6 - Caso de Uso – *Supply Chain Managemen*t

Neste capítulo apresenta-se o estudo de caso para uma cadeia de suprimentos. Os agentes foram elaborados utilizando a tecnologia JADE, comentada na seção 5.3 do capítulo 5. A escolha de uma cadeia de suprimentos baseou-se no fato de que uma cadeia de suprimentos é extremamente dinâmica e envolve um grande fluxo de informação, produtos e recursos financeiros entre diferentes agentes. Um dos pontos de maior destaque atualmente é como fazer com que os participantes da cadeia de suprimentos obedeçam a regras ou contratos previamente estabelecidos.

6.1 Cadeia de Suprimentos

No cenário de uma cadeia de suprimentos, em inglês, *Supply Chain Management (SCM)*(Sadeh et al, 2003), entidades autônomas cooperam para a obtenção, montagem e distribuição de uma ou mais famílias de produtos. Agentes se tornam sócios em uma transação, em uma tarefa ou em uma missão a fim de alcançar o objetivo global da organização a que pertencem e satisfazerem a seus próprios objetivos. O esquema da figura 38 ilustra a cadeia de suprimentos sob a perspectiva de serviços requeridos e fornecidos:

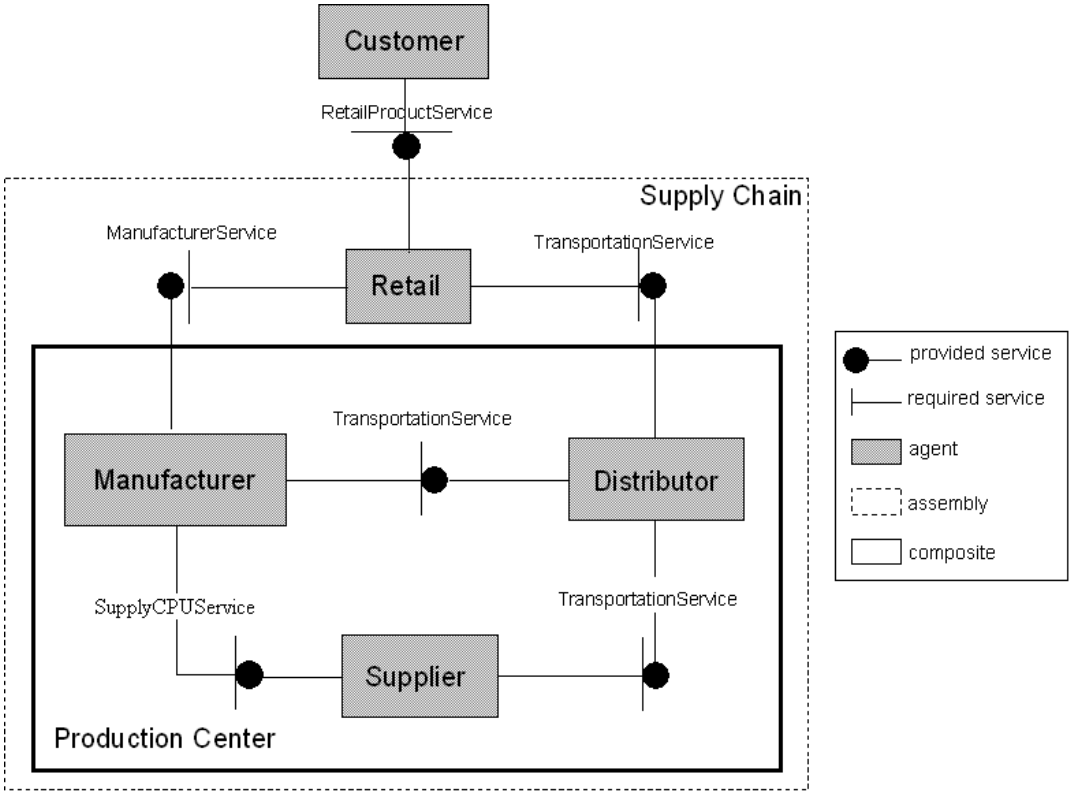

Figura 38: Esquematização de uma cadeia de suprimentos com foco em serviços [Moisés, 2007].

