Tratamento de Exceções Sensível ao Contexto

Karla Damasceno[1], Nélio Cacho[2], Alessandro Garcia[2], Carlos Lucena[1]

[1]Departamento de Informática – PUC-Rio – Brasil

[2]Computing Department – Lancaster University, United Kingdom

{karla,lucena}@inf.puc-rio.br, {n.cacho,a.garcia}@lancaster.ac.uk

Abstract. *Exception handling on mobile systems is not a trivial task due to their intrinsic characteristics: context-awareness, asynchrony, and intermittent connectivity. Conventional mechanisms of exception handling can not be used for many reasons. One of the main problems is that the error recovery and the exception handling strategies frequently need to be selected according to contextual information. This paper identifies limitations of existent mechanisms, as well as it proposes a new model and architecture for context-sensitive exception handling. The evaluation of our proposal has been performed through the implementation of three applications from heterogeneous domains.*

Resumo. *Tratamento de exceções em aplicações móveis não é uma tarefa trivial devido às características destas aplicações, como sensibilidade ao contexto, comunicação assíncrona e conexão instável. Mecanismos convencionais de exceções não podem ser utilizados por várias razões. Um dos principais problemas é que a busca e execução dos tratadores e a propagação de exceções precisam freqüentemente considerar informações de contexto. Este artigo identifica limitações dos mecanismos existentes, bem como propõe um novo modelo e uma arquitetura para tratamento de exceções sensível ao contexto. A avaliação da solução proposta foi conduzida através da implementação de três aplicações de domínios heterogêneos.*

1. Introdução

Os avanços recentes da Computação Móvel têm possibilitado a construção de aplicações sensíveis ao contexto capazes de monitorar e utilizar dinamicamente informações que provêm do ambiente ou usuário (Coutaz et al., 2005). Tais aplicações precisam tratar variações freqüentes em seus contextos de execução, tais como, mudanças de temperatura, bateria e localização. Embora a incorporação de mecanismos apropriados para tratamento de exceções seja fundamental para o desenvolvimento de sistemas móveis confiáveis, o projeto de tais mecanismos não é uma tarefa trivial em função das próprias características destes sistemas, tais como mobilidade, abertura, conexão instável e comunicação assíncrona. Estas características incorrem em maior imprevisibilidade das exceções e na necessidade de novos mecanismos modulares apropriados para realizar a propagação de eventos excepcionais. Além disso, existe um sério conflito da natureza síncrona do tratamento de exceções tradicional com a instabilidade de conexão e a comunicação assíncrona inerentes as aplicações móveis sensíveis ao contexto.

Até o momento, o que se observa é que a maior parte das aplicações utiliza apenas o mecanismo de exceções fornecido pelas linguagens de programação subjacentes (Garcia et al., 2001). Entretanto, as abstrações e mecanismos convencionais não são satisfatórios por vários motivos (Cacho et al., 2006; Garcia, 2005; Tripathi & Miller, 2000). Primeiro, a propagação de exceções deve considerar mudanças contextuais que constantemente ocorrem nas aplicações móveis. Segundo, as atividades de recuperação de erros e a estratégia de tratamento de exceções freqüentemente precisam ser selecionadas de acordo com informações de contexto, podendo ser necessário ativar diferentes tratadores para uma mesma exceção de acordo com o contexto. Além disso, a própria caracterização de uma exceção pode depender do contexto dos dispositivos - isto é, um estado do sistema pode ser considerado errôneo em uma dada localização onde o dispositivo se encontra, mas não em outra localização.

Não existem trabalhos na literatura que provêem suporte a tratamento de exceções sensível ao contexto para o desenvolvimento de aplicações móveis robustas. Em geral, as propostas existentes são restritas a solucionar questões gerais relacionadas às aplicações móveis baseadas em agentes (Tripathi & Miller, 2000; Souchon et al., 2004; Iliasov & Romanovsky, 2005). Estes mecanismos não possuem as abstrações necessárias para realizar um tratamento de exceções sensível ao contexto, como exemplo, a possibilidade de definir escopos que estejam associados diretamente a localização física dos dispositivos. Embora os sistemas de *middleware* que oferecem suporte à construção de aplicações móveis sensíveis ao contexto (Iliasov & Romanovsky, 2005; Sacramento et al., 2004; Capra et al., 2003) considerem as abstrações referentes a sensibilidade ao contexto, não existe suporte para engenheiros de software lidarem explicitamente com o tratamento de exceções contextuais.

