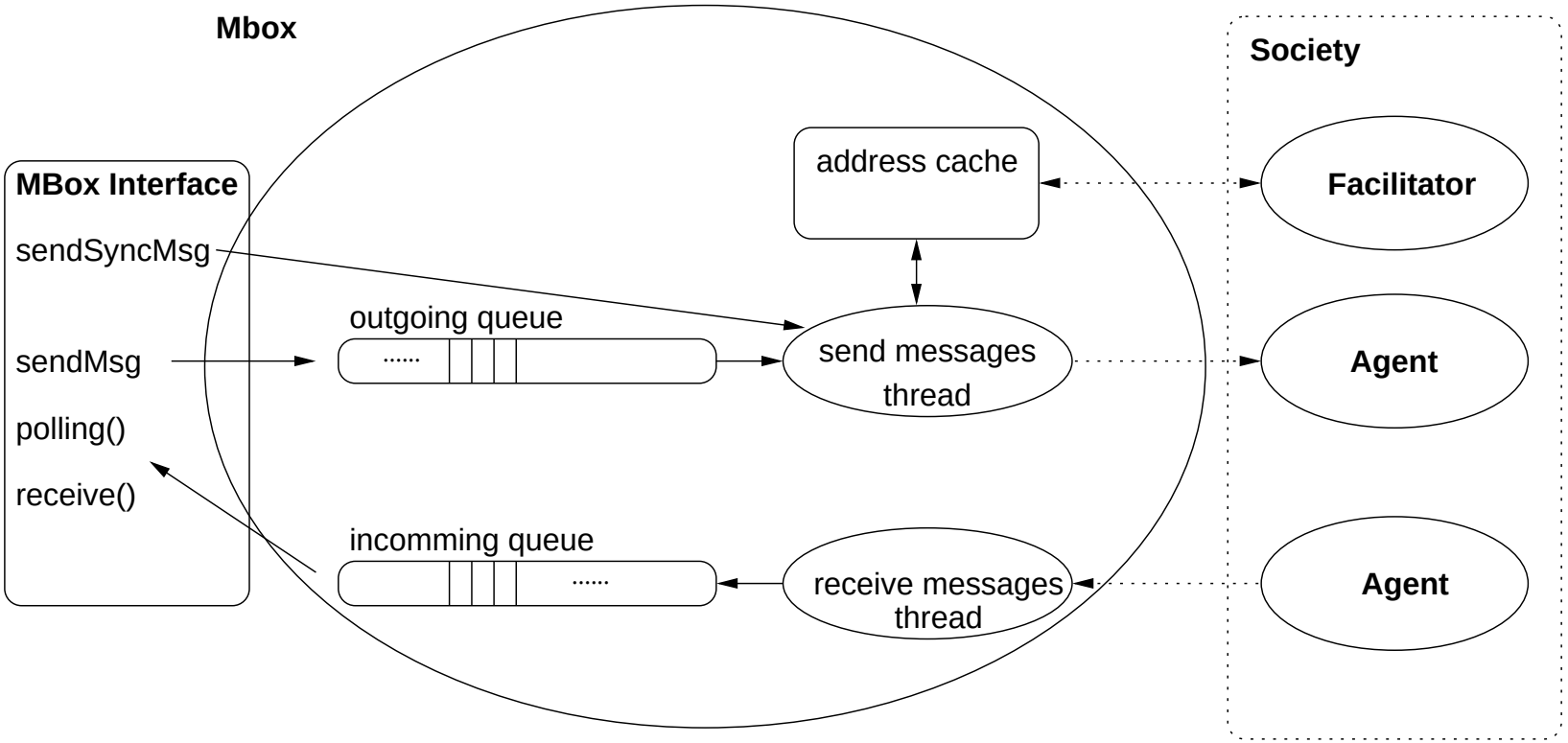
Arquitetura de **comunicação**



Envio e **recebimento** de mensagens



Arquitetura do **MBox**



Primeiro programa: agente **receptor**

```
 1 import saci.*;
 2 class Receptor {
 3   public static void main(String[] args) throws Exception {
 4     MBoxSAg mbox = new MBoxSAg("receptor");
 5     mbox.init();
 6     while (true) {
 7       Message m = mbox.polling();
 8       if (m != null) {
 9         System.out.println("recebi "+m);
10       }
11     }
12   }
13 }
```

Primeiro programa: agente **emissor**

```
1 import saci.*;
2 class Emissor {
3   public static void main(String[] args) throws Exception {
4     MBoxSAg mbox = new MBoxSAg("emissor");
5     mbox.init();
6     mbox.sendMsg(new
6        Message("(tell :receiver receptor :content oi)"));
7   }
8 }
```

Primeiro programa: **compilação** e **execução**

Em uma janela de comandos:

- cd para o diretório onde estão os fontes
- Incluir o arquivo `saci.jar` no CLASSPATH
  - ⋆ export CLASSPATH=/opt/saci/bin/saci.jar:.
  - ⋆ set CLASSPATH=c:\saci\bin\saci.jar;.
- Compilar os fontes

  ```
  javac *java
  ```

- Executar o receptor

  ```
  java Receptor
  ```

- Em outra janela de comandos, executar o emissor

  ```
  java Emissor
  ```

- Em outro computador
  - ⋆ Execute o SACI
  - ⋆ Conecte com o outro computador (menu launcher/connect)
  - ⋆ Execute o emissor

    ```
    java Emissor
    ```

API do MBox para envio e recebimento de mensagens

- sendMsg(*Message*)
- sendSyncMsg(*Message*)
- broadcast(*Message*)
- receive()
- receive(*Pattern*)
- polling([*timeout*])
- polling(*Pattern*, *timeout*)
- getMessages(*Pattern* [, *quantity*, *timeout*, *remove*])

API do MBox para protocolos KQML

- ask(*Message* [,*timeout*])
- forward(*Message*)

Exercício **MBox**

1. Faça com que o agente receptor, quanto receber uma mensagem com conteúdo “oi”, reponda “bem vindo” ao emissor da mensagem.
2. Faça com que o emissor utilize o protocolo **ask** para enviar o “oi” para o emissor e imprima a resposta recebida pelo ask.

```
1 import saci.*;
2 class Emissor {
3   public static void main(String[] args) throws Exception {
4     MBoxSAg mbox = new MBoxSAg("emissor");
5     mbox.init();
6     Message resposta = mbox.ask(new
7       Message("(ask :receiver receptor :content oi)"));
8     System.out.println("respota é "+resposta);
9   }
10 }
```