O exemplo acima considera uma cadeia de suprimentos formada por agentes clientes, vendedores, fabricantes, distribuidores e fornecedores. Para facilitar a demonstração, foi adotada a venda de computadores como foco principal da cadeia de suprimentos. Um cliente solicita ao vendedor a venda de um computador. O vendedor, por sua vez, necessita ter computador em estoque para atender a requisição do cliente. O fabricante atende as demandas provenientes dos vendedores e compra peças de diversos fornecedores para a construção dos computadores. Para efeito de simulação da montagem, foi adotado que um computador é composto por uma cpu. O distribuidor provê serviços de entrega de mercadorias tanto para agentes fabricantes, fornecedores ou vendedores.

6.1.1 Ponto Flexível – Protocolo

Um dos pontos flexíveis do *framework* é o protocolo de interação que cada agente poderá assumir. O mecanismo de gerenciamento de contratos utiliza o protocolo de interação para atualizar a máquina de estados dos agentes. O

protocolo define a ordem e mensagens válidas durante uma negociação entre agentes.

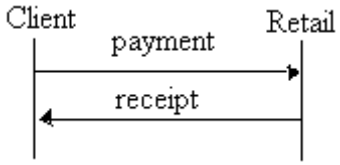

Figura 39: Interação Client – Retail.

O agente mediador (no modelo: *MediatorAgent*) intercepta as mensagem trocadas entre os agentes *client* e *retail* e atualiza o status do serviço prestado pelo vendedor ao cliente de acordo com o protocolo de comunicação do vendedor:

```
<Protocol>
    <States>
      <State id="p1" type="initial"/>
      <State id="p2" type="execution"/>
      <State id="p3" type="sucess"/>
    </States>
    <Transitions>
      <Transition id="payingTransition"
         from="p1" to="p2" message-ref="payment"/>
      <Transition id="paymentConcludedTrans"
         from="p2" to="p3" message-ref="receipt"/>
    </Transitions>
</Protocol>
```

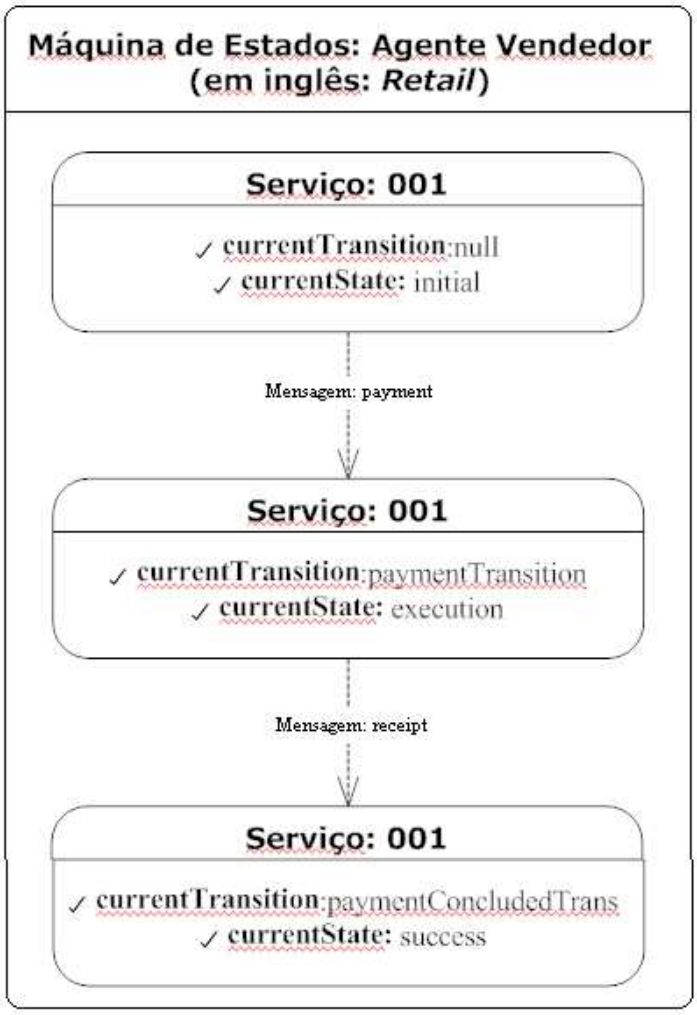

Figura 40: Máquina de Estados e Protocolo do Agente Vendedor (*Retail*).