Neste sentido, as contribuições deste trabalho são as seguintes: (a) identificação de um conjunto de requisitos necessários ao desenvolvimento de um mecanismo de exceções para aplicações móveis; (b) um modelo com abstrações adequadas para tratamento de exceções sensível ao contexto, desenvolvido para atender os requisitos anteriores; (c) uma arquitetura de software para um mecanismo de tratamento de exceções sensível ao contexto, que pode ser adotada por outras implementações; e (d) implementação de um mecanismo de tratamento de exceções sensível ao contexto usando MoCA (Sacramento et al., 2004), baseado no modelo e na arquitetura propostos.

Este artigo está organizado como a seguir. A seção 2 descreve conceitos referentes a sensibilidade ao contexto e ao tratamento de exceções e analisa os problemas para um efetivo tratamento sensível ao contexto. As seções 3 e 4 apresentam um modelo e uma arquitetura para um mecanismo de tratamento de exceções em aplicações móveis sensíveis ao contexto. A seção 5 apresenta uma avaliação do mecanismo. A seção 6 apresenta os trabalhos relacionados. Finalmente, a seção 7 apresenta as conclusões e os trabalhos futuros.

2. Sensibilidade ao Contexto e Tratamento de Exceções

Inicialmente apresentamos as definições de contexto, aplicações sensíveis ao contexto e sistemas de *middleware* para desenvolver estas aplicações (Seção 2.1). Em seguida, descrevemos os principais conceitos relacionados ao tratamento de exceções e apresentamos a evolução dos mecanismos de exceções e a necessidade de um

mecanismo específico para estas aplicações (Seção 2.2). Finalmente, a partir da análise da aplicação *Virtual Lines* desenvolvida em MoCA, identificamos um conjunto de requisitos necessários para incorporar a sensibilidade ao contexto no tratamento de exceções em aplicações móveis (Seção 2.3). Esta mesma aplicação será utilizada na Seção 5 para avaliar a aplicabilidade do mecanismo proposto.

2.1. Sensibilidade ao Contexto

Contexto é qualquer informação utilizada para caracterizar a situação de uma pessoa, lugar ou objeto relevante para a interação entre um usuário e uma aplicação (Dey & Abowd, 1999). Uma aplicação é *sensível ao contexto* se utiliza informações de contexto para fornecer serviço ou informação relevante para o usuário (Dey & Abowd, 1999). Tais aplicações podem usar diversas informações como perfil de usuários, bateria, localização, temperatura e pressão sanguínea. Desta forma, além de ter conhecimento sobre sua "situação" em um dado momento, tais aplicações possuem a habilidade para interpretar e usar o contexto como base para um comportamento adaptativo.

Para permitir esta adaptação baseada em contexto, diversos paradigmas de coordenação têm sido adotados para a implementação de sistemas de *middleware* específicos, como (a) baseado em eventos ou *publish-subscribe* (Sacramento et al., 2004), (b) baseado em espaços de tuplas (Iliasov & Romanovsky, 2005) e (c) reflexivo (Capra et al., 2003). Tais sistemas são responsáveis por realizar a coleta das informações de contexto, a disseminação destas informações e a notificação de alterações de contexto de acordo com o interesse dos clientes. Neste artigo, também discutiremos como o nosso modelo (Seção 3) e a arquitetura (Seção 4) para tratamento de exceções sensível ao contexto foram implementados em um *middleware publish-subscribe*, chamado MoCA (Sacramento et al., 2004).

De fato, o modelo e arquitetura propostos são especialmente adequados para mecanismos comumente definidos em sistemas de *middleware publish-subscribe*. A principal justificativa para a seleção deste paradigma está relacionada à própria natureza das aplicações móveis, uma vez que o paradigma *publish-subscribe* se apresenta como uma alternativa interessante por duas características fundamentais: (i) comunicação assíncrona, a comunicação pode ocorrer mesmo se o destino está indisponível; e (ii) comunicação anônima e interação desacoplada, um publicador e um subscritor não precisam conhecer um ao outro. Estas duas características impulsionaram a utilização deste paradigma, de modo que hoje ele é adotado por um número crescente de sistemas de *middleware* sensíveis ao contexto (Pietzuch & Bacon, 2002; Meier & Cahill, 2002; Sacramento et al., 2004; Muthusamy et al., 2005; Cugola & Cote, 2005), inclusive por produtos de grandes indústrias de software como a IBM (MQTT, 2006). Ademais, o modelo de tratamento de exceções também pode ser estendido para arquiteturas baseadas em espaços de tuplas, pois estas soluções apresentam vários mecanismos similares ao paradigma *publish-subscribe*. De fato, a maioria dos sistemas para aplicações móveis se apóia nestes dois paradigmas.