(Receptor)

```
4     MBoxSAg mbox = new MBoxSAg("receptor");
5     mbox.init();
6     System.out.println("Meu nome é "+mbox.getName());
7     Message pattern = new Message();
8     pattern.put("content" , "oi");
9     while (true) {
10      Message m = mbox.polling(pattern);
11      if (m != null) {
12        System.out.println("recebi "+m);
13        Message resposta = new Message("(tell :content bemVindo)");
14        resposta.put("receiver", m.get("sender"));
15        resposta.put("in-reply-to", m.get("reply-with"));
16        mbox.sendMsg(resposta);
17        System.out.println("respondi "+resposta);
18        ...
```

Manipuladores de mensagens

- Um manipulador de mensagem é um objeto que é "avisado" quando chega uma mesagem KQML de certo tipo.
- Cada mbox pode ter vários manipuladores, inclusive para o mesmo tipo de mensagem.
- O manipulador considera como "filtro" os seguintes valores:
  - ⋆ o onteúdo da mensagem,
  - ⋆ a linguagem do conteúdo,
  - ⋆ a ontologia e
  - ⋆ a performativa

Exemplo:

```
 8     mbox.addMessageHandler("oi", "ask", null, "apresentacao",
 9       new MessageHandler() {
10         public boolean processMessage(Message m) {
11           System.out.println("recebi "+m);
12           Message resposta =
13             new Message("(tell :content bemVindo)");
14           resposta.put("receiver", m.get("sender"));
15           resposta.put("in-reply-to", m.get("reply-with"));
16           mbox.sendMsg(resposta);
17           System.out.println("respondi "+resposta);
18           return true; // do not add m in mbox
19         }
20       }
21     );
```

Exercício **MBox** (2)

- Reescreva o agente Receptor utilizando message handlers.
- Existe alguma vantagem em utilizar esse tipo de "escuta"?

A classe **Agent**

- A classe `agent` permite utilizar alguns serviços de controle sobre o ciclo de vida do agente.
  - ⋆ Criar os agentes via interface do saci
  - ⋆ Matá-los
  - ⋆ Movê-los
- O desenvolvedor pode sobre-escrever os seguintes métodos da classe
  - ⋆ `initAg`: método que é chamado uma única vez quando o agente é criado.
  - ⋆ `run`: método que é chamado para o agente iniciar sua execução.
  - ⋆ `stopAg`: método que é chamado para o agente terminar sua execução.

Exemplo:

```
 1 import saci.*;
 2
 3 class Receptor extends Agent {
 4
 5   public static void main(String[] args) {
 6     Receptor r = new Receptor();
 7     if (r.enterSoc("receptor")) {
 8       r.initAg(null);
 9       r.run();
10     }
11   }
12
```

```
13  public void initAg(String[] a) {
14    try {
15      System.out.println("Meu nome é "+mbox.getName());
16
17      mbox.addMessageHandler("oi", "ask", null, "apresentacao",
18        new MessageHandler() {
19          public boolean processMessage(Message m) {
20            Message resposta =
21              new Message("(tell :content bemVindo)");
22            resposta.put("receiver", m.get("sender"));
23            resposta.put("in-reply-to", m.get("reply-with"));
24            try {
25              mbox.sendMsg(resposta);
26            } catch (Exception e) { System.err.println("Erro msg");}
27            System.out.println("respondi "+resposta);
28            return true; // do not m add in mbox
```

```
29            }
30          }
31        );
32      } catch (Exception e) {  System.err.println("Erro "+e); }
33    }
34
35    public void run() {
36      System.out.println("rodando....");
37    }
38
39    public void stopAg() {
40      System.out.println("terminando....");
41    }
42
43 }
```

Exercício **Compra de CD**

1. Faça um agente que tenha uma lista de título de CD para vender e responda às seguintes mensagens:
   - Cotação de preço para um título

     A mensagem de pedido de cotação deve ter o seguinte formato:
     - `(ask-one :ontology CD :content cotacao :titulo <uma string>)`

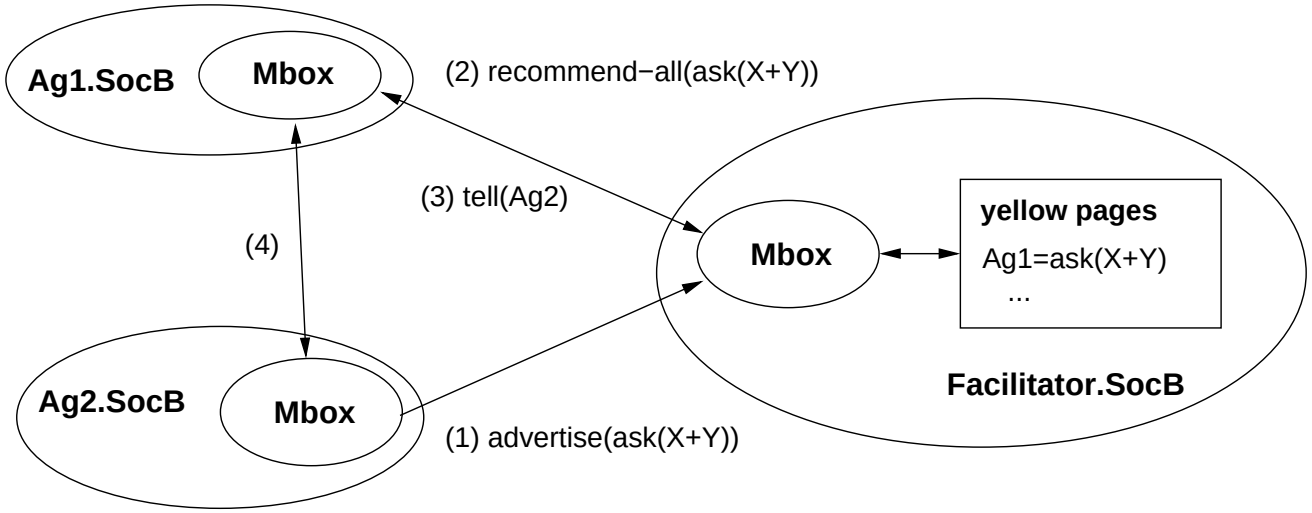
     A resposta deve ser
     - `(tell :ontology CD :content cotacao :valor <um valor real>)`
   - Venda de um título

