IMPLEMENTAÇÃO DE UM AGENTE TAC SCM EM LINGUAGEM AGENTSPEAK(L)

Roteiro

- TAC
- TAC SCM
- Especificação do agente
- Especificação da interface de comunicação
- Implementação da interface de comunicação
- Implementação do agente
- Considerações finais

Objetivos

- exploração da linguagem AgentSpeak;
- exploração das técnicas e conceitos de programação orientada a agentes na construção de um agente para o simulador TAC SCM.
- analisar o potencial do interpretador ***Jason*** na implementação de agentes BDI;
- analisar o comportamento dos agentes implementados no interpretador ***Jason*** em um ambiente competitivo e com características sociais;
- desenvolver um sistema que assuma o papel gerencial de uma fábrica, planejando e coordenando as atividades da mesma.

Trading Agent Competition (TAC)

- É um fórum internacional;
- Promove e incentiva pesquisas relacionadas com agentes.
- Atualmente disponibiliza dois simuladores.

Simuladores do TAC

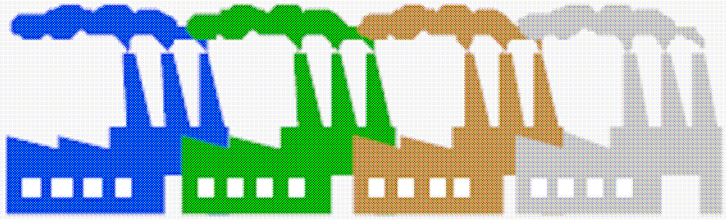
- TAC Classic: É um ambiente onde o papel do agente é montar pacotes de viagens para satisfazer os seus clientes.
- TAC SCM (Supply Chain Management): É um ambiente onde o agente planeja e coordena as atividades de um linha de montagem de computadores.

TAC SCM

- Simulador escolhido para a implementação.
- É uma competição entre seis agentes;
- Os agentes competem entre si por pedidos dos clientes e componentes dos fornecedores;
- A simulação é limitada por um determinado número de dias (representados por segundos).

TAC/SCM Scenario

Offers

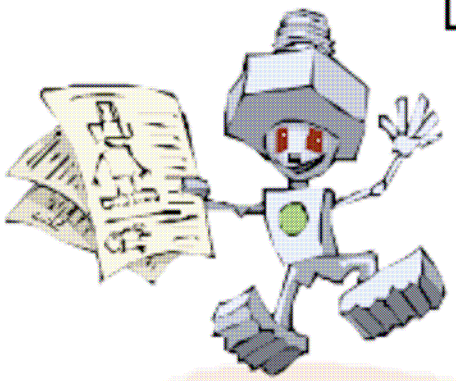


Suppliers
- strict MTO
- variable supply and prices

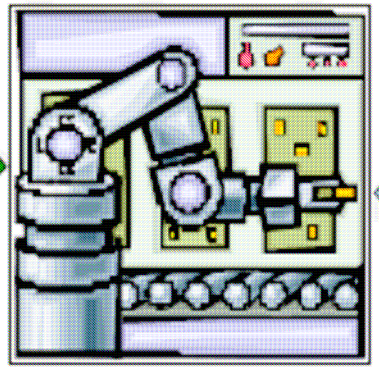
Agent
- automated
- optimizing

RFQs & orders

Manufacturers
- limited capacity
- competition for uncertain supplies and orders



Production schedule



Delivery schedule

Offers

RFQs & orders

Customers
- different levels and variability of demand
- "hard" due dates

Tarefas do Agente

- Negociar com os fornecedores;
- Fazer orçamentos dos pedidos feitos pelos consumidores;
- Planejar e controlar as atividades diárias da organização (linha de montagem)
- Ordenar o envio dos pedidos prontos aos clientes.

Linha de Montagem

- Constituída de fabrica e estoque.
  - O estoque armazena componentes e computadores.
  - A fabrica é responsável pela montagem dos computadores.
  - A fabrica possui uma limitação diária da capacidade de produção.
- Representada pelo simulador, mas comandada pelo agente.

Banco

- Cada agente possui uma conta no banco do simulador.
- A conta é iniciada sem nenhum dinheiro.
- Débitos são feitos na entrega dos componentes pelos fornecedores, ou por penalidades.
- Os créditos são adicionados a conta quando o cliente recebe o produto.
- O agente pode ficar com seu saldo negativo.

Fornecedores

- Total de oito fornecedores.
- Apenas, produzem componentes.
- Oferecem preço e prioridades aos agentes que tem melhor reputação.
- A capacidade máxima de produção é calculada diariamente.
- Cada componente é produzido por dois fornecedores.

