6
Estudo de caso: Aeroporto

Atualmente, os grandes aeroportos fazem mais do que simplesmente servir de lugar para aterrissagem e decolagem de aviões. Shopping centers com centenas de lojas, cinemas, hotéis, centros empresariais, e até mesmo centros gastronômicos são algumas das atrações dos aeroportos atuais. O grande número de usuários em potencial e a variedade de serviços oferecidos em um mesmo local fazem dos aeroportos um bom domínio para o desenvolvimento de aplicações como por exemplo, compra de produtos ou mesmo a compra de passagens aéreas.

O cenário adotado neste estudo de caso simula a situação onde uma pessoa chega ao aeroporto com um dispositivo móvel. O aeroporto é dotado de uma rede de sistemas embarcados que fornece serviços de "paquera", cinema, compras e localização. Ao chegar no aeroporto, o usuário requisita quais são os serviços disponíveis naquele momento. O anúncio da lista de serviços disponíveis é feito por um servidor do próprio aeroporto, denominado *announcer*. Uma vez que um dos serviços é escolhido pelo usuário, o *announcer* informa ao usuário quais são as opções para o serviço selecionado. Por exemplo, se o serviço selecionado for de compras, então é exibida uma lista de produtos disponíveis para a compra. Neste estudo de caso, o escopo é limitado ao serviço de compra, uma vez que o desenvolvimento dos outros serviços seria realizado de forma análoga e não traria benefícios à pesquisa.

Ao selecionar um dos produtos da lista de compras, o usuário recebe uma lista de todas as lojas que possuem aquele produto. A partir deste momento, o usuário escolhe uma das lojas e inicia o processo de negociação que, se for bem sucedido, será seguido de um processo de pagamento. Para o processo de pagamento, um outro elemento do aeroporto aparece: o banco. O banco também provê serviços de pagamentos de mercadorias via sistema embarcado. Desta forma, o usuário paga o valor negociado ao banco e apresenta o recibo eletrônico para receber a mercadoria.

O sistema do aeroporto deve inspirar um bom nível de confiança no sentido em que os clientes devem ser protegidos contra comportamentos maliciosos das

empresas situadas no aeroporto e as empresas também devem ser protegidas contra usuários maliciosos. Uma vez que ambas as partes sabem que os elementos do sistema estão interagindo mediante regras bem definidas e públicas, a confiança no sistema tende a crescer. Embora toda a motivação para a realização deste estudo de caso seja baseada em sistemas embarcados, o sistema foi simulado sem a utilização desta tecnologia, uma vez que o foco do estudo de caso para esta dissertação é a experimentação dos conceitos e do software proposto.

Na Seção 6.1, apresenta-se como os elementos do modelo conceitual representados na linguagem XMLaw foram utilizados para especificar as interações deste sistema.

6.1
Utilizando os conceitos do modelo conceitual

O processo de interação entre os agentes do sistema foi modularizado e modelado utilizando o conceito de cenas. Quatro cenas foram identificadas: *chegada*, *seleção*, *negociação* e *pagamento*. Na primeira cena, o usuário descobre quais os serviços estão disponíveis. Na segunda, seleciona dentre um dos serviços, que neste estudo de caso será o serviço de compra, recebe uma lista de todos os produtos que estão disponíveis para a venda, escolhe um dos produtos e recebe uma lista de todos os vendedores daquele produto. Na etapa de *negociação*, o usuário escolhe um dos vendedores e negocia um preço para o produto. Finalmente, na última etapa o valor negociado do produto é pago ao banco. Este processo é ilustrado na Figura 6.1.

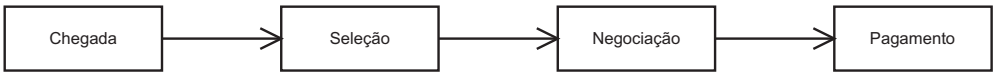

Figura 6.1: Cenas Modularizando a Interação

A seguir, o funcionamento de cada cena e suas respectivas restrições de acesso são detalhados e representados graficamente através dos símbolos da Figura 6.2 e textualmente através do XMLaw.