     A mensagem de confirmação da compra vinda de um agente deve ter o seguinte formato:
     - `(tell :ontology CD :content confCompra :titulo <uma string> :quantidade <um inteiro>)`

2. Faça um programa de compra de CD que
   - Pergunte o preço para todos os vendedores
   - Compre o CD do vendedor mais barato

Páginas Amarelas

Anúncio de habilidades




```
(advertise :receiver    Facilitator
           :sender      Ag2
           :language    KQML
           :ontology    yp
           :content     (ask-one :receiver    Ag2
                                 :language    alg
                                 :ontology    math
                                 :content     "X + Y"))
```

```
(tell :receiver    Ag1
      :sender      Facilitator
      :in-reply-to id1
      :language    KQML
      :ontology    yp
      :content     (Ag2))
```

API do MBox para uso de **YP**

- boolean **advertise**(*performative*, *language*, *ontology*, *content*)
- String **recommendOne**(*performative*, *language*, *ontology*, *content*)
- Message **brokerOne**(*message [, timeout]*)

Exercício

1. Faça o agente receptor se registrar nas páginas amarelas
2. Reimplemente o agente emissor das seguintes formas
   - Utilizando o **recommendOne**
   - Utilizando o protocolo **brokerOne**
   - Utilizando o **recommendAll** e mandando "oi" para todos os agentes. O emissor deve aAguardar a resposta de todos e terminar (sugestão: usar getMessages).
3. Alterar o sistema de venda de CDs para utilizar páginas amarelas.

Monitoramento

Protocolo de Monitoramento

- Se um agente receber a mensagem

  `(subscribe :ontology saci-events :sender Ag)`

  passa a informar os eventos **sendMsg**, **receiveMsg**, **readMsg** para `Ag`

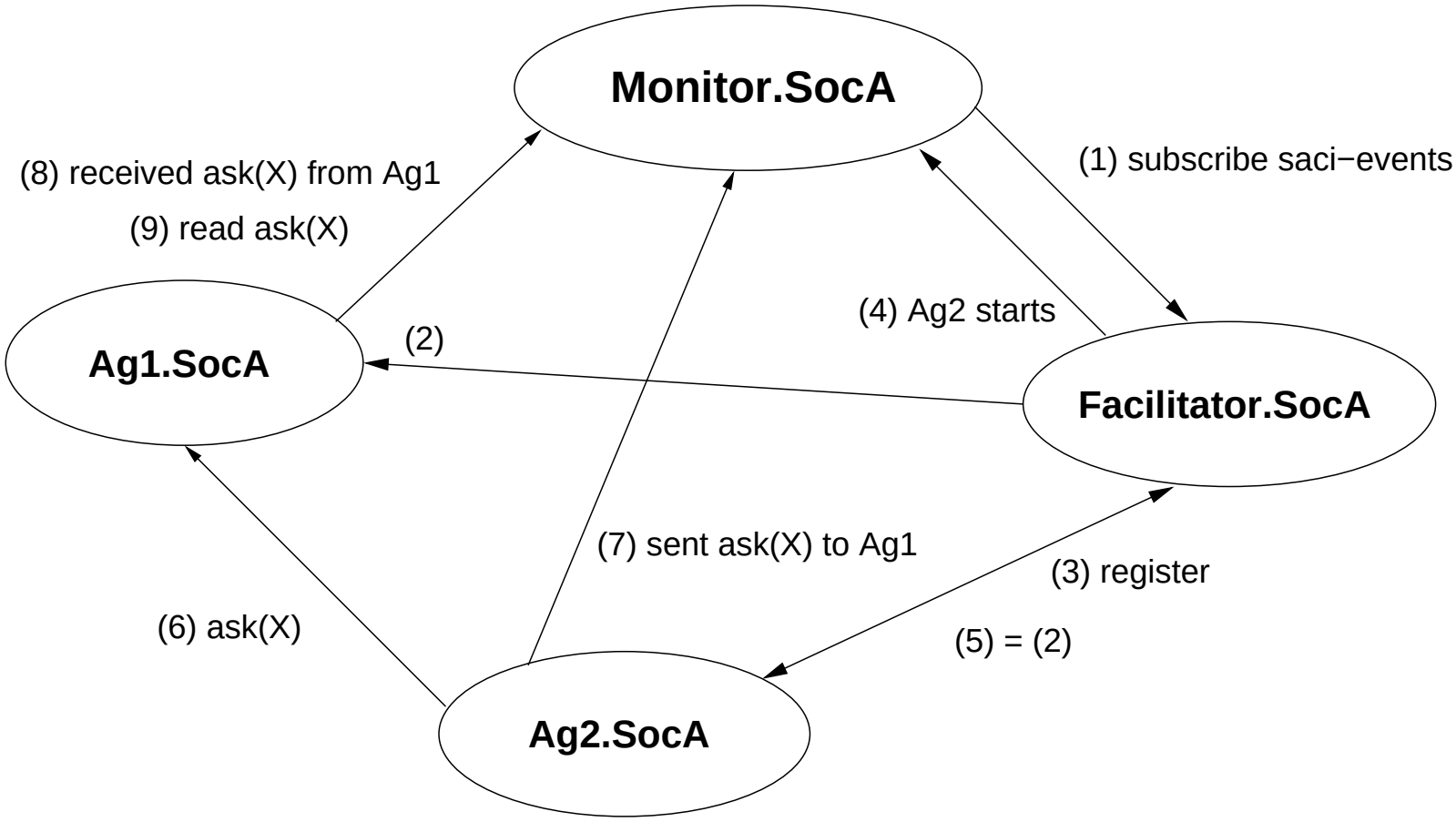
- Se o facilitador de uma sociedade receber a mensagem