Componentes

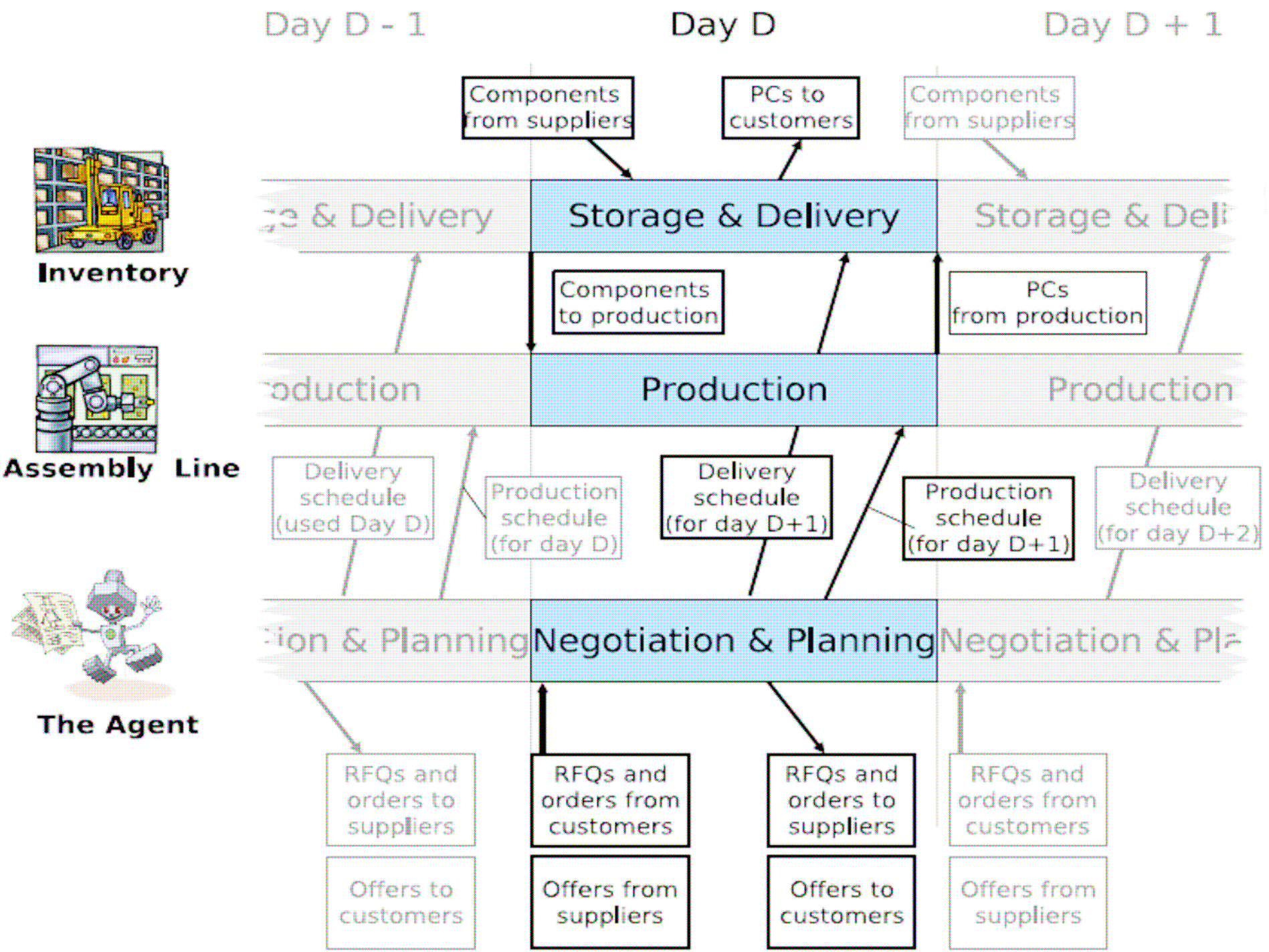
Componente	Preço base	Fabricante	Descrição
100	1000	Pintel	CPU Pintel, 2.0 GHz
101	1500	Pintel	CPU Pintel, 5.0 GHz
110	1000	IMD	CPU IMD, 2.0 GHz
111	1500	IMD	CPU IMD, 5.0 GHz
200	250	Basus, Macrostar	Placa mãe Pintel
210	250	Basus, Macrostar	Placa mãe IMD
300	100	MEC, Queenmax	Memória, 1 GB
301	200	MEC, Queenmax	Memória, 2 GB
400	300	Watergate, Mintor	Disco rígido, 300 GB
401	400	Watergate, Mintor	Disco rígido, 500 GB

Consumidor

- São controlados pelo simulador.
- Os pedidos possuem quantidades variadas.
- A quantidade de clientes é gerada randomicamente.
- Só selecionam as ofertas que atendem aos seus requesitos.
- Fazem requisições de diferentes tipos de PCs.

Combinação de PCs

Combinação	Componentes	Ciclos
1	100, 200, 300, 400	4
2	100, 200, 300, 401	5
3	100, 200, 301, 400	5
4	100, 200, 301, 401	6
5	101, 200, 300, 400	5
6	101, 200, 300, 401	6
7	101, 200, 301, 400	6
8	101, 200, 301, 401	7
9	110, 210, 300, 400	4
10	110, 210, 300, 401	5
11	110, 210, 301, 400	5
12	110, 210, 301, 401	6
13	111, 210, 300, 400	5
14	111, 210, 300, 401	6
15	111, 210, 301, 400	6
16	111, 210, 301, 401	7