Cena: *chegada*

Cada cena especifica o **protocolo** de interação que os agentes devem seguir e, consequentemente, quais são as mensagens válidas. A Figura 6.3 mostra as leis da cena *chegada*. Esta cena possui o protocolo ilustrado na Figura 6.4 e é especificada usando-se o XMLaw mostrado na Tabela 6.1. Nesta cena, considera-se arbitrariamente que todo o processo não deve durar mais do que 10 segundos

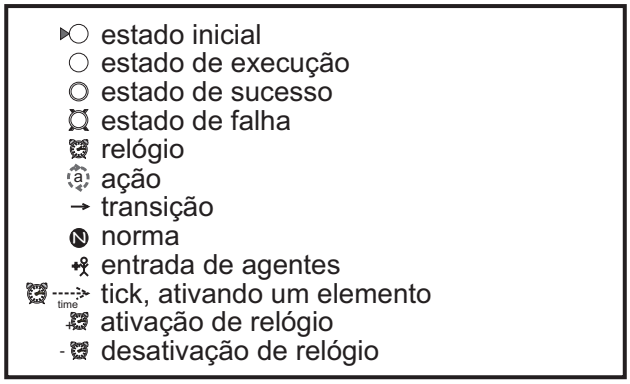

Figura 6.2: Símbolos Utilizados para Representar as Cenas

(**time-to-live**). A cena pode ser instanciada por qualquer agente desempenhando qualquer papel (tag *Creator*). Porém, apenas agentes desempenhando papéis de *customer* e *announcer* podem **participar** das interações (tag *Participant*). Além disso, eles só podem participar se o estado de execução do protocolo estiver no estado *s0* para a entrado do *customer* e no estado *s1* para a entrada do *announcer*. As interações desta cena contêm apenas duas **mensagens**, sendo uma do agente *customer* para o agente *announcer* avisando que chegou no aeroporto (mensagem *m1*) e a resposta do agente *announcer* através da lista de serviços disponíveis no aeroporto (mensagem *m2*). Um **relógio** é ativado toda vez que um agente envia uma mensagem requisitando os serviços. A finalidade deste relógio é manter uma bom tempo de resposta no sistema, pois uma vez que o relógio contar 5 segundos, uma **ação** de recuperação automática do sistema (*announcer-is-down*) é ativada.

Cena: chegada
1. Qualquer agente pode criar a cena.
2. Agentes desempenhando o papel de “customer” só podem entrar na cena se o protocolo estiver no estado inicial, ou seja, nenhuma conversação ocorreu antes.
3. Agentes desempenhando o papel de “announcer” só podem entrar na cena após um agente “customer” ter iniciado a conversação.
4. Para iniciar a conversação, o agente “customer” envia uma mensagem “se apresentando” ao agente “announcer”, que responde com uma lista de serviços disponíveis.
5. Após um agente “customer” iniciar a conversação, o agente “announcer” possui 5 segundos para enviar uma resposta. Caso nenhuma resposta tenha sido enviada, pode significar, por exemplo, que o agente “announcer” não está funcionando corretamente, ou existem problemas de comunicação. Desta forma, deve ser executada uma ação de recuperação para verificar o motivo.