O estado *p1* representa o estado inicial de uma conversação do agente *Retail*. A transação *payingTransition* é ativada quando o agente *Retail*,

inicialmente no estado *p1*, troca uma mensagem com a referencia *payment. Retail* assume então o estado *p2*, do tipo execução. Por fim, ao trocar uma mensagem com a referencia *receipt* , a transação *paymentConcludedTrans* é ativada e o estado corrente passa para *p3*, do tipo *success*.

6.1.2 Ponto Flexível – Contratos

Agentes poderão assumir diferentes tipos de contratos, de acordo com seu papel na organização e suas necessidades individuais.

6.1.2.1 Contrato de Serviços

Para exemplo de um contrato de serviço descrevendo um serviço de entrega de CPU, a pré-condição necessária para a execução do serviço é o agente ter cpu no estoque, enquanto que a pós-condição é o agente ter recebido o pagamento pelo produto fornecido:

```
service SupplyCPUService {
 property State cpuAvailableState ;
 property State paymentConcludedState;

 pre cpuAvailableState fail "Unable to delivery a cpu" ;
 post paymentConcludedState fail "CPU should be paid " ;
}
```

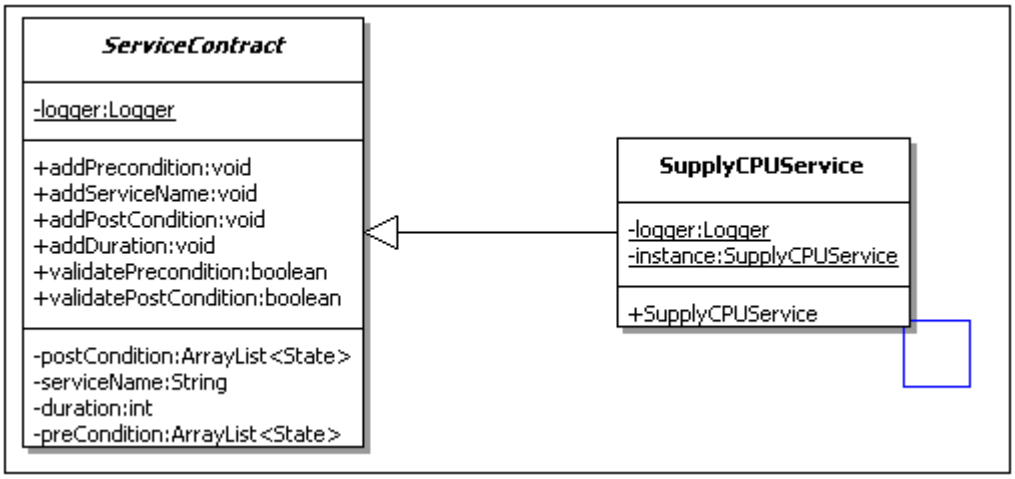

Figura 41: Contrato de serviço para o fornecimento de CPU.

```
public class  SupplyCPUService extends ServiceContract{

    private static SupplyCPUService instance;

    public SupplyCPUService (){

        ArrayList<State> postcondition = new ArrayList<State>();
        ArrayList<State> precondition = new ArrayList<State>();

        precondition.add(new SupplyCPUAvailableState());
        postcondition.add(new PaymentConcludedState());

        addPrecondition(precondition);
        addPostCondition(postcondition);
        addDuration(1000);
        addService("supply_cpu-service");
    }
                ...
}
```

Figura 42: Código do contrato de serviço para o fornecimento de cpu.