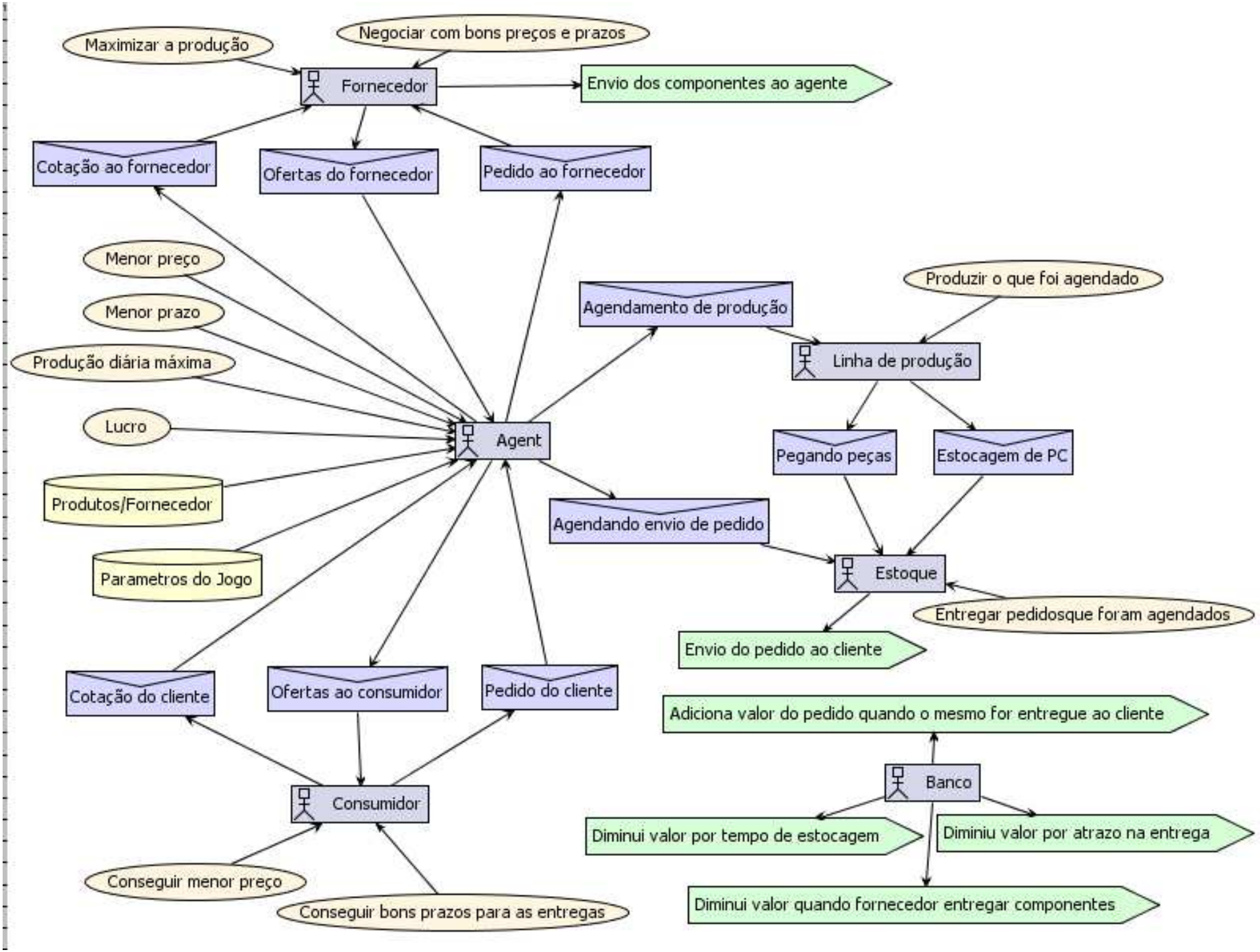

Day D - 1
Day D
Day D + 1
Components from suppliers
PCs to customers
Components from suppliers
Inventory
Storage & Delivery
Storage & Delivery
Components to production
PCs from production
Assembly Line
Production
Production
Delivery schedule (used Day D)
Production schedule (for day D)
Delivery schedule (for day D+1)
Production schedule (for day D+1)
Delivery schedule (for day D+2)
The Agent
Negotiation & Planning
Negotiation & Planning
RFQs and orders to suppliers
RFQs and orders from customers
RFQs and orders to suppliers
RFQs and orders from customers
Offers to customers
Offers from suppliers
Offers to customers
Offers from suppliers

