7
Estudos de Caso

Este capítulo apresenta as aplicações móveis utilizadas como estudos de caso para a avaliação do mecanismo de tratamento de exceções sensível ao contexto proposto neste trabalho. *Virtual Lines* é uma aplicação desenvolvida no LAC/PUC-Rio que realiza o controle de filas virtuais em parques de diversão. *Health Care* é uma aplicação móvel que oferece suporte ao monitoramento de pacientes com doença cardiovascular sob cuidados médicos.

7.1.
Virtual Lines (VL)

Virtual Lines é uma aplicação móvel desenvolvida para exemplificar o uso da arquitetura MoCA (Sacramento et al., 2004). Esta aplicação realiza o controle de filas virtuais em parques de diversão e tem como principal objetivo impedir que as pessoas esperem muito tempo nas filas, ao mesmo tempo que aproveitam da melhor forma as atrações existentes em um parque. Ao passar próximo a uma atração, um dispositivo móvel pode coletar um *ticket* virtual que corresponde a um lugar na fila. O sistema avisa ao usuário sobre a proximidade de sua vez a fim de que ele possa retornar a tempo de participar da atração. Quando o usuário não retorna a tempo, o sistema emite um alerta avisando que ele perdeu sua vez.

Um dos fatores que motivou a escolha de VL como estudo de caso deste trabalho foi o fato desta aplicação estar disponível como protótipo já no início de nosso trabalho. Embora algumas adaptações tenham sido necessárias, acrescentamos basicamente o funcionamento excepcional da aplicação, utilizando para isso a API de nosso mecanismo de tratamento de exceções sensível ao contexto. A implementação de *Virtual Lines* permitiu-nos ter uma primeira visão sobre os pontos importantes a serem tratados no tratamento de exceções sensível ao contexto de uma aplicação móvel.

A aplicação VL está dividida em três módulos (i) classes de negócio, (ii) gerenciador das filas virtuais e (iii) interface com o cliente. O primeiro módulo contém as classes de negócio, usadas pelos demais módulos. Estão presentes classes para representar atrações, filas, usuários, etc., além de algumas classes para a comunicação entre cliente e servidor. O segundo módulo funciona como um servidor no papel de gerenciador das filas virtuais. Contém uma instância do parque e é responsável por administrar as filas de cada uma das atrações. O terceiro módulo é utilizado diretamente pelos usuários e representa a aplicação cliente. Este módulo indica qual a atração mais próxima de um usuário, permitindo a entrada deste em uma fila virtual; além disso, mantém atualizada a informação sobre a situação dos usuários nas filas.

Na aplicação VL, a comunicação síncrona é utilizada para a comunicação do cliente com o servidor quando, por exemplo, um usuário solicita entrada em uma fila. A comunicação assíncrona é usada quando o servidor precisa enviar alguma notificação ao cliente, por exemplo, quando o gerenciador de filas precisa avisar a um usuário que este deve comparecer a uma atração.

7.1.1. Especificação de Tratadores e Propagação de Exceções

Para ilustrar a especificação de exceções contextuais, foram criadas as exceções `DeviceNotFoundException` e `AttractionOffException`. Cada uma destas exceções possui diferentes tratadores, os quais devem ser executados de acordo com suas próprias condições de contexto. A exceção `DeviceNotFoundException` ocorre quando um dispositivo móvel que deveria estar no parque não é encontrado. A subscrição de contexto excepcional que representa esta exceção contém a expressão `((Online = false) and (DeltaT > 50000))`, que é enviada ao MoCA sempre que um novo dispositivo móvel registra sua entrada no parque. A princípio, esta condição de contexto pode parecer muito simples; porém, para o domínio da aplicação *Virtual Lines*, é fundamental ter conhecimento sobre os usuários que ainda estão fisicamente no parque. Sem a existência desta exceção, mesmo após sua saída, os usuários podem ainda estar ocupando lugares nas filas, o que ocasiona uma espera desnecessária aos outros clientes presentes.