6.1.2.2 Contrato de Componente

O contrato de componente pode ser visto como uma individualização dos agentes, uma vez que ele define os serviços oferecidos pelo agente dentro da organização. A seguir é demonstrado um contrato de componente para um agente que assume apenas o serviço de fornecimento de entregas de CPU:

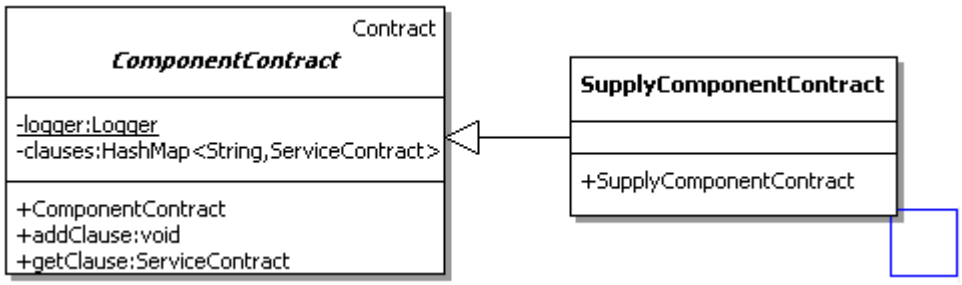

Figura 43: Contrato de componente de um agente fornecedor.

```
public class SupplyComponentContract extends ComponentContract{
    /**
     * Retorna a instancia do singleton.
     * @return instancia do singleton.
     */
    private static SupplyComponentContract instance;

    public synchronized static SupplyComponentContract getInstance(){
        if ( instance == null) {
            instance = new SupplyComponentContract();
        }
        return instance;
    }

    public SupplyComponentContract (){
        addClause (new SupplyCPUService());
    }
}
```

Figura 44: Código de um contrato de componente de um agente fornecedor.

6.1.2.3 Contrato de *Assembly*

O contrato de *assembly* é define parcerias entre agentes ao especificar quem são os agentes requisitantes e os provedores de serviços. A seguir está representado um contrato de *assembly* entre um agente cliente e um agente montador de uma cadeia de suprimentos:

```
assembly ClientAssemblyContract  {
 use  Manufacturer as montador1;
 use  Client as cliente1;

 bind service MountService
                    client cliente1
                           provider montador1;
}
```

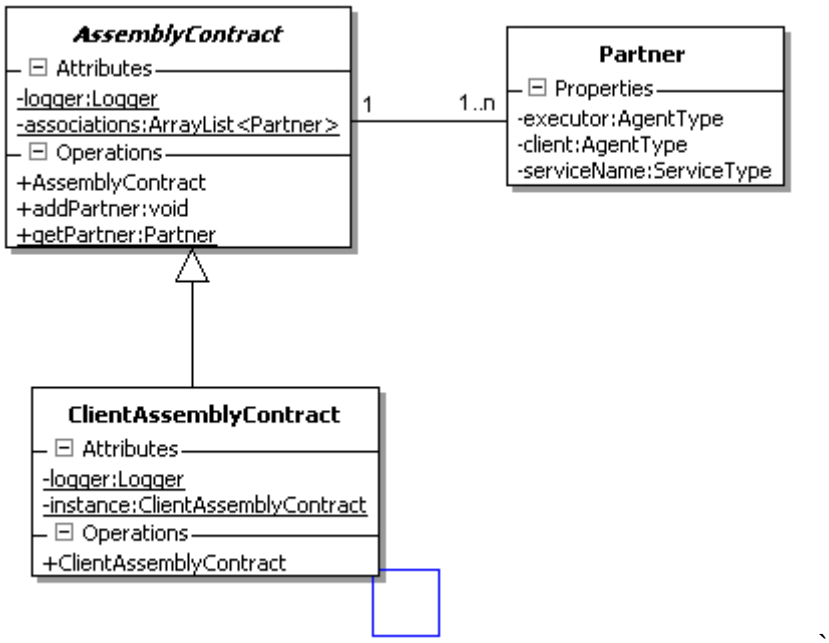

Figura 45: Contrato de assembly de um agente cliente e um agente montador (manufacturer).

A classe *Partner* especifica a ligação entre agentes parceiros na execução de um serviço, através dos atributos *executor, client* e *serviceName*. A classe abstrata *AssemblyContract* representa o contrato de ligação, contendo como atributo uma lista de parceiros (partners).

```
public class ClientAssemblyContract extends AssemblyContract {

    public ClientAssemblyContract (){
        Partner partner = new Partner();
        AgentType des1 = AgentManager.getInstance().getAgentType("cliente1");
        AgentType des2 = AgentManager.getInstance().getAgentType("montador1");

        ServiceTypeDescriptor ser1 = new ServiceTypeDescriptor("mount-service",
                  "br.pucrio.framework.contract.servicecontract.MountService");

        partner.setClient(des1);
        partner.setExecutor(des2);
        partner.setServiceName(ser1);
        addPartner(partner);
    }
          ...
}
```

Figura 46: Código de um contrato de *assembly* entre um agente cliente e um montador.