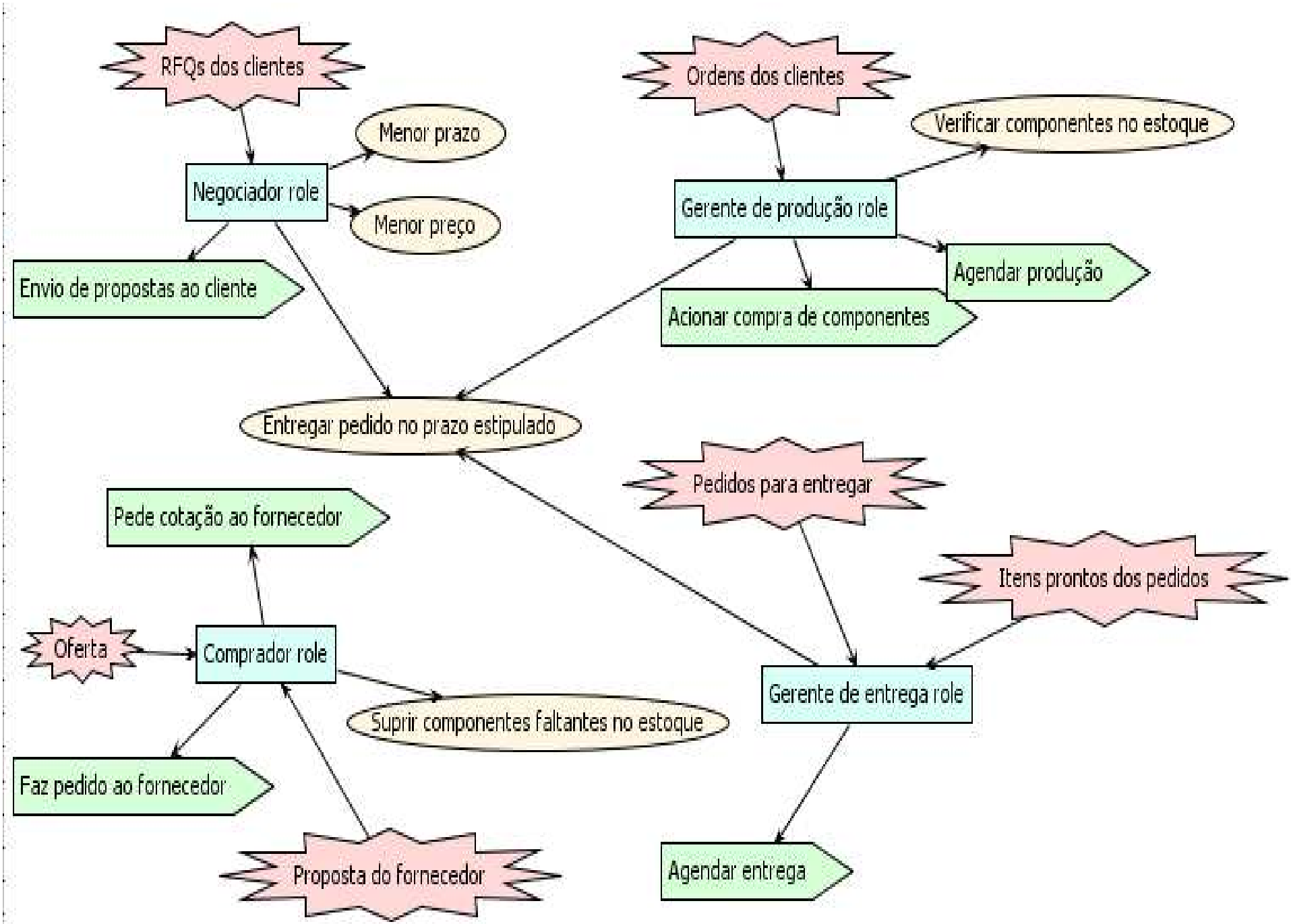
Especificação e implementação

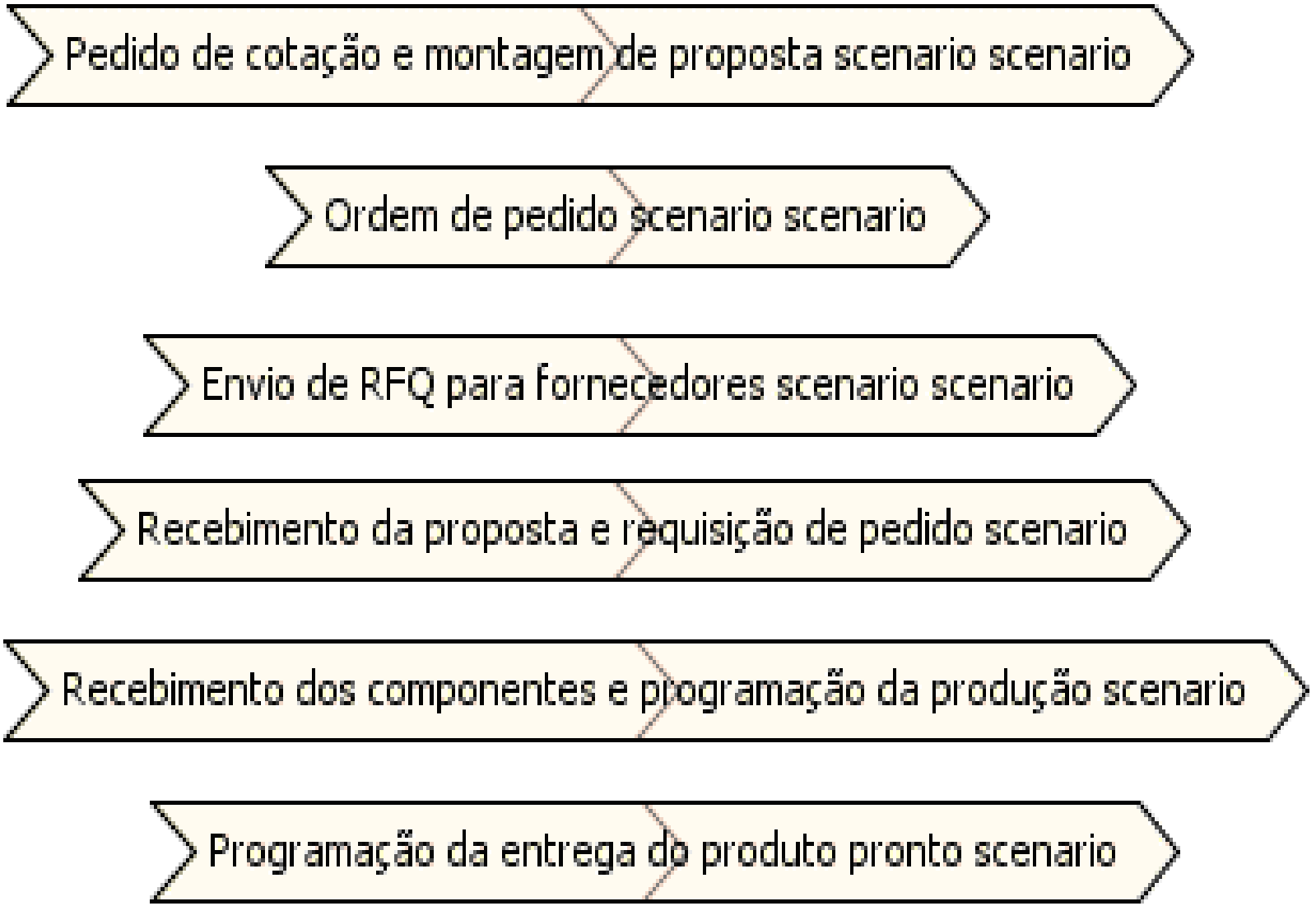
Ferramentas

- Prometheus Design Tool: Usado na modelagem do agente.
- Jason: Interpretador AgentSpeak(L), usado para implementação do agente.
- Enterprise Architect: Usado para criar diagramas da interface de comunicação.
- Eclipse: Editor da linguagem Java utilizado na construção da interface de comunicação.