Figura 6.3: Leis da Cena *chegada*

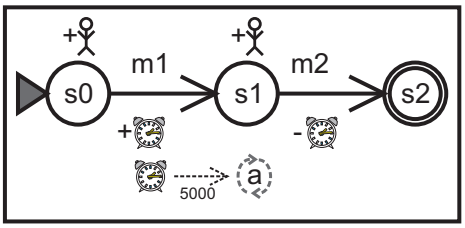

Figura 6.4: Cena: *chegada*

```
<Scene id="anouncement" time-to-live="10000"> <!-- 10 sec. -->
    <Creators>
        <Creator agent="any" role="any"/>
    </Creators>
    <Entrance>
        <Participant agent="any" role="customer">
            <StatesRef>
                <StateRef ref="s0"/>
            </StatesRef>
        </Participant>
        <Participant agent="any" role="announcer">
            <StatesRef>
                <StateRef ref="s1"/>
            </StatesRef>
        </Participant>
    </Entrance>
    <Messages>
        <Message id="m1" template="message(request,sender(_,customer),receiver(_
            ,announcer),content(hello))."/>
        <Message id="m2" template="message(inform,sender(_,announcer),receiver(
            CustomerAgent,customer),content(services([_|_])))."/>
    </Messages>
    <Protocol>
        <States>
            <State id="s0" type="initial" label="Initial␣state"/>
            <State id="s1" type="execution" label="Message␣sent"/>
            <State id="s2" type="success" label="Response␣answered"/>
        </States>
        <Transitions>
            <Transition id="t1" from="s0" to="s1" message-ref="m1"/>
            <Transition id="t2" from="s1" to="s2" message-ref="m2"/>
        </Transitions>
    </Protocol>
    <Clocks>
        <Clock id="time-for-answering-hello" type="regular" tick-period="5000">
            <!-- 5 sec. -->
            <Activations>
                <Element ref="t1" event-type="transition_activation"/>
            </Activations>
            <Deactivations>
                <Element ref="t2" event-type="transition_activation"/>
            </Deactivations>
        </Clock>
    </Clocks>
    <Actions>
        <Action id="announcer-is-down" class="br.pucrio.inf.les.law.app.airport.
            repairactions.HealAnnouncerAction">
```

```
            <Element ref="time-for-answering-hello" event-type="clock_tick"/>
        </Action>
    </Actions>
</Scene>
```

Tabela 6.1: Implementação da Cena *chegada* Utilizando o XMLaw

Cena: *seleção*

Esta cena tem duração de 5 minutos, que é o tempo dado ao usuário (agente *customer*) para escolher entre os serviços e entre os produtos. A primeira mensagem é enviada pelo *customer* ao *announcer* informando qual o serviço foi escolhido. O envio desta mensagem causa o disparo da transição *t3* do XMLaw. A resposta do *announcer* contém a lista de produtos disponíveis para o serviço requisitado (mensagem *m4*), lembrando que o único serviço disponível neste estudo de caso é o serviço de compras. O *customer* escolhe um dos produtos e requisita a lista de vendedores do produto (mensagem *m5*). Finalmente, a lista de vendedores é informada (mensagem *m6*) e o protocolo alcança o estado de sucesso *s7* e o protocolo e, conseqüentemente, a cena são encerrados.

Cena: seleção
1. Qualquer agente pode criar a cena.
2. Um agente "customer" é responsável pode iniciar a conversação e, portanto, somente pode entrar na cena se o protocolo estiver no estado inicial.
3. Um agente "announcer" só pode entrar na cena em resposta a conversação iniciada por um agente "customer".

Figura 6.5: Leis da cena *seleção*

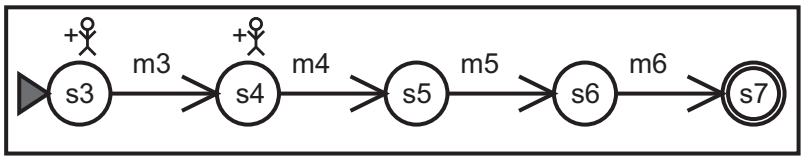

Figura 6.6: Cena: *seleção*

```
<Scene id="selection" time-to-live="300000"> <!-- 5 min. -->
    <Creators>
        <Creator agent="any" role="any"/>
    </Creators>
    <Entrance>
        <Participant agent="any" role="customer">
            <StatesRef>
                <StateRef ref="s3"/>
            </States>
        </Participant>
        <Participant agent="any" role="announcer">
            <StatesRef>
                <StateRef ref="s4"/>
            </StatesRef>
        </Participant>
    </Entrance>
    <Messages>
        <Message id="m3" template="message(request,sender(_,customer),receiver(_
            ,announcer),content(option(Service))."/>
        <Message id="m4" template="message(inform,sender(_,announcer),receiver(_
            ,customer),content(products(ListOfProducts))."/>
        <Message id="m5" template="message(request,sender(_,customer),receiver(_
            ,announcer),content(sellers-of(Product))."/>
        <Message id="m6" template="message(inform,sender(_,announcer),receiver(
            Customer,customer),content(sellers([_|_]))."/>
    </Messages>
    <Protocol>
        <States>
            <State id="s3" type="initial" label="Ready to ask for options"/>
            <State id="s4" type="execution" label="List of products requested"/>
            <State id="s5" type="execution" label="List of products informed"/>
            <State id="s6" type="execution" label="List of sellers requested"/>
            <State id="s7" type="success" label="List of sellers informed"/>
        </States>
        <Transitions>
            <Transition id="t3" from="s3" to="s4" message-ref="m3"/>
            <Transition id="t4" from="s4" to="s5" message-ref="m4"/>
            <Transition id="t5" from="s5" to="s6" message-ref="m5"/>
            <Transition id="t6" from="s6" to="s7" message-ref="m6"/>
        </Transitions>
    </Protocol>
</Scene>
```