6.1.2.4 Contrato de Composição

Para a cadeia de suprimentos, foi criada a composição *ProdutionCenter*, composta pelos agentes montador, fornecedor e distribuidor, conforme esquematizado na figura 38. A seguir seu contrato de composição:

```
composite ProductionCenter {

 use  Manufacturer as montador1;
 use  Supplier as fornecedor1;
 use  Distributor as distribuidor1;

 bind service SupplyCPUService
                  client montador1 provider fornecedor1;

 bind service TransportationProductService
                  client montador1
                                provider distribuidor1;

 bind service TransportationProductService
                  client fornecedor1
                                provider distribuidor1;

 export MountService by montador1;
 export TransportationService by distribuidor1;
}
```

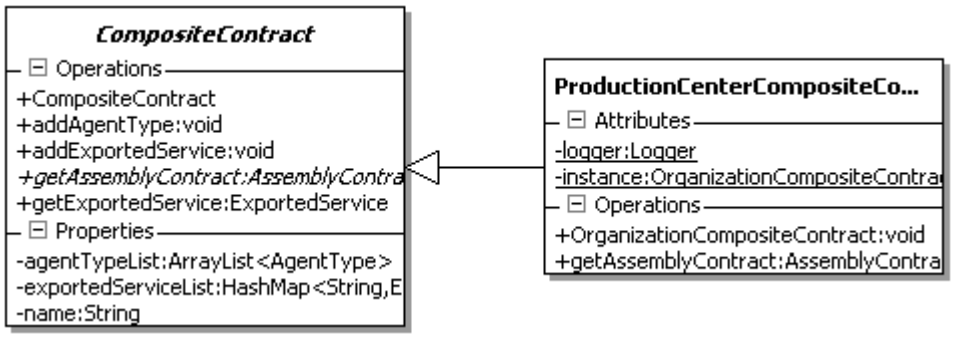

Figura 47: Exemplo de um contrato de composição.

Na composição *ProductionCenter*, o serviço exportado é o serviço de montagem de computadores, executado através do agente *montador1*. Três contratos de *assembly* foram definidos: um entre o agente *montador1* e o agente *fornecedor1* sobre o serviço *SupplyCPUService*, um entre o agente *montador1* e o agente *distribuidor1* sobre o serviço *TransportationProductService*, e um entre

o agente *fornecedor1* e o agente *distribuidor1* sobre o serviço *TransportationProductService.*

```
public class ProductionCenterCompositeContract extends CompositeContract{

     * Variável usada nas operações de log.
    private static Logger logger = Logger.getLogger(AlocatorServiceBehaviour.class);

    private static ProductionCenterCompositeContract instance;

     * Retorna a instancia do singleton.
    public synchronized static ProductionCenterCompositeContract getInstance(){

    public ProductionCenterCompositeContract(){

        AgentType des1 = AgentManager.getInstance().getAgentType("montador1");
        AgentType des2 = AgentManager.getInstance().getAgentType("fornecedor1");
        AgentType des3 = AgentManager.getInstance().getAgentType("distribuidor1");

        addAgentType(des1);
        addAgentType(des2);
        addAgentType(des3);

        ServiceTypeDescriptor ser1 = new ServiceTypeDescriptor("mount-service",
                "br.pucrio.framework.contract.servicecontract.MountService");
        ExportedService es = new ExportedService (des1,ser1);

        addExportedService(es);

    }

    public AssemblyContract getAssemblyContract() {


}
```

Figura 48: Código de um contrato de composição.

6.1.3 Geração de relatórios

O agente gerenciador de contratos (*Contract Manager Agent*), descrito no capítulo 4, seção 4.3.4.6, é responsável por, periodicamente, verificar se os contratos estão sendo cumpridos conforme sua especificação. Este agente produz um relatório sobre o andamento dos serviços, identificando o agente cliente, o executor, o serviço e a situação do contrato (cumprido ou quebrado).