Diagramas: agente


Maximizar a produção
Negociar com bons preços e prazos
Fornecedor
Envio dos componentes ao agente
Cotação ao fornecedor
Ofertas do fornecedor
Pedido ao fornecedor
Menor preço
Menor prazo
Produção diária máxima
Lucro
Produtos/Fornecedor
Parametros do Jogo
Agent
Agendamento de produção
Produzir o que foi agendado
Linha de produção
Pegando peças
Estocagem de PC
Agendando envio de pedido
Estoque
Entregar pedidosque foram agendados
Envio do pedido ao cliente
Cotação do cliente
Ofertas ao consumidor
Pedido do cliente
Adiciona valor do pedido quando o mesmo for entregue ao cliente
Banco
Consumidor
Diminui valor por tempo de estocagem
Diminiu valor por atrazo na entrega
Conseguir menor preço
Conseguir bons prazos para as entregas
Diminui valor quando fornecedor entregar componentes

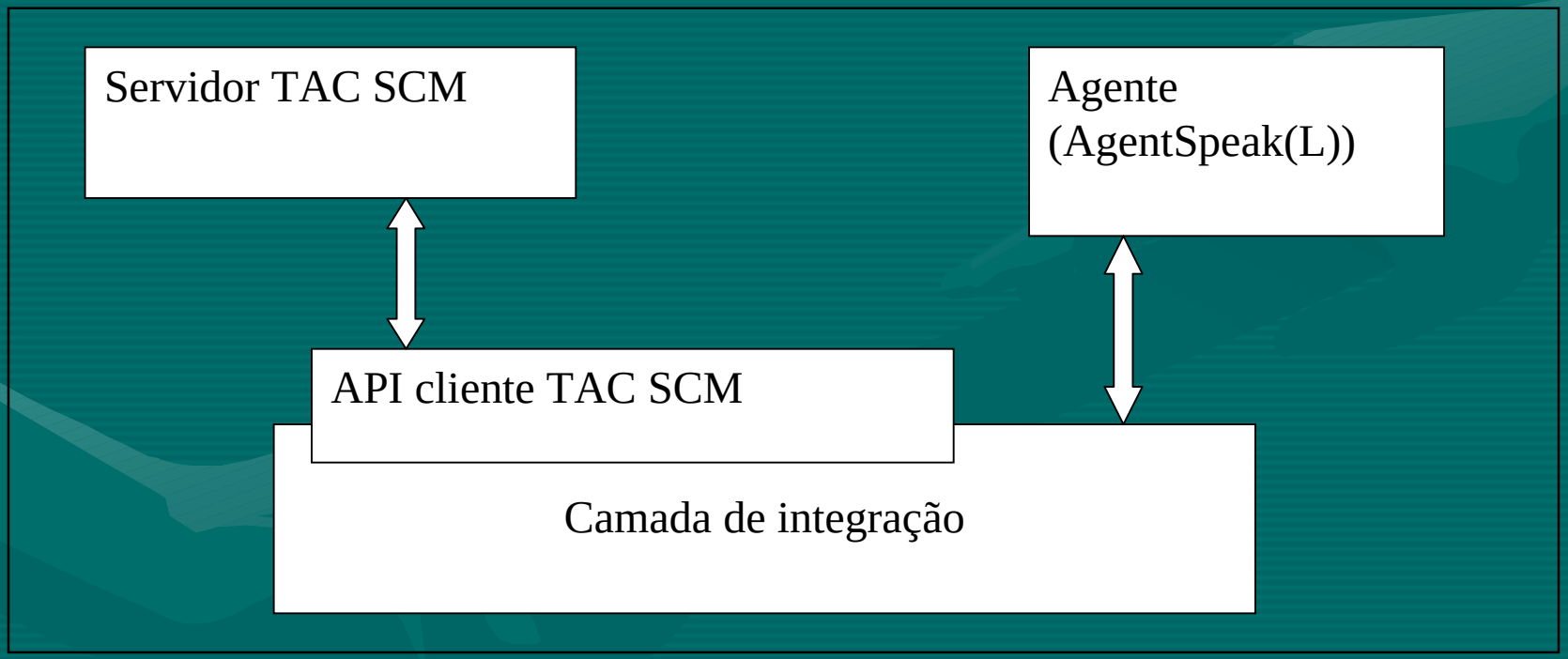

RFQs dos clientes
Ordens dos clientes
Menor prazo
Verificar componentes no estoque
Negociador role
Gerente de produção role
Menor preço
Envio de propostas ao cliente
Agendar produção
Acionar compra de componentes
Entregar pedido no prazo estipulado
Pedidos para entregar
Pede cotação ao fornecedor
Itens prontos dos pedidos
Oferta
Comprador role
Gerente de entrega role
Suprir componentes faltantes no estoque
Faz pedido ao fornecedor
Proposta do fornecedor
Agendar entrega