Tabela 6.2: Implementação da Cena *seleção* Utilizando o XMLaw

Cena: *negociação*

A protocolo de interação utilizado para a cena de negociação é baseado no FIPA-CONTRACT-NET [72]. A primeira mensagem é enviada pelo *customer* para o vendedor (*seller*) requisitando uma proposta para o produto selecionado. Além disso, o *customer* também faz algumas exigências sobre o preço máximo que ele pagará pelo produto e qual a marca do produto desejada. O vendedor só pode enviar propostas com valores menores ou iguais ao preço máximo

informado pelo *customer*. Esta restrição de preço é implementada nas leis através da classe *EnforceValue*, listada na tabela 6.4, que é chamada pelas restrições em *t7* e *t9*. Em *t7* o valor do preço máximo é capturado e em *t9* o valor informado é verificado. O vendedor pode responder com uma proposta de compra (mensagem *m8* e transição *s9* para *s10*) ou pode negar enviando a mensagem *m11*. Caso o vendedor negue, então o protocolo termina no estado *s13*, que é um estado de falha. Ao receber a proposta de compra, o agente *customer* possui 20 segundos para decidir se vai aceitar ou não a proposta. Caso os 20 segundos expirem, o *seller* ganha permissão para cancelar a proposta feita. Esta regra é importante pois protege o vendedor caso outro cliente esteja interessado no mesmo produto, mas o vendedor está impedido de vender o produto por ter iniciado a negociação com um cliente. O estado *s15* representa que o *seller* cancelou a proposta feita após pelo menos 20 segundos de indecisão do *customer*, e o estado *s14* significa que o *customer* não aceitou a proposta enviada pelo *seller*. Se o *customer* aceitar a proposta, o protocolo passa ao estado *s11* e o seller informa onde o pagamento deve ser feito (mensagem *m10*).

Cena: negociação
1. Qualquer agente pode criar a cena.
2. Um agente "customer" é responsável pode iniciar a conversação e, portanto, somente pode entrar na cena se o protocolo estiver no estado inicial.
3. Um agente "seller" só pode entrar na cena em resposta a conversação iniciada por um agente "customer".
4. Agentes "customer" possuem 20 segundos para decidir se aceitam ou não uma proposta enviada por um "seller". Após decorridos 20 segundos, será dada permissão ao "seller" para cancelar as negociações com o agente "customer".
5. Agentes que desempenham o papel "seller" precisam enviar propostas com o preço sempre igual ou abaixo do preço sugerido pelo agente "customer".