Os testes para geração do relatório foram realizados através de agentes mutantes. A idéia de se utilizar mutantes surgiu na década de 70 na *Yale University e Georgia Institute of Technology*. O critério de Análise de Mutantes utiliza um conjunto de programas ligeiramente modificados (mutantes) obtidos a partir de determinado programa *P* para avaliar o quanto um conjunto de casos de teste *T* é adequado para o teste de *P*. O objetivo é determinar um conjunto de casos de teste que consiga revelar, através da execução de *P*, as diferenças de comportamento existentes entre *P* e seus mutantes (Demillo, 1987).

A idéia básica da técnica apresentada por DeMillo em (Demillo et al, 1978), conhecida como **hipótese do programador competente** (*competent programme hypothesis*), assume que programadores experientes escrevem programas corretos ou muito próximos do correto. Assumindo a validade desta hipótese, pode-se afirmar que erros são introduzidos nos programas através de pequenos desvios sintáticos que, embora não causem erros sintáticos, alteram a semântica do programa e, conseqüentemente, conduzem o programa a um comportamento incorreto. Para revelar tais erros, a Análise de Mutantes identifica os desvios sintáticos mais comuns e, através da aplicação de pequenas transformações sobre o programa em teste, encoraja o testador a construir casos de testes que mostrem que tais transformações levam a um programa incorreto (Agrawal et al, 1989).

Uma outra hipótese explorada na aplicação do critério Análise de Mutantes é o **efeito de acoplamento** (*coupling effect*) (Demillo et al, 1978), a qual assume que erros complexos estão relacionados a erros simples. Assim sendo, espera-se, e alguns estudos empíricos já confirmaram esta hipótese (Acree et al, 1979, Budd, 1980), que conjuntos de casos de teste capazes de revelar erros simples são também capazes de revelar erros complexos. Nesse sentido, aplica-se uma mutação de cada vez no programa *P* em teste, ou seja, cada mutante contém apenas uma transformação sintática. Um mutante com k transformações sintáticas é referenciado por *k*-mutante (Vicenzi et al, 2006); para efeito de teste do modelo contratual proposto foram utilizados apenas 1-mutantes.

Partindo-se da hipótese do programador competente e do efeito de acoplamento, a princípio, o testador deve fornecer um programa *P* a ser testado e um conjunto de casos de teste *T* cuja adequação deseja-se avaliar. O programa é executado com *T* e se apresentar resultados incorretos então um erro foi encontrado e o teste termina. Caso contrário, o programa ainda pode conter erros que o conjunto *T* não conseguiu revelar. O programa *P* sofre então pequenas alterações, dando origem aos programas P_1 , P_2 . . . P_n denominados mutantes de *P*, diferindo de *P* apenas pela ocorrência de erros simples (Delamaro et al, 1993).

Os contratos de serviços descritos abaixo foram assumidos pelos agentes da cadeia de suprimentos, representada pela figura 38. Inicialmente, foi verificado se os agentes implementados realmente cumpriam os contratos de serviço de acordo com o programado, para posteriormente executar os testes com agentes mutantes:

Agente Solicitante: *Client* / Agente Executor: *Retail*

Contrato de Serviço		
Serviço	Venda_de_Produtos(*RetailProductService*)	
Pré-Condições		
Estado	*AvailableComputerState*	O agente que executará o serviço deverá ter um computador disponível em seu estoque.
Pós-Condições		
Estado	*PaymentConcludedState*	O agente solicitante do serviço deverá efetuar o pagamento do produto.

Agente Solicitante: *Retail* / Agente Executor: *Manufacturer*

Contrato de Serviço		
Serviço	Montagem_de_Computador (*ManufacturerService*)	
Pré-Condições		
Estado	*SupplyAvailableCPUState*	O agente que executará o serviço deverá ter CPU disponível em seu estoque.
Pós-Condições		
Estado	*PaymentConcludedState*	O agente solicitante do serviço deverá efetuar o pagamento do produto.
Estado	*SendProductToDeliveryState*	O agente executor deverá ter enviado o produto para a entrega.

Agente Solicitante: *Retail* , *Manufacturer, Supplier*

Agente Executor: *Distributor*

Contrato de Serviço	
Serviço	Transporte_de_Produtos(*TransportationService*)
Pré-Condições	

Nenhum estado como pré-condição é requerido		
Pós-Condições		
Estado	*ProductDeliveredState*	O agente executante do serviço deverá ter entregado o produto para o destinatário.