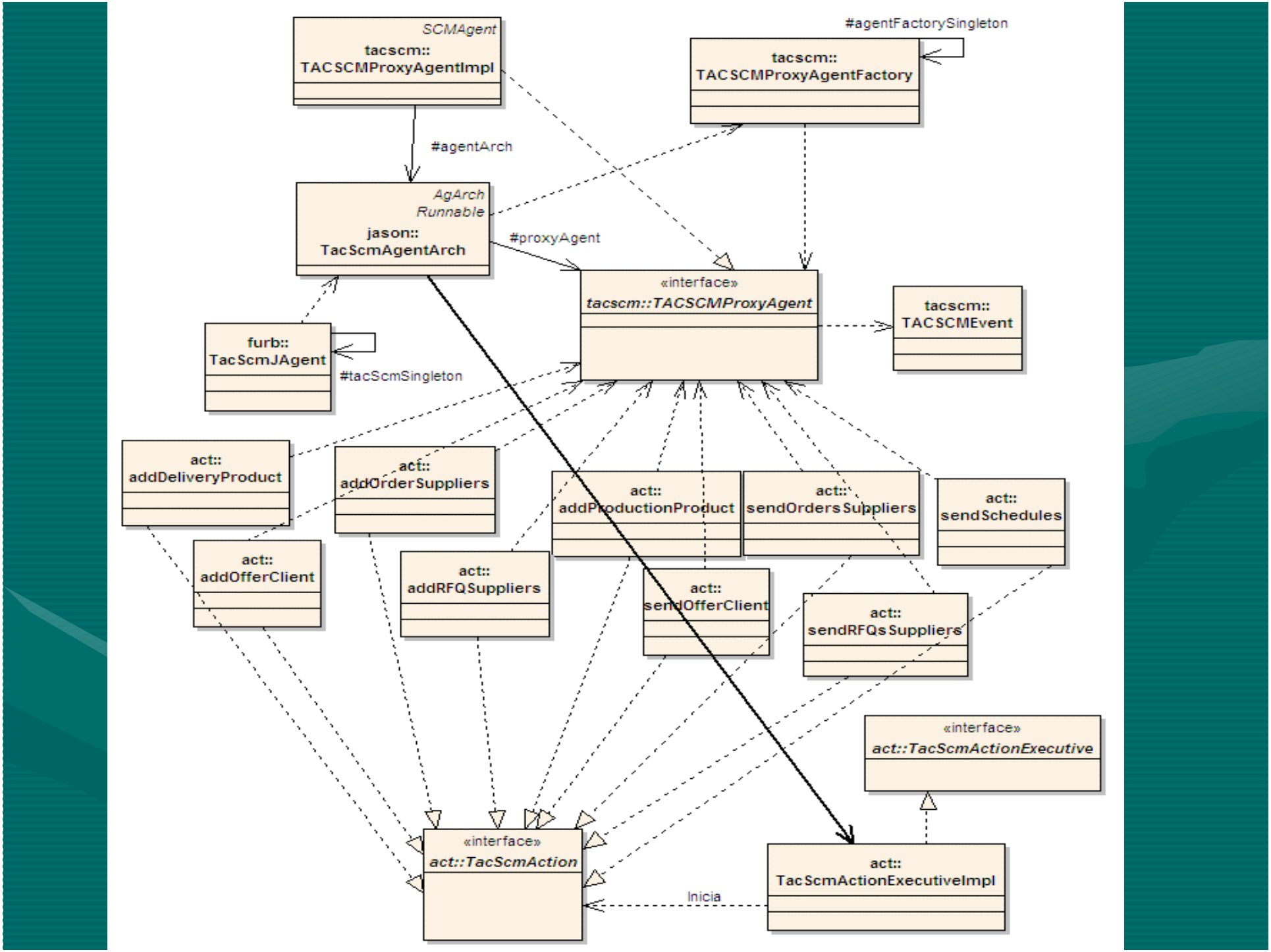

Pedido de cotação e montagem de proposta scenario scenario
Ordem de pedido scenario scenario
Envio de RFQ para fornecedores scenario scenario
Recebimento da proposta e requisição de pedido scenario
Recebimento dos componentes e programação da produção scenario
Programação da entrega do produto pronto scenario