  `(subscribe :ontology saci-events :sender Ag)`

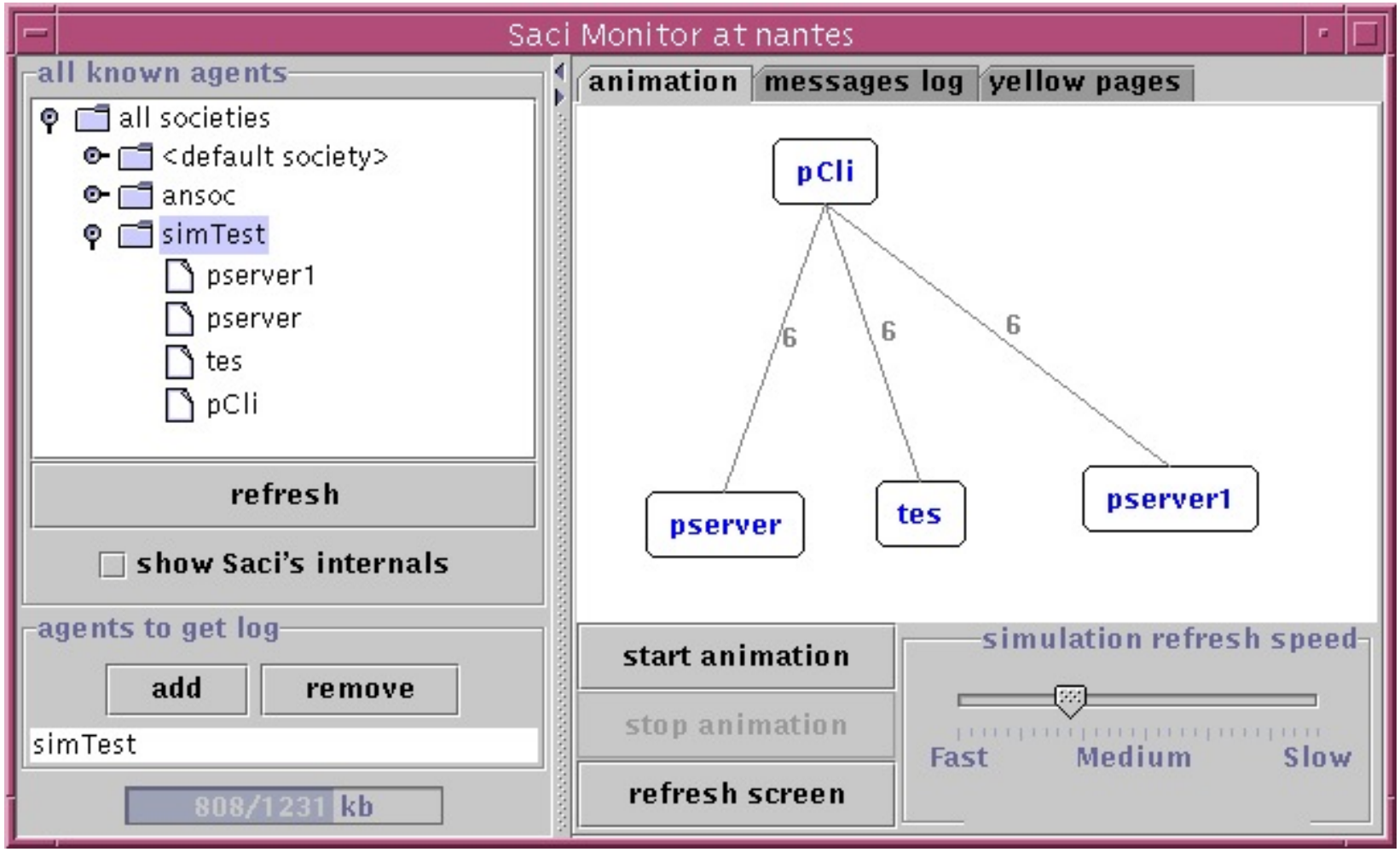
  - ⋆ passa a informar `Ag` dos eventos: **agentStart**, **agentStop**, **agentAdvertise**
  - ⋆ faz um broadcast da mensagem (assim todos os agentes passam a informar `Ag`)
  - ⋆ manda esta mensagem para cada novo agente que entrar na sociedade

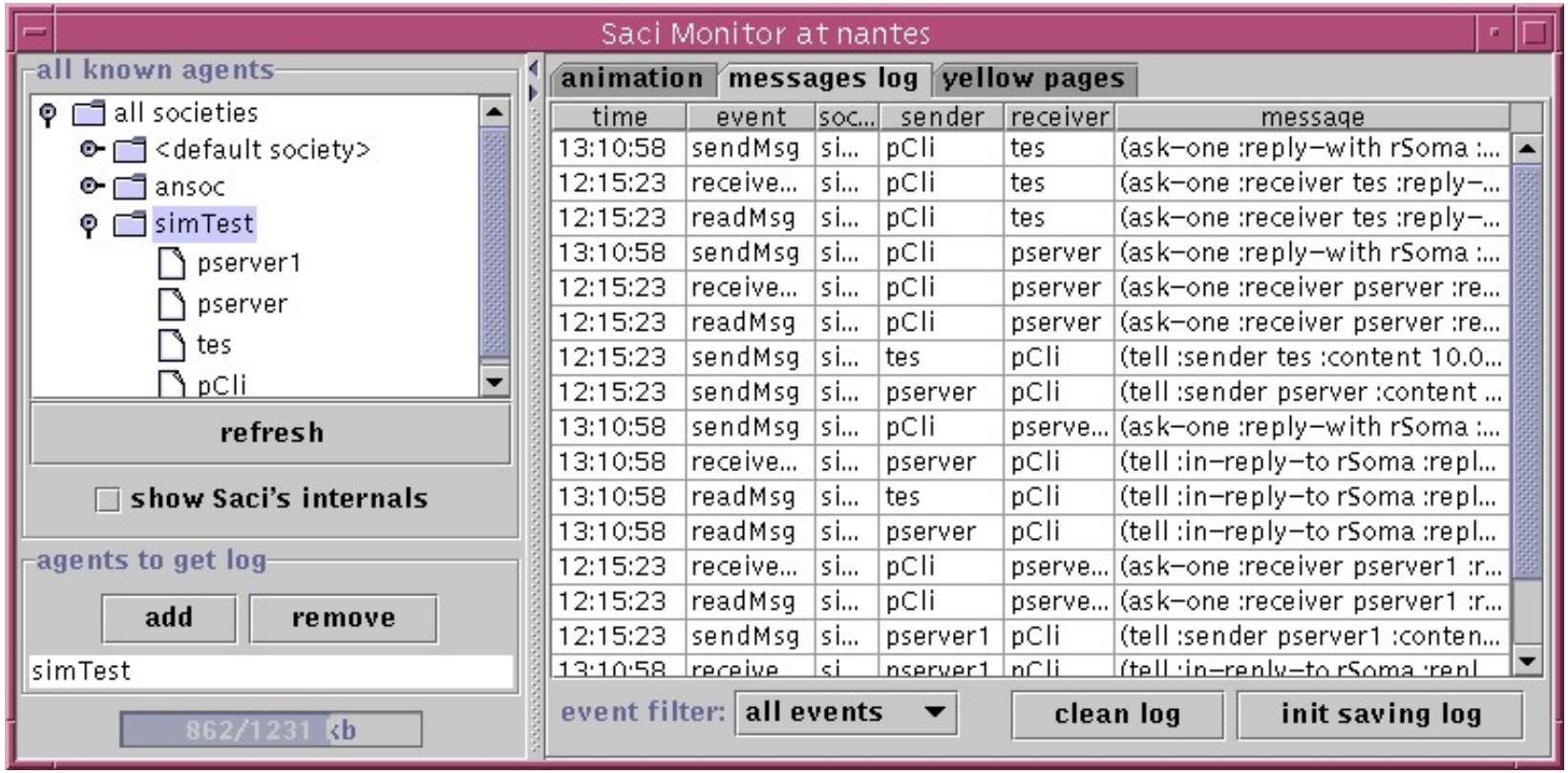
- Analogamente, existe a mensagem `unsubscribe`