Para o tratamento da exceção `DeviceNotFoundException`, as condições de contexto verificadas pelos tratadores são: (i) o número de vezes que o usuário não compareceu a uma atração, mesmo após receber notificações sobre sua vez nas filas, e (ii) a existência de algum registro de problemas com o dispositivo. O tratador `ExitedUserHandler` verifica se o usuário está "desaparecido" há mais de duas atrações. Neste caso, o tratador é executado quando o mecanismo assume que o usuário saiu efetivamente do parque e procede cancelando a reserva do usuário em todas as filas nas quais está cadastrado. Por outro lado, o tratador `OffUserHandler` é executado quando o mecanismo assume que o usuário está com problemas no dispositivo móvel, mas ainda permanece no parque. Este tratador modifica o *status* do usuário a fim de assegurar que ele poderá participar das atrações mesmo depois de ter perdido sua vez nas filas. Ambos os tratadores sensíveis ao contexto estão associados ao escopo local, isto é, ao gerenciador das filas, não sendo necessário realizar a propagação da exceção.

A exceção `AttractionOffException` ocorre sempre que uma dada atração deixou de funcionar por algum defeito ou para sofrer algum tipo de manutenção urgente. Embora seja levantada diretamente pelo servidor no papel de gerenciador das filas, a exceção `AttractionOffException` também deve ser tratada pelos clientes nos dispositivos móveis, os quais podem receber sua ocorrência via propagação. O gerenciador das filas especifica dois tratadores para a exceção `AttractionOffException`: `NotifyRegionHandler` e `NotifyGroupHandler`. Estes tratadores utilizam diferentes condições de contexto para propagar a exceção aos clientes na região da atração com defeito ou para um grupo de dispositivos que possui reserva para a atração. Dependendo das condições de contexto, ambos os tratadores do gerenciador das filas podem ser executados.

Por outro lado, ao receber a propagação da exceção, os dispositivos móveis da região ou grupo podem executar três tratamentos distintos, selecionados de acordo com a busca e as condições de contexto locais. O tratador `NotifyHandler` é executado quando o usuário não possui uma reserva para esta atração e procede apenas informando o usuário do problema bem como a previsão de retorno da atração. O tratador `RemoveHandler` é executado sempre que a atração estava reservada para o usuário e, além disso, a lista de reservas do usuário é muito grande. Este tratador procede simplesmente cancelando a reserva desta atração e informando ao usuário sobre o cancelamento da reserva. O tratador

`ModifyHandler` é executado quando o dispositivo possui uma pequena lista de reservas, sendo uma delas uma reserva para a atração com defeito. O tratador procura a atração mais próxima de acordo com a localização e as preferências do usuário, solicita entrada na fila da atração escolhida e finalmente notifica a mudança de reserva ao usuário.

A Tabela 2 resume as principais informações relacionadas a cada um dos tratadores definidos para esta aplicação, agrupando-os por exceção associada. Note que os tratadores para a exceção `DeviceNotFoundException` são executados no escopo do gerenciador das filas, não sendo necessário propagar a exceção. Note também que, para a exceção `AttractionOffException` foram especificados os tratadores `NotifyRegionHandler` e `NotifyGroupHandler` nos escopos de região e grupo, respectivamente. Após a execução destes tratadores ocorre a propagação automática para os escopos relacionados. Quando um dispositivo móvel recebe a propagação, procede levantando localmente a exceção e pesquisando os tratadores de acordo com as informações de contexto correntes.

Tabela 2. Tratadores e Escopos Associados em VL

Exceção: DeviceNotFoundException			
Tratador	Escopo	Verificação de Contexto	Tratamento
ExitedUserHandler (servidor)	Servidor	Não registrou defeito e não compareceu > 2 vezes.	Retirar das filas no parque.
OffUserHandler (servidor)	Servidor	Existe registro de defeito.	Alterar *status* do usuário nas filas.
Exceção: AttractionOffException			
Tratador	Escopo	Verificação de Contexto	Tratamento
NotifyRegionHandler (servidor)	Região	Exceção é grave.	Excluir fila. [Propagação Automática]
NotifyGroupHandler (servidor)	Grupo	Previsão de retorno > 30 min.	Excluir fila. [Propagação Automática]
NotifyHandler (dispositivo)	Dispositivo	Não existe reserva.	Informar problema e previsão de retorno da atração.
RemoveHandler (dispositivo)	Dispositivo	Existe reserva e lista de reservas grande.	Cancelar reserva.
ModifyHandler (dispositivo)	Dispositivo	Existe reserva e lista de reservas pequena.	Adicionar usuário em outra atração e notificar mudança.