Figura 6.7: Leis da cena *negociação*

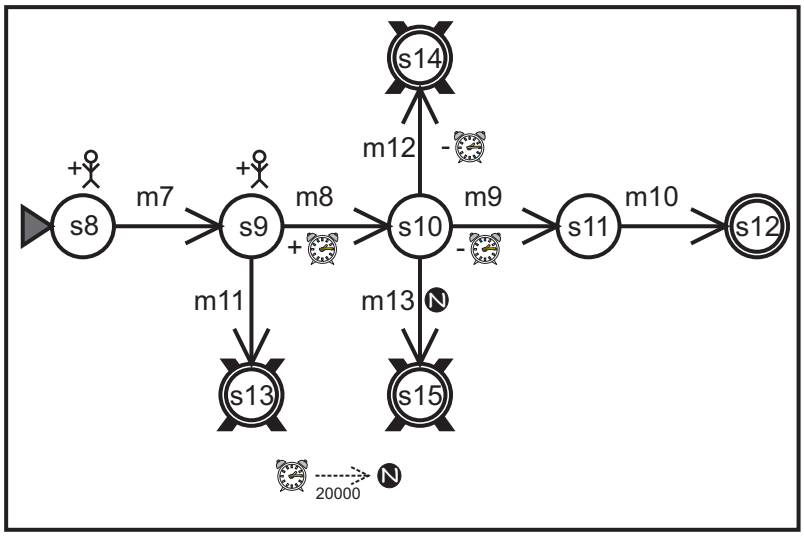

Figura 6.8: Cena: *negociação*

```
<Scene id="negotiation" time-to-live="infinity">
    <Creators>
        <Creator agent="any" role="any"/>
    </Creators>
    <Entrance>
        <Participant agent="any" role="customer">
            <StatesRef>
                <StateRef ref="s8"/>
            </StatesRef>
        </Participant>
        <Participant agent="any" role="seller">
            <StatesRef>
                <StateRef ref="s9"/>
            </StatesRef>
        </Participant>
    </Entrance>
    <Messages>
        <Message id="m7" template="message(cfp,sender(_,customer),receiver(_,
            seller),content(product-details(product(Product),maxprice(Price),
            brand(Brand)))."/>
        <Message id="m8" template="message(propose,sender(SellerAgent,seller),
            receiver(_,customer),content(product(name(Product),price(Price),
            brand(Brand)))."/>
        <Message id="m9" template="message(accept-proposal,sender(_,customer),
            receiver(_,seller),content(_))."/>
        <Message id="m10" template="message(inform,sender(_,seller),receiver(
            CustomerAgent,customer),content(payto(Bank)))."/>

        <Message id="m11" template="message(refuse,sender(_,seller),receiver(_,
            customer),content(Reason))."/>
        <Message id="m12" template="message(reject-proposal,sender(_,customer),
            receiver(_,seller),content(Reason))."/>

        <Message id="m13" template="message(cancel,sender(_,seller),receiver(_,
            customer),content(Reason))."/>
    </Messages>
    <Protocol>
```

```
        <States>
            <State id="s8" type="initial" label="Ready for starting
                negotiations"/>
            <State id="s9" type="execution" label="Call for proposal requested"
                />
            <State id="s10" type="execution" label="Proposal sent"/>
            <State id="s11" type="execution" label="Proposal accepted"/>
            <State id="s12" type="success" label="Bank informed"/>
            <State id="s13" type="failure" label="Refuse sending proposal"/>
            <State id="s14" type="failure" label="Proposal rejected"/>
            <State id="s15" type="failure" label="Too long time to decide"/>
        </States>
        <Transitions>
            <Transition id="t7" from="s8" to="s9" message-ref="m7">
                <Constraints>
                    <Constraint class="br.pucrio.inf.les.law.app.airport.
                        repairactions.EnforceValue">
                        <Semantic>Gets value on price</Semantic>
                    </Constraint>
                </Constraints>
            </Transition>
            <Transition id="t8" from="s9" to="s10" message-ref="m8">
                <Constraints>
                    <Constraint class="br.pucrio.inf.les.law.app.airport.
                        repairactions.EnforceValue">
                        <Semantic>Enforces value on price</Semantic>
                    </Constraint>
                </Constraints>
            </Transition>
            <Transition id="t9" from="s10" to="s11" message-ref="m9"/>
            <Transition id="t10" from="s11" to="s12" message-ref="m10"/>
            <Transition id="t11" from="s9" to="s13" message-ref="m11"/>
            <Transition id="t12" from="s10" to="s14" message-ref="m12"/>
            <Transition id="t13" from="s10" to="s15" message-ref="m13">
                <ActiveNorms>
                    <Norm ref="seller-permission-to-cancel"/>
                </ActiveNorms>
            </Transition>
        </Transitions>
    </Protocol>
    <Clocks>
        <Clock id="time-to-decide" type="regular" tick-period="20000"> <!-- 20
            sec. -->
            <Activations>
                <Element ref="t8" event-type="transition_activation"/>
            </Activations>
            <Deactivations>
                <Element ref="t9" event-type="transition_activation"/>
                <Element ref="t12" event-type="transition_activation"/>
            </Deactivations>
        </Clock>
    </Clocks>
    <Norms>
        <Permission id="seller-permission-to-cancel">
            <Owner>SellerAgent</Owner>
            <Activations>
                <Element ref="time-to-decide" event-type="clock_tick"/>
            </Activations>
        </Permission>
```

```
    </Norms>
</Scene>
```