- Este protocolo já está implementado no facilitador e no MBox



Exemplo de aplicação do monitoramento




Saci Monitor at nantes
all known agents
all societies
<default society>
ansoc
simTest
pserver1
pserver
tes
pCli
refresh
show Saci's internals
agents to get log
add
remove
simTest
862/1231 kb
animation
messages log
yellow pages
time
event
soc...
sender
receiver
message
13:10:58 sendMsg si... pCli tes (ask-one :reply-with rSoma :...
12:15:23 receive... si... pCli tes (ask-one :receiver tes :reply-...
12:15:23 readMsg si... pCli tes (ask-one :receiver tes :reply-...
13:10:58 sendMsg si... pCli pserver (ask-one :reply-with rSoma :...
12:15:23 receive... si... pCli pserver (ask-one :receiver pserver :re...
12:15:23 readMsg si... pCli pserver (ask-one :receiver pserver :re...
12:15:23 sendMsg si... tes pCli (tell :sender tes :content 10.0...
12:15:23 sendMsg si... pserver pCli (tell :sender pserver :content ...
13:10:58 sendMsg si... pCli pserve... (ask-one :reply-with rSoma :...
13:10:58 receive... si... pserver pCli (tell :in-reply-to rSoma :repl...
13:10:58 readMsg si... tes pCli (tell :in-reply-to rSoma :repl...
13:10:58 readMsg si... pserver pCli (tell :in-reply-to rSoma :repl...
12:15:23 receive... si... pCli pserve... (ask-one :receiver pserver1 :r...
12:15:23 readMsg si... pCli pserve... (ask-one :receiver pserver1 :r...
12:15:23 sendMsg si... pserver1 pCli (tell :sender pserver1 :conten...
event filter:
all events
clean log
init saving log

Mobilidade

API da classe agent para mobilidade

- Callbacks
  - ⋆ onMoving(*host*): este método do agente é chamado pelo saci antes do agente deixar um host.
  - ⋆ onMoved(): este método do agente é chamado pelo saci assim que o agente chega em um host e antes de iniciar sua execução (método run).

    O desenvolvedor pode sobre-escrever estes dois métodos para realizar alguma operação antes de migrar.
- move(*host*): move o agente para um host (o saci deve estar rodando em *host*)

  Os seguintes passos são executados na migração de um agente:
  - ⋆ o método onMoving é chamado
  - ⋆ copia o **estado** do agente para o host remoto (inclusive o

mailbox)

- o método onMoved é chamado
- inicia a thread/processo do agente no host remoto
- termina a thread/processo do agente no host local

Um agente móvel

```
import saci.*;
public class SampleMoveAg extends Agent {
    boolean go = true;
    public static void main(String[] args) {
        SampleMoveAg a = new SampleMoveAg();
        if (a.enterSoc("SampleMoveAg")) {
            a.run();
    }  }
    public void run() {
        if (go) {
            go = false;
            move("dijon.pcs.usp.br"); // and do something there
        } else {
            move("annecy.pcs.usp.br"); // and show the results
}  }  }
```

Exercício

- Faça um agente comprador de CD que vai até o host onde está um agente vendedor de CD e se comunique localmente com o ele. Ao final da conversa, retorna para o host de origem e mostra o resultado ao usuário.

Serviços

Launcher Demon

Executa em todas as máquinas onde irão funcionar agentes Saci. É um servidor com as seguintes funções:

- Facilitar a criação (remota) de agentes e facilitadores
- Serviço de MailBox para Applets (como o mail box é um processo servidor, um applet não tem permissão para executá-lo)
- Analisar e diponibilizar os recursos das aplicações