7.1.2.
Outras Especificações

A arquitetura da aplicação VL define um servidor no papel de gerenciador das filas de um parque e um cliente correspondendo a um dispositivo móvel. Utilizamos os aspectos abstratos `MetaServer` e `MetaClient` (Seção 6.2.5) a fim de especificar a infra-estrutura de propagação de exceções no mecanismo. Nos aspectos `ConfigurationServer` e `ConfigurationClient`, subaspectos de `MetaServer` e `MetaClient`, respectivamente, foram definidas as exceções, os tratadores, os escopos, as associações entre tratadores e escopos, etc. A Figura 23 apresenta trechos de código do aspecto `ConfigurationServer`.

```
1: public aspect ConfigurationServer extends MetaServer {
2: ...
3: protected pointcut initialization() :
4:    execution(void ChamadaFilasPublisher.getInstance());
5: after() : initialization(){
6:    exception1 = new DeviceNotFoundException();
7:    exitedUser = new ExitedUserHandler(exception1);
8:    offUser = new OffUserHandler(exception1);
9:    ServerScope serverScope = ServerScope.getInstance();
10:   serverScope.attachHandler(exitedUser);
11:   serverScope.attachHandler(offUser);
12:   exception2 = new AttractionOffException(); ...
13: }
14: pointcut novoDispositivo(String mac): (execution(void
15:   Gerenciador.novoUsuario(*, String)) && args(*, mac));
16: after (String mac) : novoDispositivo(mac) { ...
17:   exception1.addContext(mac,
18:   "((Online = false) and (DeltaT > 50000))", cis);
19: }
20: pointcut atracaoFora(Atracao atracao, boolean status):
21:   (execution(void Atracao.setStatus(boolean))
22:   && args(status) && target(atracao));
23: after (Atracao atracao, boolean status) :
24:         atracaoFora (atracao, status) { ...
25:  if (status == false){
26:  regionScope.setRegionName(areaAtracao);
27:  groupScope.setDeviceList(atracao.getFila().getMacUsuarios());
28:  exception2.getContext().
29:         setProperty("Atracao",atracao.getNome()); ...
30:  exception2.raise();
31: }}}
```

Figura 23. Implementação de ConfigurationServer em VL

O conjunto de junção `initialization`, abstrato em `MetaServer`, é especificado como sendo a execução do método `getInstance` no objeto `ChamadaFilasPublisher` (linhas 3 e 4). Em outras palavras, através do conjunto de junção `initialization`, a aplicação móvel especifica o instante em que deseja dar início às atividades de tratamento de exceções. No adendo associado ao conjunto de junção `initialization`, são inicializadas as exceções contextuais (linhas 6 e 12), os tratadores sensíveis ao contexto (linhas 7 e 8) e os escopos (linhas 9, 10 e 11) da aplicação. Os outros conjuntos de junção da Figura 23 são definidos segundo com os interesses da aplicação em especificar comportamentos excepcionais particulares. Por exemplo, os conjuntos de junção `novoDispositivo` (linhas 14 e 15) e `paradaDeAtracao` (linhas 20 a 22) utilizam em sua especificação classes importantes para a definição das exceções contextuais. Observe que o método `addContext` é utilizado na linha 17 para adicionar interesses excepcionais à exceção `DeviceNotFoundException`. Em conseqüência, não é necessário que esta exceção seja levantada diretamente pela aplicação, uma vez que sua ocorrência é detectada pelo mecanismo. Por outro lado, quando uma atração tem seu *status* modificado para *false*, é realizada a definição do escopo para a região da atração (linha 26) e para o escopo dos dispositivos com reserva para esta atração (linha 27). Em seguida, `AttractionOffException` é diretamente levantada através do método `raise` (linha 30).