Agente Solicitante: *Manufacturer* / Agente Executor: *Supplier*

Contrato de Serviço		
Serviço	Fornecimento_de_Materiais (*SupplyCPUService*)	
Pré-Condições		
Estado	*SupplyAvailableCPUState*	O agente que executará o serviço deverá ter CPU disponível em seu estoque.
Pós-Condições		
Estado	*PaymentConcludedState*	O agente solicitante do serviço deverá efetuar o pagamento do produto.
Estado	*SendProductToDeliveryState*	O agente executor deverá ter enviado o produto para a entrega.

O exemplo a seguir é referente à execução de um serviço de fornecimento de cpu (*SupplyCPUService*) fornecido pelo agente supplier, pertencente a composição ProductionCenter, ao agente manufacturer. Ao analisar o desenvolvimento do contrato de componente do agente *supplier* sobre o serviço *SupplierCPUService*, o mecanismo de validação de contratos conclui que o contrato foi executado de acordo com o esperado.

```
[contractManager] (CheckContractBehaviour.java:85) - --------------------
[contractManager] (CheckContractBehaviour.java:86) - Agente = supplier
[contractManager] (CheckContractBehaviour.java:88) - Composição = ProductionCenter
[contractManager] (CheckContractBehaviour.java:90) - Servico ID= 1387128098
[contractManager] (CheckContractBehaviour.java:91) - Servico Para = manufacturer
[contractManager] (CheckContractBehaviour.java:92) - Servico de Nome = supply_cpu-service
[contractManager] (CheckContractBehaviour.java:93) - Verificando post-condition
[contractManager] (CheckContractBehaviour.java:98) - Agente cumpriu o contrato
[contractManager] (CheckContractBehaviour.java:85) - --------------------
```

Figura 49: Resultado da execução do serviço supply_cpu-service fornecido pelo agente supplier ao agente manufacturer.

Com o objetivo de verificar se o mecanismo de gerenciamento de contratos detecta agentes que não cumpram os contratos assumidos, foram criados

agentes mutantes baseando-se nos agentes da cadeia de suprimentos. O diagrama de pacotes abaixo mostra os agentes originais da cadeia de suprimentos com seus respectivos comportamentos, implementados através da plataforma Jade, descrita no capítulo 6, seção 6.3:

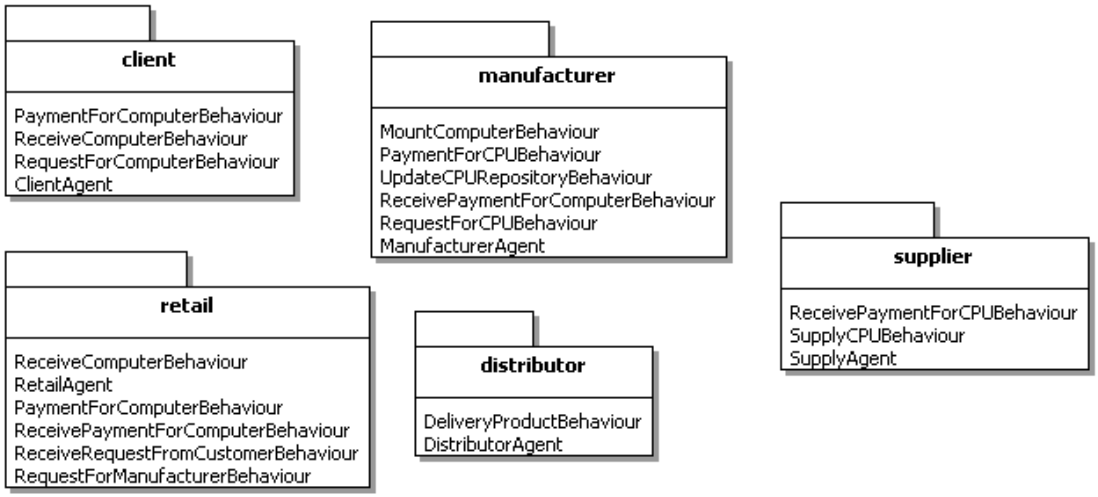

Figura 50: Diagrama de pacotes mostrando os agentes e seus respectivos comportamentos.