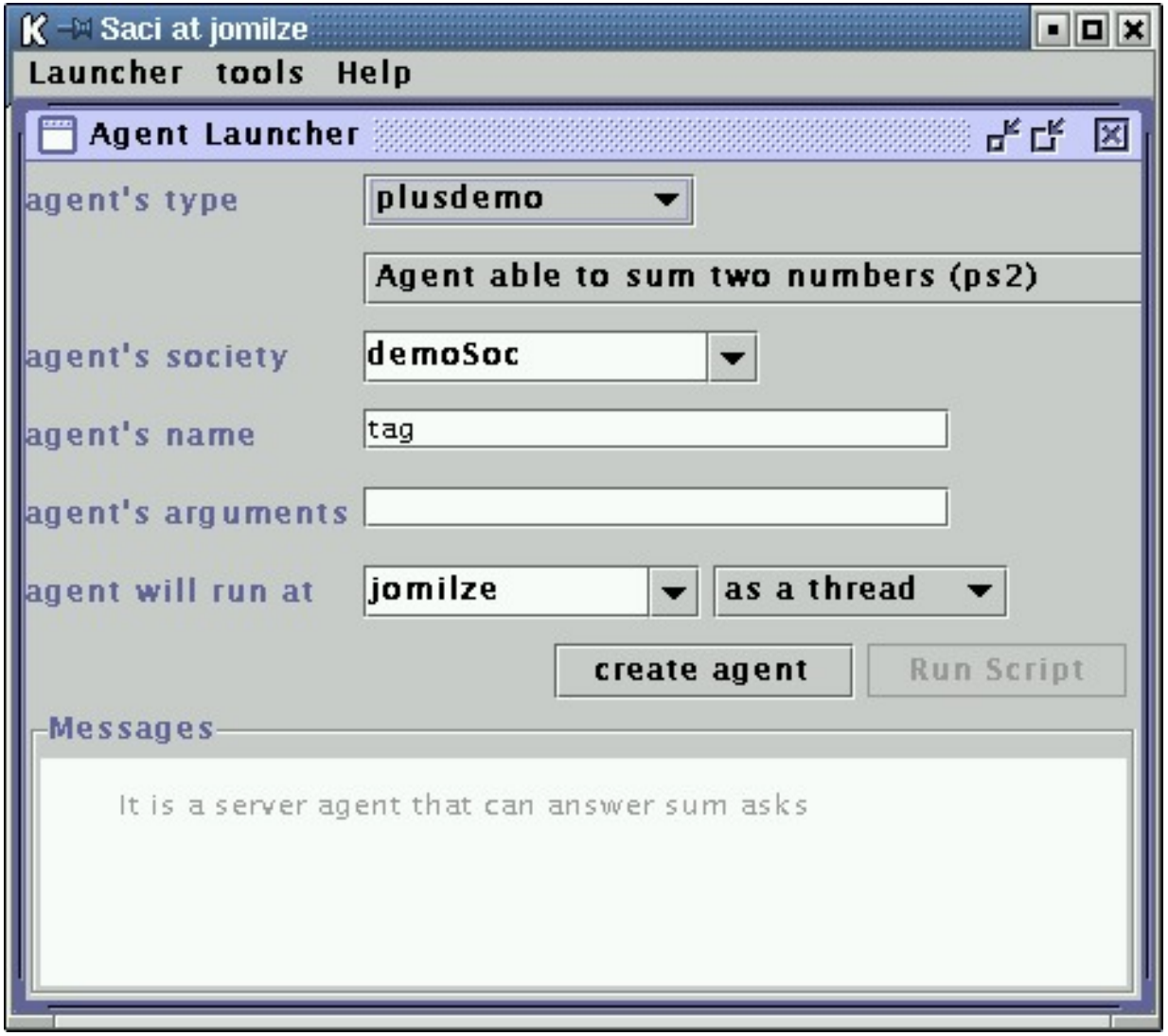
```
<agentType id="prey"
           description="Prey agent"
           defaultName="prey"
           defaultSociety="pursuitSoc"
           class="PreyAg">
</agentType>



<script id="s4h"
        description="Starts 4 hunters">
    <startAgent agId="hunter" args="U" />
    <startAgent agId="hunter" args="D" />
    <startAgent agId="hunter" args="R" />
    <startAgent agId="hunter" args="L" />
</script>
```

```
<script id="sall"
        description="Starts 1 world, 1 prey, and 4 hunters">

    <startAgent agId="world" />
    <sleep milisec="2000" />

    <startAgent agId="prey" />

    <runScript scriptId="s4h" />

</script>
```

SocLauncher

Solicita a criação de sociedades ao launcher. Pode-se escolher o host onde o facilitador da sociedade irá executar.