A Figura 24 apresenta trechos de código do aspecto `ConfigurationClient`.

```
1 : public aspect ConfigurationClient extends MetaClient{
2 : . . .
3 : protected pointcut initialization() :
4:    initialization(ChamadaFilaListener.new(..));
5 : after() : initialization() {
6 :   exception = new AttractionOffException();
7:    NotifyHandler notify = new NotifyHandler(exception);
8:    RemoveHandler remove = new RemoveHandler(exception);
9:    ModifyHandler modify = new ModifyHandler(exception);
10:   DeviceScope deviceScope = DeviceScope.getInstance();
11:   deviceScope.attachHandler(notify);
12:   deviceScope.attachHandler(remove);
13:   deviceScope.attachHandler(modify);
14: }}
```

Figura 24. Implementação de ConfigurationClient em VL

Na Figura 24, no adendo associado ao conjunto de junção `initialization` (linhas de 5 a 14), é inicializada a exceção contextual `AttractionOffException` (linha 6). Também são inicializados três tratadores sensíveis ao contexto (linhas 7 a 9) para esta exceção. Os tratadores são então associados ao escopo do dispositivo (linhas 11 a 13). A Figura 25 apresenta o código de um destes tratadores, o tratador `NotifyHandler`.

```
1: public class NotifyHandler extends Handler {
2: public boolean verifyContextCondition(){
3:    CompositeContext context = getException().getContext();
4:    String atracao = context.getProperty("Atracao");
5:    for(Iterator reservas =
6:        MainCliente.getUsuario().getReservas().iterator();
7:        reservas.hasNext();){
8:          Reserva reserva = (Reserva)reservas.next();
9:          if (reserva.getAtracao().getNome().equals(atracao)){
10:               return false;
11:         }
12:   }
13:   return true;
14: }
15: public boolean execute(){
16:   CompositeContext context = getException().getContext();
17:   String atracao = context.getProperty("Atracao");
18:   String prob = context.getProperty("Problema");
19:   String previsao = context.getProperty("Previsao");
20:   System.out.println(atracao+" - "+ prob +" - "+ previsao);
21:   return true;
22: }
23: }
```

Figura 25. Implementação de NotifyHandler em VL

Na Figura 25, o método `verifyContextCondition` (linhas 2 a 14) é especificado pela aplicação para retornar *true* quando o dispositivo móvel não possui reservas para a atração com defeito. Após verificar as condições de contexto, o mecanismo invoca o método `execute` (linhas 15 a 22), que recupera informações sobre o contexto excepcional (linhas 17 a 19) e exibe estas informações para o usuário.

7.2.
Health Care (HC)

Este segundo estudo de caso utiliza o protótipo da aplicação HC, que oferece suporte ao monitoramento de pacientes com doença cardiovascular sob cuidados médicos. Este estudo foi introduzido no Capítulo 3 deste trabalho. As próximas seções discutem o uso de nosso mecanismo para o tratamento de exceções sensível ao contexto na aplicação HC.

7.2.1.
Exceções Contextuais e Tratadores Sensíveis ao Contexto

A exceção contextual `HeartAttack` ocorre quando os sensores de HC identificam alguma situação anormal na atividade cardiovascular de um paciente. Esta exceção pode estar relacionada a vários contextos excepcionais, associados aos diferentes sensores da aplicação. As expressões `(BPM<40 or BPM>120)`, `(SystolicBloodPressure > 80)` e `(DiastolicBloodPressure < 100)` ilustram exemplos de contextos excepcionais.

A Tabela 3 apresenta uma descrição de possíveis tratadores sensíveis ao contexto para a exceção `HeartAttack`, bem como os escopos, verificação de contexto e descrição do funcionamento geral destes tratadores.