Os operadores de mutação (entende-se por operadores de mutação as regras que definem as alterações sobre o programa original *P* (Vicenzi et al, 2006)) realizados sobre os agentes originais da cadeia de suprimentos para a implementação dos mutantes, bem como os resultados obtidos com os testes estão descritos nas tabelas abaixo:

Mutante	
Nome	*Client_1*
Agente de Origem	*Client*
Operador de Mutação	
O comportamento *PaymentForComputerBehavior* do agente *Client* foi alterado para que o cliente não efetue o pagamento ao receber um computador.	
Resultado	
O serviço de venda de produto (*RetailProductService*) foi quebrado, apontado como inválido o estado *PaymentConcludedState* da pós-condição.	

Mutante	
Nome	*Retail_1*
Agente de Origem	*Retail*
Operador de Mutação	
O comportamento *PaymentForComputerBehavior* do agente *Retail* foi alterado para que o vendedor não efetue o pagamento ao receber um computador do agente *Manufacturer*.	
Resultado	
O serviço de montagem de computador (*ManufacturerService)* foi quebrado, apontando como inválido o estado *PaymentConcludedState* da pós condição.	

Mutante	
Nome	*Manufacturer_1*
Agente de Origem	*Manufacturer*
Operador de Mutação	
O comportamento *PaymentForCPUBehavior* do agente *Manufacturer* foi alterado para que o montador não efetue o pagamento ao receber uma cup.	
Resultado	
O serviço de montagem de computador (*SupplyCPUService*) foi quebrado, apontado como inválido o estado *PaymentConcludedState* da pós-condição.	

Mutante	
Nome	*Manufacturer_2*
Agente de Origem	*Manufacturer*
Operador de Mutação	
O comportamento *MountComputerBehavior* do agente *Manufacturer* foi alterado para que o montador não efetue a montagem de um computador, ao receber a solicitação de um serviço de montagem pelo agente *Retail*.	
Resultado	
O serviço de montagem de computador (*ManufacturerService)* foi quebrado, apontando como inválido o estado *SendProductToDelivery* da pós condição.	

Mutante	
Nome	*Distributor_1*
Agente de Origem	*Distributor*
Operador de Mutação	

O comportamento *DeliveryProductBehavior* do agente *Distributor* foi alterado para que o distribuidor não realize o transporte do produto para o destinatário.
Resultado
O serviço de transporte de produtos (*TransportationService)* foi quebrado, apontando como inválido o estado *ProductDeliveredState* da pós condição.

Mutante	
Nome	*Supplier_1*
Agente de Origem	*Supplier*
Operador de Mutação	
O comportamento *SupplyCPUBehavior* do agente *Supplier* foi alterado para que o fornecedor não forneça CPU para o agente montador.	
Resultado	
O serviço de fornecimento de CPU (*SupplyCPUService)* foi quebrado, apontando como inválido o estado *SendProductToDeliveryState* da pós condição.	

A seguir está demonstrado um exemplo de saída de relatório para um caso de quebra do contrato. Neste exemplo, o mutante *Supplier_1* quebrou o contrato ao não fornecer a CPU para o agente montador:

```
[contractManager] (CheckContractBehaviour.java:85) - --------------------
[contractManager] (CheckContractBehaviour.java:86) - Agente = Supplier_1
[contractManager] (CheckContractBehaviour.java:88) - Composição = ProductionCenter
[contractManager] (CheckContractBehaviour.java:90) - Servico ID= 1593828667
[contractManager] (CheckContractBehaviour.java:91) - Servico Para = Manufacturer
[contractManager] (CheckContractBehaviour.java:92) - Servico de Nome = SupplierCPUservice
[contractManager] (CheckContractBehaviour.java:93) - Verificando post-condition
[contractManager] (CheckContractBehaviour.java:98) - O contrato foi quebrado
[contractManager] (CheckContractBehaviour.java:99) - --> SendProductToDeliveryState inválido.
[contractManager] (CheckContractBehaviour.java:85) - --------------------
```

Figura 51: Resultado de um contrato não cumprido.