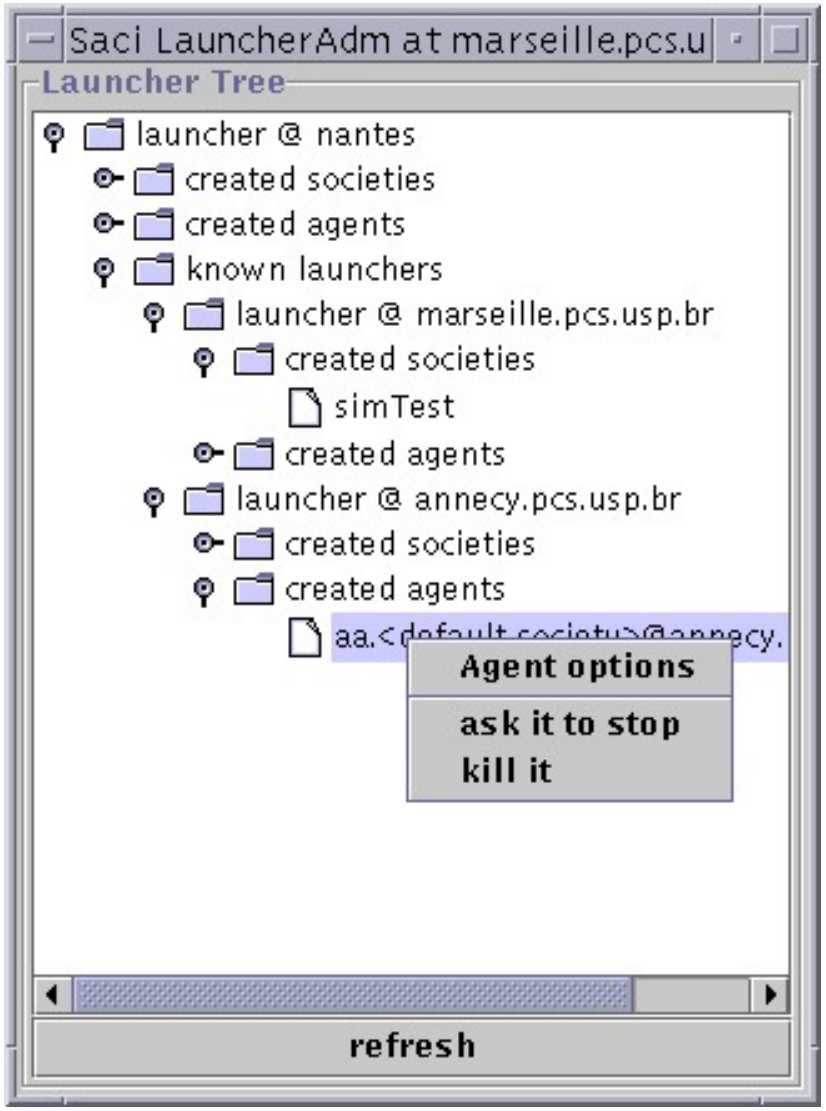
AgentLauncher



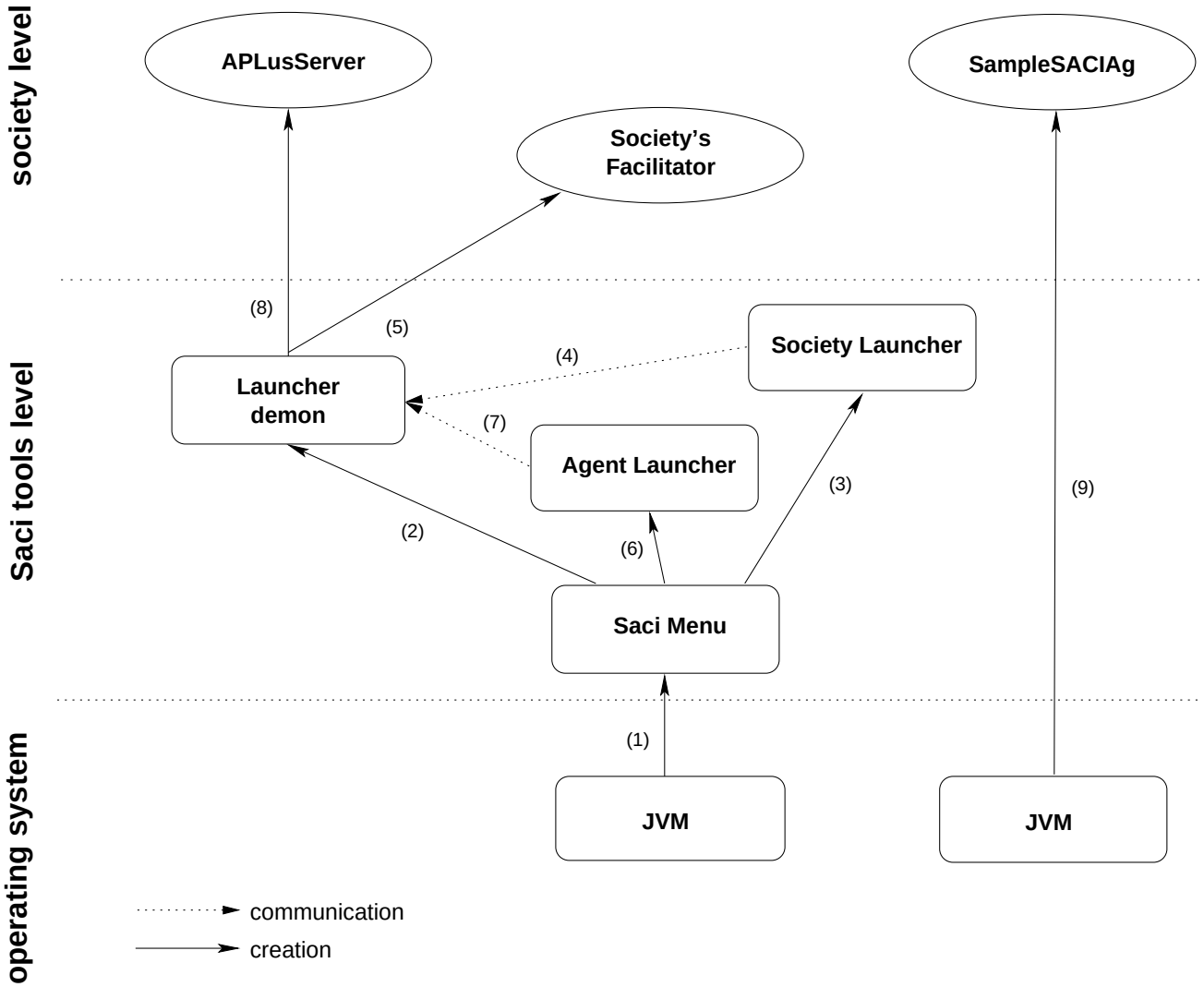
Launcher **Admin**

O Launcher Admin permite

- ver a árvore de launchers (que launcher conhece quais)
- parar um launcher
- ver que sociedades e agentes cada launcher criou
- parar
- mover agentes



Visão **Geral**


society level
Saci tools level
operating system
APLusServer
SampleSACIAg
Society's Facilitator
Society Launcher
Launcher demon
Agent Launcher
Saci Menu
JVM
JVM
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
communication
creation

Exemplo de uso da **API** do Launcher

```
Launcher l = Agent.getLauncher();

Command c1 = new Command(Command.START_AGENT);
c1.addArg("class", "PlusServer2");
c1.addArg("name", "john");
c1.addArg("args", "xpto xpto");
c1.addArg("qty", "5");
c1.addArg("host", "localhost");
// see application.dtd for other arguments

l.execCommand("",c1);
```

Exercícios

- Incluir os agentes Receptor e Emissor no menu do AgentLauncher
- Alterar o agent emissor para que, quando não houver nenhum agente receptor nas páginas amarelas, criar um agente receptor.

Comparação com outras ferramentas

Características

c1. Os agentes poderem ser executados como applets

c2. Mecanismo de monitoramento

c3. Simplicidade de utilização. (a) o número de palavras do código fonte e (b) o número de parâmetros de configuração

c4. Definição de protocolos de interação

	Critérios				
Ferramenta	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

Desempenho

Foi medido o número médio de mensagens trocadas por segundo entre dois agentes:

- servidor: anuncia seu serviço, espera requisições e as responde.
- 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 JVM como threads.

t2. Os dois agentes executando em JVMs diferentes mas no mesmo computador.

t3. Os dois agentes executando em computadores diferentes.