Tabela 3. Exceção Contextual HeartAttack e seus Tratadores em HC

Exceção: HeartAttack			
Tratador	Escopo	Verificação de Contexto	Funcionamento
NotifyRegionHandler	Região	A região ainda não foi notificada.	Informar sobre o ataque cardíaco aos dispositivos na mesma região. Como gravidade da exceção é alta, propagar automaticamente para o grupo de suporte à emergência.
NotifyNeighborhoodHandler	Região	Tratador NotifyRegionHandler já foi executado sem sucesso e a vizinhança ainda não foi notificada.	Informar sobre o ataque cardíaco aos dispositivos na vizinhança (regiões próximas).
NotifyMyDoctorHandler	Dispositivo	Status do médico está ativo.	Solicitar atendimento urgente ao médico.
EmergencySupportHandler	Grupo	Grupo de suporte à emergência ainda não foi chamado.	Solicitar atendimento ao grupo de suporte à emergência.

7.2.2. Propagação Sensível ao Contexto

Um cenário que ilustra o tratamento da exceção contextual `HeartAttack`, considerando os tratadores apresentados na Tabela 3 é como segue:

1. no dispositivo do cliente, o mecanismo de tratamento de exceções detecta a ocorrência da exceção contextual `HeartAttack` e inicia a atividade de busca por tratadores associados à exceção;
2. de acordo com a seqüência de escopos estabelecida, são encontrados os tratadores `NotifyRegionHandler` e `NotifyNeighborhoodHandler`, os quais pertencem ao primeiro nível de granularidade dos escopos, ou seja, ao escopo de região;
3. considerando as informações de contexto, somente o tratador `NotifyRegionHandler` é selecionado. Este tratador procede informando aos dispositivos da região do paciente quais os procedimentos de emergência que devem ser realizados e solicitando apoio ao grupo de suporte à emergência;
4. caso não obtenha nenhuma resposta no tempo esperado, a exceção é novamente levantada no dispositivo do cliente. Neste caso, as condições de contexto do tratador `NotifyNeighborhoodHandler` são atendidas e o alerta sobre o paciente e a necessidade do procedimento de emergência é enviado aos dispositivos nas regiões vizinhas;
5. se novamente não for obtida alguma resposta, a busca por tratadores passa para o próximo nível de granularidade, os escopos de dispositivo. O tratador `NotifyMyDoctorHandler` está associado ao escopo do dispositivo pertencente ao médico do paciente.

A Figura 26 ilustra a definição da exceção `HeartAttack` e seus tratadores sensíveis ao contexto especificados no dispositivo móvel do paciente.

```
1 : public aspect ConfigurationClient extends MetaClient{
2 : . . .
3 : protected pointcut initialization() :
4:    execution(void Sistema.getInstance());
5: after() : initialization() {
6:    exception = new HeartAttack();
7:    exception.addContext("(SystolicBloodPressure > 180 and
8:          DiastolicBloodPressure > 100 and BPM<40)”);
9:    notifyRegion = new NotifyRegionHandler(exception);
10:   notifyNeighbor = new NotifyNeighborhoodHandler(exception);
11:   notifyDoctor = new NotifyMyDoctorHandler(exception);
12:   notifyEmergency = new EmergencySupportHandler(exception);
13:   regionScope = RegionScope.getInstance();
14:   regionScope.attachHandler(notifyRegion);
15:   regionScope = RegionScope.getInstance(“Neighborhood”);
16:   regionScope.attachHandler(notifyNeighbor);
17:   deviceScope = DeviceScope.getInstance(“MyDoctor”);
18:   deviceScope.attachHandler(notifyDoctor);
19:   grpScope = DeviceGroupScope.getInstance(“EmergencySupport”);
20:   grpScope.attachHandler(notifyEmergency);
21: } }
```

Figura 26. Implementação de ConfigurationClient em HC

Na Figura 26, a exceção contextual `HeartAttack` é inicializada (linhas 6 a 8) no adendo associado ao conjunto de junção `initialization` (linhas 5 a 21). Também são inicializados quatro tratadores sensíveis ao contexto (linhas 9 a 12) para esta exceção. Os tratadores são então associados aos escopos de região (linhas 13 a 16), de dispositivo (linhas 17 e 18) e de grupo (linhas 19 e 20).