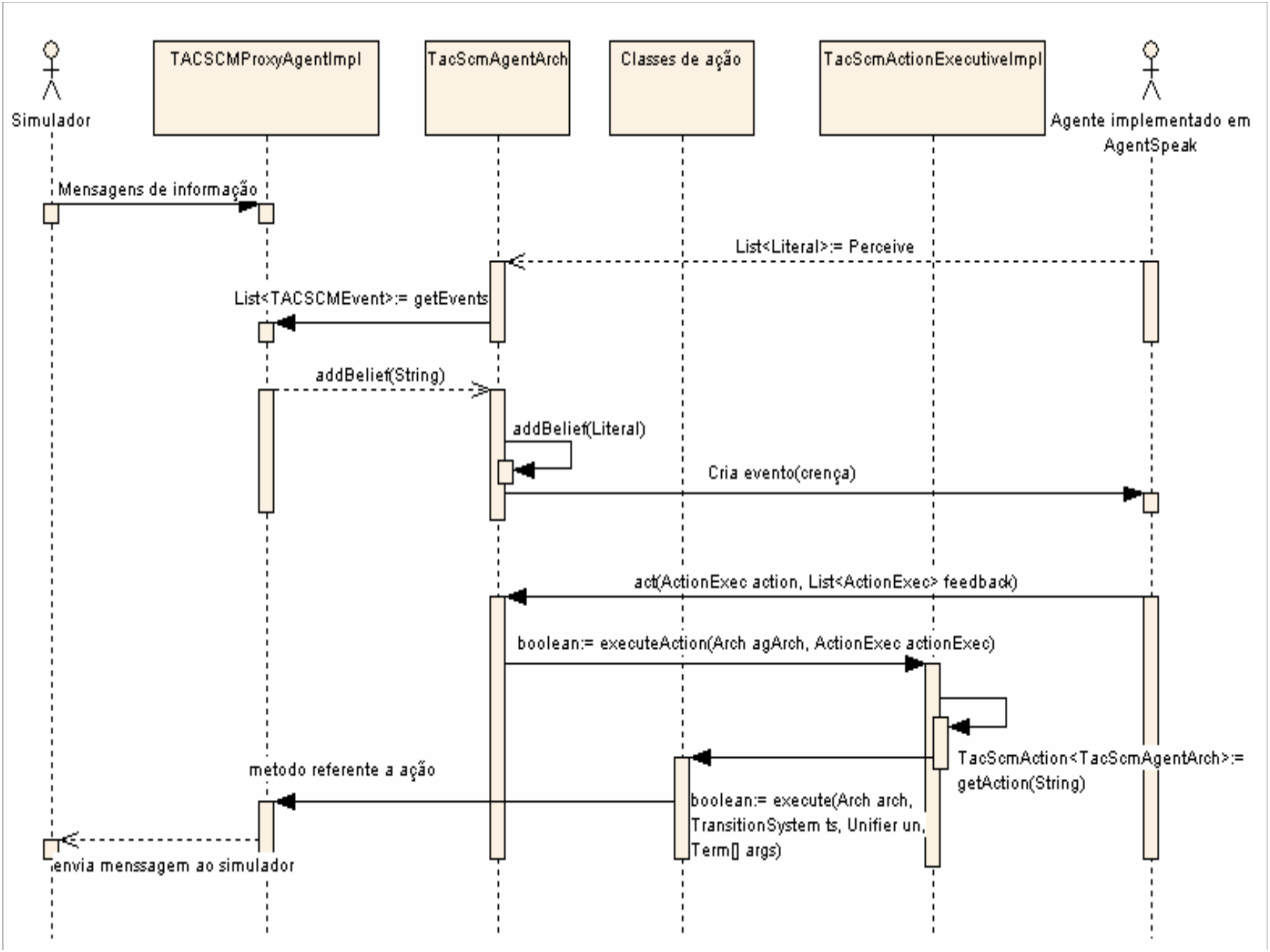
Interface de integração

- Faz a ligação entre a API do interpretador Jason e a API do TAC SCM (cliente).
- Implementado em Java.

Estrutura


Servidor TAC SCM
Agente
(AgentSpeak(L))
API cliente TAC SCM
Camada de integração


SCMAgent
tacscm::
TACSCMProxyAgentImpl
#agentFactorySingleton
tacscm::
TACSCMProxyAgentFactory
#agentArch
AgArch
Runnable
jason::
TacScmAgentArch
#proxyAgent
«interface»
tacscm::TACSCMProxyAgent
tacscm::
TACSCMEvent
furb::
TacScmJAgent
#tacScmSingleton
act::
addDeliveryProduct
act::
addOrderSuppliers
act::
addProductionProduct
act::
sendOrdersSuppliers
act::
sendSchedules
act::
addOfferClient
act::
addRFQSuppliers
act::
sendOfferClient
act::
sendRFQsSuppliers
«interface»
act::TacScmActionExecutive
«interface»
act::TacScmAction
act::
TacScmActionExecutiveImpl
Inicia


Simulador
TACSCMProxyAgentImpl
TacScmAgentArch
Classes de ação
TacScmActionExecutiveImpl
Agente implementado em AgentSpeak
Mensagens de informação
List<Literal>:= Perceive
List<TACSCMEvent>:= getEvents
addBelief(String)
addBelief(Literal)
Cria evento(crença)
act(ActionExec action, List<ActionExec> feedback)
boolean:= executeAction(Arch agArch, ActionExec actionExec)
TacScmAction<TacScmAgentArch>:= getAction(String)
metodo referente a ação
boolean:= execute(Arch arch, TransitionSystem ts, Unifier un, Term[] args)
envia menssagem ao simulador

Implementação da interface de comunicação

```
public TacScmAgentArch(String asSource) throws JasonException {
    super();
    this.actionExecutive = new TacScmActionExecutiveImpl();
   try {
        Settings settings = new Settings();

        ClassParameters bbPars = new ClassParameters();
        bbPars.className =
                                DefaultBeliefBase.class.getCanonicalName();

        this.initAg(Agent.class.getCanonicalName(), bbPars,
        asSource, settings);
        this.setArchInfraTier(new CentralisedAgArch());
    } catch (JasonException e) {
        logger.log(Level.SEVERE, "Init error", e);
    }
    try {
        TACSCMProxyAgentFactory proxyAgentFactory =
                  TACSCMProxyAgentFactory.getTACSCMProxyAgentFactory();
        URL config =
   TacScmAgentArch.class.getResource("/br/furb/agent.conf");

        proxyAgent =
    proxyAgentFactory.createTACSCMProxyAgent(this, config);
    } catch (URISyntaxException e) {
        throw new JasonException(e.getMessage());
    }
}
```

```
public TACSCMProxyAgent createTACSCMProxyAgent(TacScmAgentArch agentArch, URL
configFile) throws URISyntaxException {
                                                .
                                                .
                                                .

    TACSCMProxyAgent agent = null;
                                                .
                                                .
                                                .

   SimClient client = new SimClient(configManager, host, port,
                                                        name, password, agentImpl);

    try {
         agent = (TACSCMProxyAgent) client.getAgent();
    } catch (Exception e) {
         logger.log(Level.SEVERE, "could not create AgentImpl object of class "+
agentImpl, e);
         TacScmJAgent.getInstance().fatalError("no agent implementation available");
    }

          agent.init(agentArch);


                return agent;
}
```