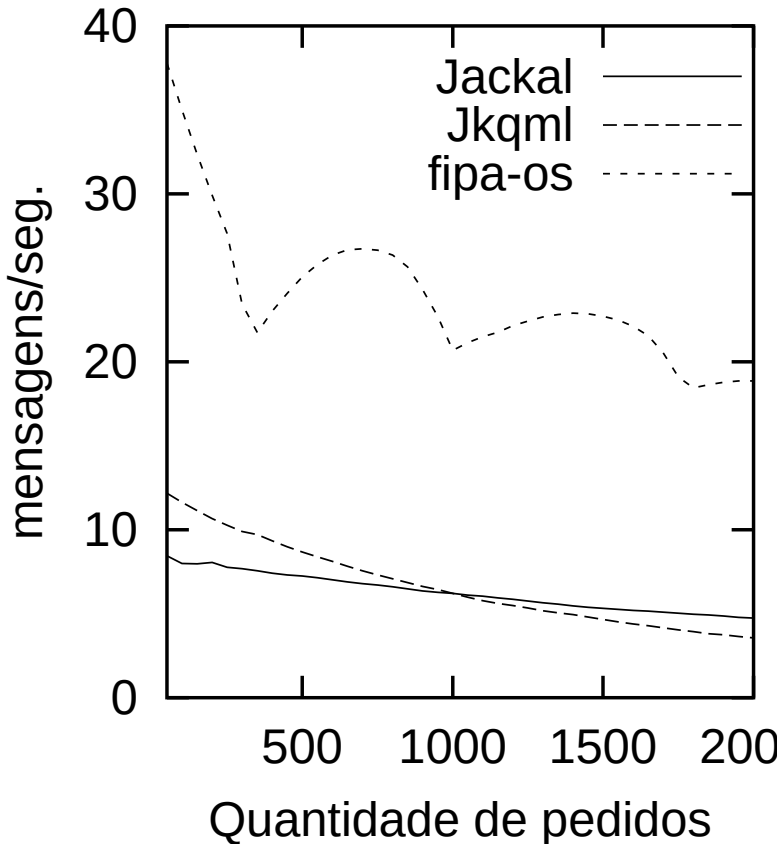
Resultado

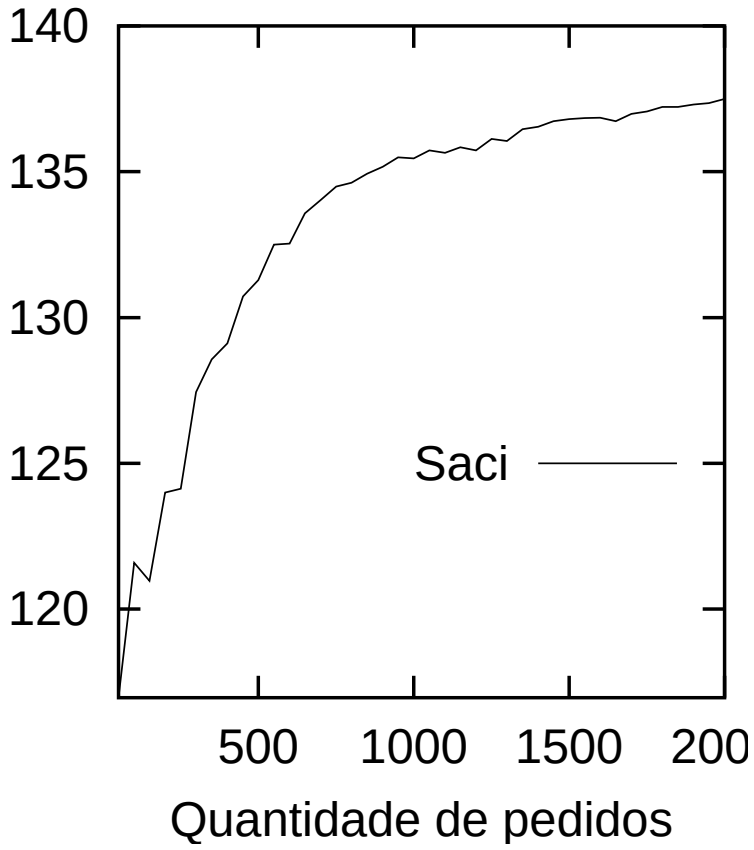
	Testes					
Ferramentas	t1		t2		t3	
Saci	2.412,54		197,24		137,49	
Jackal	6,03	400x-	6,64	29x-	4,73	29x-
JKQML	1,46	1652x-	2,55	77x-	3,56	38x-
FIPA-OS	17,95	134x-	25,13	7x-	18,86	7x-

Observações:

- nx-: n vezes mais lento que o Saci
- No saci, a comunicação entre agentes na mesma JVM é 12 vezes mais rápida

Desempenho no **tempo**





Conclusões

Trabalhos futuros

- Protocolos definidos pelo usuário
  - ⋆ Linguagem de descrição de protocolos (transição de estados)
  - ⋆ Como o agente irá usar o protocolo

Mais **informações**

Na página do Saci (`http://www.lti.pcs.usp.br/saci`) encontra-se:

- Manual de programação de agentes utilizando o Saci
  - ⋆ Conceitos utilizados na especificação do Saci
  - ⋆ Funcionamento do Saci
  - ⋆ Programação de agentes Saci
  - ⋆ Uso das facilidades do Saci
- Artigo publicado no IBERAMIA/SBIA'2000 comparando o desempenho do Saci com outras ferramentas similares [Hübner and Sichman, 2000].
- Documentação das classes
- Programas exemplo

Referências

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems. In *Proceedings of the European Conference on Cognitive Science*, Saint-Malo (France).

[Hübner and Sichman, 2000] Hübner, J. F. and Sichman, J. S. (2000). SACI: Uma ferramenta para implementação e monitoração da comunicação entre agentes. In Monard, M. C. and Sichman, J. S., editors, *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI, 15th Brazilian Symposium on AI (Open Discussion Track)*, pages 47–56, São Carlos. ICMC/USP. http://www.inf.furb.br/~jomi/pubs/2000/Hubner-iberamia2000.pdf.

[Labrou and Finin, 1997] Labrou, Y. and Finin, T. (1997). *A proposal for a new KQML specification*. UMBC, Baltimore.

[Weiß, 1999] Weiß, G., editor (1999). *Multiagent Systems: A modern approach to distributed artificial intelligence*. MIT Press, London.