A Figura 27 apresenta o código de um dos tratadores para a exceção `HeartAttack`, o tratador `NotifyRegionHandler`. Tal tratador estende a classe abstrata `Handler`, que requer a implementação dos métodos abstratos `verifyContextCondition` e `execute`. O primeiro método verifica se o contexto de execução corrente é realmente apropriado para a execução de tal tratador e o segundo executa a funcionalidade do tratador.

```
1: public class NotifyRegionHandler extends Handler {
2: public boolean verifyContextCondition(){
3:    Vector list = getException().getHandledScopes();
4:    if (list.contains(this.getScope())) return false;
5:    return true;
6: }
7: public boolean execute(){
8:    if (getException().getSeverity() == veryHigh){
9:       DeviceGroupScope groupScope =
10:         DeviceGroupScope.getInstance("EmergencySupport");
11:      getException().propagateTo(groupScope);
12:   }
13:   return true;
14: }
15: }
```

Figura 27. Implementação de NotifyHandler em HC

Na Figura 27, o método `verifyContextCondition` (linhas 2 a 6) retorna *true* quando a exceção ainda não foi tratada pelo escopo da região onde está localizada a vítima de ataque cardíaco. Após verificar as condições de contexto, o mecanismo automaticamente invoca o método `execute` (linhas 7 a 22). É verificada a necessidade de se realizar a propagação explícita da exceção contextual para o grupo `EmergencySupport`, de acordo com a gravidade da exceção (linha 8). A propagação é efetivada através da chamada ao método `propagateTo` (linha 11). Ao final da execução do método `execute`, o mecanismo realiza a propagação automática para a região associada ao tratador.

7.2.3. Seqüência de Prioridade Entre os Escopos

Para a aplicação HC, exemplos típicos de escopos de grupo e de região incluem, respectivamente, a equipe de suporte à emergência e a região onde se encontra a vítima de ataque cardíaco. Nosso mecanismo de tratamento de exceções sensível ao contexto permite que as aplicações implementem uma estratégia diferente da estratégia pré-definida no mecanismo para estabelecer a seqüência entre os escopos. Isto permite aumentar a variabilidade do mecanismo na tarefa de realizar a busca por tratadores e a propagação de exceções contextuais. Por exemplo, a aplicação HC pode definir uma estratégia diferente para a exceção `HeartAttack`; pela própria definição desta exceção, é interessante deixar a escolha da estratégia para o paciente. Conseqüentemente, um paciente

pode estabelecer que o primeiro escopo a ser avisado é a região em que ele se encontra no momento da ocorrência da exceção contextual, depois o escopo deve ser o dispositivo do médico e, finalmente, o escopo deve ser a equipe de emergência. Outro paciente poderia preferir avisar em primeiro lugar o escopo da equipe de emergência, depois o escopo que representa a sua família e por último o escopo relacionado ao dispositivo do médico. A Figura 28 ilustra um exemplo de como esta seqüência pode ser estabelecida por um paciente para auxiliar a busca por tratadores e a propagação de exceções contextuais.

```
1: public class HeartAttackStrategy extends HandlerStrategy {
2: public HeartAttackStrategy (ContextException exception){
3:    super(exception);
4: }
5: public void defineSequence(){
6: addNextLevel(RegionScope.getInstance());
7: addNextLevel(RegionScope.getInstance("Neighborhood"));
8: addNextLevel(DeviceScope.getInstance("MyDoctor"));
9: addNextLevel(DeviceGroupScope.getInstance("EmergencySupport"));
10:}
11: . . .
12: }
```

Figura 28. Estratégia para Busca de Tratadores de HeartAttack em HC

Na Figura 28, o método `defineSequence` (linhas de 5 a 10), definido como abstrato na classe `HandlerStrategy`, é especificado em `HeartAttackStrategy` a fim de permitir que a aplicação HC estabeleça a seqüência desejada para os escopos que deve ser utilizada na busca por tratadores. Para associar a estratégia `HandlerStrategy` e a exceção `HeartAttack`, deve ser utilizado o método `setStrategy` definido na classe `ContextualException`.