SCMAgent

```
protected void messageReceived(Message message) {
     Transportable content = message.getContent();

     } else if (content instanceof RFQBundle) {
          RFQBundle rfqBundle = (RFQBundle)
content;

          this.customerAddress =
message.getSender();
          this.customerOfferBundle = null;
          this.currentCustomerRFQs = rfqBundle;
          handleCustomerRFQs(rfqBundle);

    } else if (content instanceof OrderBundle) {

}
```

```
protected void handleCustomerRFQs(RFQBundle rfqBundle) {
        String str;
        for (int i = 0; i < rfqBundle.size(); i++) {
                // identificador unico da rfq
                str = rfqBundle.getRFQID(i)+",";
                // index da RFQ
                str+= i+",";
                // tipo de PC
                str+= rfqBundle.getProductID(i)+",";
                //quantidade pedida
                str+= rfqBundle.getQuantity(i)+",";
                //preço ao qual o consumidor quer pagar
                str+= rfqBundle.getReservePricePerUnit(i)+",";
                // o valor da penalidade por atrazo
                str+= rfqBundle.getPenalty(i)+",";
                //data para entrega
                str+= rfqBundle.getDueDate(i)+",";
                // data de validade para responder a RFQ
                str+= rfqBundle.getValidDate();

            this.agentArch.addBelief("client_rfq("+str+")");
        }

        //Atualizando as crenças do agente sobre o estoque
        this.updateCurrentInventory();
}
```

```java
public void addBelief(String belief) {
    logger.finer("Adding belief '"+belief+"'");
    this.addBelief(Literal.parseLiteral(belief));
}

public void addBelief(Literal belief) {
    getTS().getAg().addBel(belief);
}



protected void handleSimulationStatus(SimulationStatus status) {
    this.addEvent("update_current_date("+status.getCurrentDate()+")");
}



public List<Literal> perceive() {
    List<TACSCMEvent> events = proxyAgent.getEvents();
    List<Literal> percepts = new Vector<Literal>();
    for (TACSCMEvent event : events) {
        logger.finer("Handling event "+event);
        percepts.add(Literal.parseLiteral(event.getEvent()));
    }
    return percepts;
}
```

```
public void act(ActionExec action, List<ActionExec> feedback) {

    ActionExec acExec = getTS().getC().getAction();
    if (acExec == null) {
        return;
    }
    Term acTerm = acExec.getActionTerm();
    logger.fine("doing: "+acTerm);
    boolean actionResult;
    try {
        actionResult = actionExecutive.executeAction(this, acExec);
        logger.fine("done: '"+acTerm+"' ->"+actionResult);
    } catch (JasonException e) {
        actionResult = false;
        e.printStackTrace();
    }
    acExec.setResult(actionResult);
    getTS().getC().getFeedbackActions().add(acExec);
}
```

```
    public boolean executeAction(TacScmAgentArch agArch, ActionExec actionExec)
                                                          throws   JasonException {
      String actionName = actionExec.getActionTerm().getFunctor();
      TacScmAction<TacScmAgentArch> action = getAction(actionName);

      TransitionSystem ts = agArch.getTS();
      Term args[] = actionExec.getActionTerm().getTermsArray();
      Unifier un = ts.getC().getSelectedIntention().peek().getUnif();

      return action.execute(agArch, ts, un, args);
   }
```

```
     public boolean execute(TacScmAgentArch arch, TransitionSystem ts, Unifier un,
Term[] args) {
        try {
           TACSCMProxyAgent agent =  arch.getProxyAgent();
           agent.sendSchedules();
           return true;
        } catch (Exception e) {
            logger.warning("Error in internal action 'sendSchedules'! "+e);
        }
        return false;
    }
```

Implementação do agente

Eventos, crenças e ações

```
+update_current_date(D)
        <- -+current_date(D);
           sendSchedules;
           -+cycles_production(0).


+client_rfq(RfqId,RfqIndex,ProdId,Qtd,PriceUn,Penalty,DueDate,Val
idDate)
      :pc_config(ProdId,CPU,MB,Me,HD,PriceUnBase,ProdCycles,Desc)
      <- !price_definition(PriceUn,PriceUnBase, PriceFinal);
         addOfferClient(DueDate, RfqIndex, PriceFinal);
     !remove_client_rfq(RfqId,RfqIndex,ProdId,
                        Qtd,PriceUn,Penalty,DueDate,ValidDate);
         !send_offers_customer.
```

Considerações finais

- O PDT auxilia a clarear as idéias de todo o processo.
- AgentSpeak(L) possui um nível de abstração elevado.
- O Jason como interpretador é de grande ajuda.
- A complexidade do simulador TAC SCM ajudou na curva de aprendizado da linguagem AgentSpeak(L).

Considerações finais

- Dificuldade diante a falta de documentação das APIs.
- Desafios: novas ferramentas, nova linguagem e novo paradigma.