Apêndice I – Código Fonte das Meta-Tuplas e Reações Presentes no Marketplace Implementado

```
public class MessageProtectionMetaTuple extends MetaTuple{
    public MessageProtectionMetaTuple( Operation op )
    {
        /*Associa a reação "MessageProtectionReaction" à operação
          específica "op" realizada por qualquer agente sobre qualquer men-
          sagem.
         A operação "op" em geral é especificada como a leitura (READ) ou
         retirada (TAKE) de mensagens.
        */
        super( new MessageProtectionReaction() ,
               op ,
               AgentID.class ,
               TRexMessage.class );
    }
}
```

Fragmento de Código 5 – Meta-tupla de proteção à leitura de mensagens por agentes que não são seus destinatários

```
public SuperTuple react( SuperTuple t , Operation op , AgentID id )
{
   SuperTuple retVal = null;
   //a tupla que origina esta reação sempre será uma mensagem
   TRexMessage msg = ( TRexMessage ) t;
   //recupera o destinatário da mensagem
   AgentID receiver = msg.getReceiver();
   //se o destinatário da mensagem for realmente o agente que
   //solicitou sua leitura ou retirada a mensagem será
   //retornada para ele
   if( receiver.equals( id ) )
   {
      retVal = msg;
   }
   //caso contrário lhe será retornado "null" indicando que a
   //tupla procurada não existe para a visão do agente
   else
   {
      retVal = null;
      //Caso a operação solicitada tenha sido a retirada da tupla
      //ela será reescrita no ambiente (espaço base)
      if ( op.equals( Operation.TAKE ) )
      {
         TRexAccessor acc = SingletonTRexAccessor.getInstance();
         acc.getBaseSpace().write( msg );
      }

   }
   return retVal;

}
```

Fragmento de Código 6 – Reação (MessageProtectionReaction) associada à Meta-Tupla de proteção à leitura

```
public class PersistenceMetaTuple extends MetaTuple
{
    public PersistenceMetaTuple( AgentID messageSender ,
                                 AgentID persistentAgent )
    {
        /*Associa a reação "PersistenceReaction" à escrita realizada de
          qualquer mensagem endereçada ao agente que se tornou persistente
          (persistentAgent) realizada por um agente específico
          (messageSender).
          Este agente específico deve ser o agente cujas mensagens devem
          acordar o agente persistente.*/
         super( new PersistenceReaction( ) ,
                Operation.WRITE ,
                messageSender ,
                new TRexMessage( persistentAgent ) );
    }

}
```

Fragmento de Código 7 – Meta-Tupla para a ativação de agentes persistentes

```
public class PersistenceReaction implements Reaction
{

    public SuperTuple react(SuperTuple t, Operation op, AgentID id)
    {

        //adquirindo acesso ao TRexAccessor
        TRexAccessor accessor = SingletonTRexAccessor.getInstance();
        //a tupla que origina esta reação é sempre uma TRexMessage
        TRexMessage msg = ( TRexMessage ) t;
        //recupera o destinatário da mensagem
        //(agente que deve ser acordado)
        AgentID receiver = msg.getReceiver();
        //ativa (acorda) o agente persistente
        accessor.activatePersistentAgent( receiver );
        return t;
    }
}
```

Fragmento de Código 8 – Reação associada à meta-tupla de ativação de agentes persistentes

```
public class EconomyMetaTuple extends MetaTuple{

    /** Creates new EconomyMetaTuple */
    public EconomyMetaTuple( String item , int maxQuantity )
    {
        /* Associa a reação “EconomyReaction” à operação de retirada (TAKE)
        de uma proposta (representada pela ProposalTuple) efetuada por
        qualquer agente. Esta operação é de fato a tentativa de compra de um
        item*/
         super( new EconomyReaction( maxQuantity ) ,
                Operation.TAKE ,
                AgentID.class ,
                new ProposalTuple( item ) );
    }

}
```

Fragmento de Código 9 - Meta-Tupla responsável pela implementação da estratégia de racionamento de itens

```
public class EconomyReaction implements Reaction
{
    private int maxQty;
    private Hashtable qtyPerAgentHashtable;

    public EconomyReaction( int maxQuantity )
    {
        maxQty = maxQuantity;
    }

    public SuperTuple react( SuperTuple t , Operation op , AgentID id )
    {
        SuperTuple retVal = null;
        //verifica se o agente possui a permissão para comprar o item
        boolean buyPermission = checkBuyPermission( id );
        //caso o agente não possua a permissão a retirada da proposta de
        //venda é impedida, sendo retornado para ele o valor "null"
        if ( buyPermission == false )
        {
            TRexAccessor acc = SingletonTRexAccessor.getInstance();
            acc.getBaseSpace().write( t );
            retVal = null;
        }
        else
        //se o agente possui a permissão de compra ele recebe a proposta
        //e a quantidade comprada do item é armazenada na Hashtable
        {
            retVal = t;
            Integer qty = qtyPerAgentHashtable.get( id );
            //caso o agente ainda não tenha comprado o item a quantidade
            //inserida na Hashtable é igual a 1
            if ( qty == null )
            {
                qtyPerAgentHashtable.put( id , new Integer( 1 ) );
            }
            else
            //caso o agente já tenha comprado o item, a quantidade
            //comprada é incrementada e atualizada na Hashtable
            {
                qtyPerAgentHashtable.put( id , qty.intValue()++ );
            }
            //a ordem de compra relativa ao item comprado
            //também é enviada para o vendedor
            sendPurchaseOrder( t );
        }
        return retVal;
    }
    private boolean checkBuyPermission( AgentID id )
    {
        /*verifica se um agente possui a permissão para comprar o item em
          questão (definido pela meta-tupla). A verificação é baseada na
          quantidade do item já comprada pelo agente. Caso esta
          ultrapasse a quantidade máxima permitida (maxQty) a permissão é
          negada*/
        boolean buyPermission = false;
        Integer qty = qtyPerAgentHashtable.get( id );
        if ( qty != null )
        {
            if ( qty.intValue() >= maxQty )
            {
                buyPermission = false;
            }
            else
            {
                buyPermission = true;
            }
        }
        else
        {
            buyPermission = true;
        }
        return buyPermission;
    }
```

Fragmento de Código 10 – Reação responsável pelo racionamento de itens. É também a responsável pelo envio das ordens de compra para os fornecedores