7.2.4. Tratadores Sensíveis ao Contexto

Para lidar com os contextos excepcionais, é necessário definir um tratador sensível ao contexto que permita a definição das condições de contexto relacionadas antes que ele seja executado. Por exemplo, a Figura 29 descreve um novo tratador de exceções, o tratador `FirstHelp`. No HC, este tratador está

associado com a região do paciente (neste caso é a sua casa), uma vez que toda pessoa que mora junto com ele pode ajudar em uma situação excepcional.

Quando uma exceção contextual é detectada, o mecanismo executa o método `verifyContextCondition` para cada tratador definido naquele escopo. Se este método retorna *true*, o mecanismo invoca `execute`; se não, o mecanismo segue para o próximo tratador definido. Esta abordagem promove flexibilidade no suporte à definição de tratadores sensíveis ao contexto. Em MoCA, um evento excepcional pode ser detectado a partir de uma definição de um observador de eventos; contudo, esta abordagem usa apenas os dados do evento a fim de decidir o que fazer com este evento. Como mostra a Figura 29, em nosso mecanismo de tratamento de exceções sensível ao contexto, o tratador pode checar a situação corrente do contexto da aplicação a fim de decidir se ele trata ou não uma exceção específica. Isto permite a definição de um tratador sensível ao contexto para lidar com condições excepcionais de contexto.

```
1: public class FirstHelp extends Handler {
2: public boolean verifyContextCondition(){
3:    CompositeContext context = getException().getContext();
4:    if (Integer.parseInt(context.getProperty(“BPM”)) < 20)
5:         return false;
6:    else return true;
7: }
8: public boolean execute(){
9:    return inform_sequence_of_first_help();
10: }
11:}
```

Figura 29. Implementação de FirstHelp em HC

7.2.5. Propagação de Exceções Proativa

No tratamento proativo de exceções imprevistas, o receptor deve iniciar uma colaboração excepcional a fim de obter um tratador adequado para estas exceções. Nesta situação, o receptor pode colaborar com o dispositivo da vítima, executando os primeiros socorros enquanto a ambulância não chega. A Figura 30 descreve a definição do tratador `NotifyNeighborhoodHandler` através do uso da propagação proativa.

```
1: public class NotifyNeighborhoodHandler extends Handler {
2: public boolean verifyContextCondition(){
3:    Vector list = getException().getHandledScopes();
4:    if (list.contains(this.getScope())) return false;
5:    return true;
6: }
7: public boolean execute(){
8:    RegionScope regionScope =
9:          RegionScope.getInstance("Neighborhood");
10:   getException().propagateAsProactive(regionScope);
11:   return false;
12: }
13: }
```

Figura 30. Definição de Propagação Proativa em HC

A principal diferença deste tratador em relação ao tratador `NotifyHandler` da Figura 27 é a utilização do método `propagateAsProactive` (linha 10). Este método propaga a exceção `HeartAttack` para todos os dispositivos disponíveis na região da vizinhança, caso anteriormente não tenha sido tratada na própria região da vítima. Esta propagação permite uma expansão para um novo grupo de colaboração. Além disso, a proatividade da propagação suporta um mecanismo dinâmico para descoberta de tratadores. Provavelmente o grupo da vizinhança não sabe como tratar esta exceção. A fim de tratar este problema, o grupo da vizinhança inicia uma atividade colaborativa a fim de descobrir e executar possíveis tratadores. O mecanismo procede enviando os possíveis tratadores juntamente com a exceção.

7.3. Análise do Mecanismo

O mecanismo tratou as restrições impostas pelo paradigma *publish-subscribe* diretamente através da implementação do modelo de tratamento de exceções proposto no Capítulo 4. Vale ressaltar que, de acordo com a abordagem proposta, a ocorrência de determinados contextos está diretamente relacionada a ocorrência de exceções e a forma como tais exceções devem ser tratadas. Desta forma, para testar o mecanismo e simular a ocorrência das exceções nos estudos de caso implementados, foi necessário apenas simular a ocorrência de certos contextos, relacionados as condições excepcionais, no middleware MoCA.