Tabela 6.3: Implementação da Cena *negociação* Utilizando o XMLaw

```
public class EnforceValue implements IConstraint{

    private double maxPrice;
    private double price;

    public boolean constrain(InfoCarrier info) {

        Message msg = (Message)info.getValue(Constants.MESSAGE_INFO_KEY);
        if ( msg.getPerformative().equals(Message.CALL_FOR_PROPOSAL) ){
            maxPrice = Double.parseDouble((String)info.getValue("Price"));
        }else if ( msg.getPerformative().equals(Message.PROPOSE) ){
            price = Double.parseDouble((String)info.getValue("Price"));
            if (price <= maxPrice){
                return false;
            }else{
                return true;
            }
        }
        return false;
    }
}
```

Tabela 6.4: Implementação da Restrição no Preço Oferecido

Cena: *pagamento*

A criação de uma cena de pagamento só pode ser efetivada por um agente desempenhando o papel de *customer* e que possua a permissão *permission-to-pay*. Esta permissão é obtida pelos agentes que concluíram o cena de negociação com sucesso e é representada como uma norma global, ou seja, uma norma especificada no contexto da organização. A especificação desta norma está ilustrada na Tabela 6.5.

```
    ...
    </Scenes>
    <Norms>
        <Permission id="permission-to-pay">
            <Owner>CustomerAgent</Owner>
            <Activations>
                <Element ref="negotiation" event-type="
                    sucessful_scene_completion"/>
            </Activations>
            <Deactivations>
                <Element ref="payment" event-type="sucessful_scene_completion"/>
            </Deactivations>
        </Permission>
    </Norms>
</LawOrganization>
```

Tabela 6.5: Restrição de Acesso à Cena *pagamento*

Com exceção desta restrição na entrada da cena *pagamento*, esta cena é uma das mais simples e sua especificação é ilustrada na Figura 6.10 e na Tabela 6.6.

Cena: pagamento
1. Só quem pode criar esta cena é um agente "customer" que tenha completado com sucesso a cena de negociação. Ou seja, somente os agentes que realizaram negócios na cena de negociação é que podem iniciar a cena de pagamento.
2. Um agente "customer" somente pode entrar na cena se o protocolo estiver no estado inicial.
3. Um agente "bank" só pode entrar na cena em resposta a conversação iniciada por um agente "customer".

Figura 6.9: Leis da cena *pagamento*

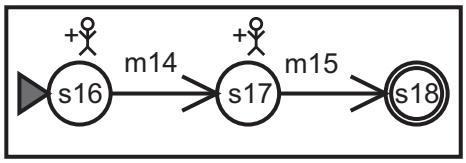

Figura 6.10: Cena: *pagamento*

```
<Scene id="payment" time-to-live="infinity">
    <Creators>
        <Creator agent="any" role="customer">
            <ActiveNorms>
                <Norm ref="permission-to-pay"/>
            </ActiveNorms>
        </Creator>
    </Creators>
    <Entrance>
        <Participant agent="any" role="customer">
            <StatesRef>
                <StateRef ref="s16"/>
            </StatesRef>
        </Participant>
        <Participant agent="any" role="bank">
            <StatesRef>
                <StateRef ref="s17"/>
            </StatesRef>
        </Participant>
    </Entrance>
    <Messages>
        <Message id="m14" template="message(request,sender(_,initiator),receiver
            (_,participant),content(pay(amount(Amount),to(Seller)))."/>
        <Message id="m15" template="message(inform,sender(_,participant),
            receiver(_,initiator),content(receipt(number(Number))))."/>
    </Messages>
    <Protocol>
        <States>
            <State id="s16" type="initial" label="Ready for payment"/>
            <State id="s17" type="execution" label="Payment order emitted"/>
            <State id="s18" type="success" label="Receipt sent"/>
        </States>
        <Transitions>
            <Transition id="t14" from="s16" to="s17" message-ref="m14"/>
            <Transition id="t15" from="s17" to="s18" message-ref="m15"/>
        </Transitions>
    </Protocol>
</Scene>
```

Tabela 6.6: Restrição de Acesso à Cena *pagamento*