Considerando a propagação de exceções, por exemplo, o processo de subscrição de MoCA não suporta a definição de níveis de usuários e somente aqueles que registraram interesse recebem as informações da ocorrência de contexto. O modelo de propagação do nosso mecanismo utiliza os níveis de escopo especificados para realizar propagação uma exceção, permitindo assim propagar uma exceção apenas para o grupo de dispositivos desejado. Além disso, leva em consideração a severidade de uma exceção para realizar a propagação de forma proativa, mesmo sem registro de interesse na exceção.

Quanto ao modelo de contextos adotado por MoCA, embora seja simples, de fácil implementação, podemos dizer que este modelo é também bastante limitado. Por esta razão, tivemos que adicionar certo nível de complexidade ao nosso mecanismo de tratamento de exceções, a fim de que contextos excepcionais mais complexos pudessem ser devidamente tratados. Esta complexidade adicional afetou principalmente a concepção do componente `ContextualException`, responsável pelo gerenciamento dos contextos excepcionais no mecanismo. Entre as questões que precisaram ser contornadas, podemos citar: (i) a impossibilidade de especificar contextos complexos, formados hierarquicamente por outros contextos mais simples; deste deriva (ii) a impossibilidade de definir uma relação entre os contextos.

Para tratar a questão levantada no item (i) definimos as classes `SimpleContext` e `CompositeContext`, que permitiram a especificação de contextos mais complexos, formados por contextos simples. Adicionalmente, definimos também métodos específicos para manipulação destas informações, como `matching(subject, expression)`, que permite estabelecer uma relação entre contextos. Com isso, solucionamos o problema citado em (ii).

Considerando os estudos de caso, a aplicação *Virtual Lines* permitiu-nos iniciar a definição do modelo de tratamento de exceções sensível ao contexto e de suas abstrações mais gerais como “contextos excepcionais”, escopos (dispositivo, servidor, região e grupo) e tratamento sensível ao contexto. Entretanto, somente o estudo da aplicação *Health Care* permitiu-nos explorar mais profundamente as questões relacionadas ao tratamento de exceções em aplicações móveis colaborativas, como a propagação de exceções e o tratamento de exceções proativo.

De fato, embora a aplicação *Virtual Lines* tenha sido desenvolvida utilizando todos os escopos propostos pelo modelo de tratamento de exceções (dispositivo, grupo, região e servidor), isso não redundou na implementação de cenários complexos de colaboração entre os dispositivos móveis que permitissem a demonstração das muitas possibilidades de tratamento e propagação de exceções contextuais. Ao contrário, embora a aplicação HC fosse um pouco mais restrita quanto à utilização dos diferentes tipos de escopo, ela permitiu-nos especificar situações excepcionais mais interessantes que envolveram, por exemplo, a propagação de exceção passando por vários níveis de escopo, a propagação automática e a propagação proativa de exceções. Além disso, pelas próprias características da aplicação HC, tornou-se mais fácil pensar em cenários que descrevessem contextos excepcionais em aplicações móveis.

Através da aplicação HC, pudemos perceber também a existência de situações em que algumas exceções que podem ser levantadas concorrentemente significam a ocorrência de uma situação normal mais séria. Um exemplo é a detecção simultânea de uma exceção de ataque cardíaco e uma impossibilidade de atendimento pelo sistema de ambulância. Neste caso, não é satisfatório o procedimento de ativação de vários tratadores para cada condição excepcional individual. Para resolver este problema, um sistema de controle central precisaria ser capaz de coletar todas as ocorrências de exceções concorrentes e resolvê-las de maneira que ação apropriada seja disparada. Em outras palavras, torna-se necessária uma resolução de exceções concorrentes que infere uma única exceção a partir de múltiplas situações concorrentes anormais. MoCA não fornece qualquer suporte para tal resolução concorrente de eventos e esta é ainda uma questão aberta em nosso mecanismo de tratamento de exceções.

Enfim, por tudo o que expomos e apesar de algumas questões em aberto, podemos dizer que nosso mecanismo oferece um efetivo tratamento de exceções sensível ao contexto no domínio de aplicações móveis, pois leva em consideração conceitos originais como contextos excepcionais, fluxo de controle excepcional e colaboração excepcional.