2.2. Tratamento de Exceções

Desenvolvedores de sistemas confiáveis freqüentemente se referem a erros como exceções porque erros raramente se manifestam durante a atividade normal do sistema (Goodenough, 1975; Parnas & Würges, 1976). Em situações de erro, um componente

gera exceções que modelam a condição de erro e o sistema deve realizar o tratamento daquelas exceções (Lee & Anderson, 1990). Desta forma, *tratamento de exceções* é a capacidade que um software possui de reagir apropriadamente diante da ocorrência de exceções, continuando ou interrompendo sua execução, a fim de preservar a integridade do estado do sistema (Garcia et al., 2001).

Quando um serviço requerido não pode ser realizado, ele retorna uma *resposta anormal*, ou exceção (Figura 1a). Ao receber uma resposta anormal de um outro componente (exceção externa) ou levantar uma exceção durante sua própria atividade normal (exceção interna), as atividades adequadas para tratamento da exceção devem ser executadas. Um *tratador* de exceções é a parte da *atividade anormal* (ou excepcional), é, portanto, a parte do código da aplicação que fornece medidas específicas da aplicação para o tratamento da exceção levantada (Garcia et al., 2001). Este é vinculado a uma região particular do código normal, chamada *região protegida* ou *escopo de tratamento*. Se uma exceção é levantada em uma região protegida, o fluxo de controle normal é desviado para um *fluxo de controle excepcional* (Garcia et al., 2001). Após o tratamento da exceção, o componente pode voltar a prover o serviço normal ou *propagar* a exceção para um componente de mais alto nível.

Mecanismos de tratamento de exceções provêem suporte explícito a propagação de exceções, e mudanças no fluxo de controle normal para o fluxo de controle excepcional (Goodenough, 1975; Parnas & Würges, 1976). Eles também são responsáveis por suportar diferentes estratégias de fluxo excepcional e procurar os tratadores apropriados depois que a ocorrência de uma exceção for detectada. Além disso, eles devem ter um projeto simples, serem fáceis de usar, e fornecer uma separação explícita entre o código normal e excepcional (Garcia et al., 2001). Mecanismos de tratamento de exceções são tipicamente parte de linguagens de programação ou são uma característica de sistemas de *middleware* dedicadas a diferentes domínios de aplicação e seguindo diferentes estilos de arquitetura.

A Figura 1 ilustra a evolução dos mecanismos de tratamento de exceções, considerando (a) programas seqüenciais, (b) programas distribuídos concorrentes e (c) aplicações móveis sensíveis ao contexto. O modelo tradicional para tratamento de exceções em programas seqüenciais e apresentado na Figura 1a. Este modelo consiste basicamente de um componente ideal tolerante a falhas (Lee & Anderson, 1990), com sua parte de atividade normal e anormal (tratadores), as requisições de serviços e as exceções internas que são tratadas de forma síncrona ou precisam ser propagadas sincronamente aos componentes de mais alto nível. Entretanto este mecanismo torna-se inapropriado para programas distribuídos concorrentes (Xu et al., 1995), onde uma exceção que ocorre em uma *thread*, envolve todas as *threads* participantes da colaboração. Assim, como ilustra a Figura 1b, para tratar exceções que ocorrem em programas distribuídos concorrentes, é necessário adicionar o conceito de transação, e as exceções precisam ser propagadas de forma síncrona para todas as *threads* envolvidas.

Entretanto, o modelo do componente ideal (Figura 1a) e o conceito de transação (Figura 1b) não são satisfatórios para incorporação de tratamento de exceções em aplicações móveis sensíveis ao contexto. Isto ocorre porque a instabilidade nas conexões e a presença de comunicação assíncrona inviabilizam a implementação da sincronicidade exigida pela manutenção das transações.

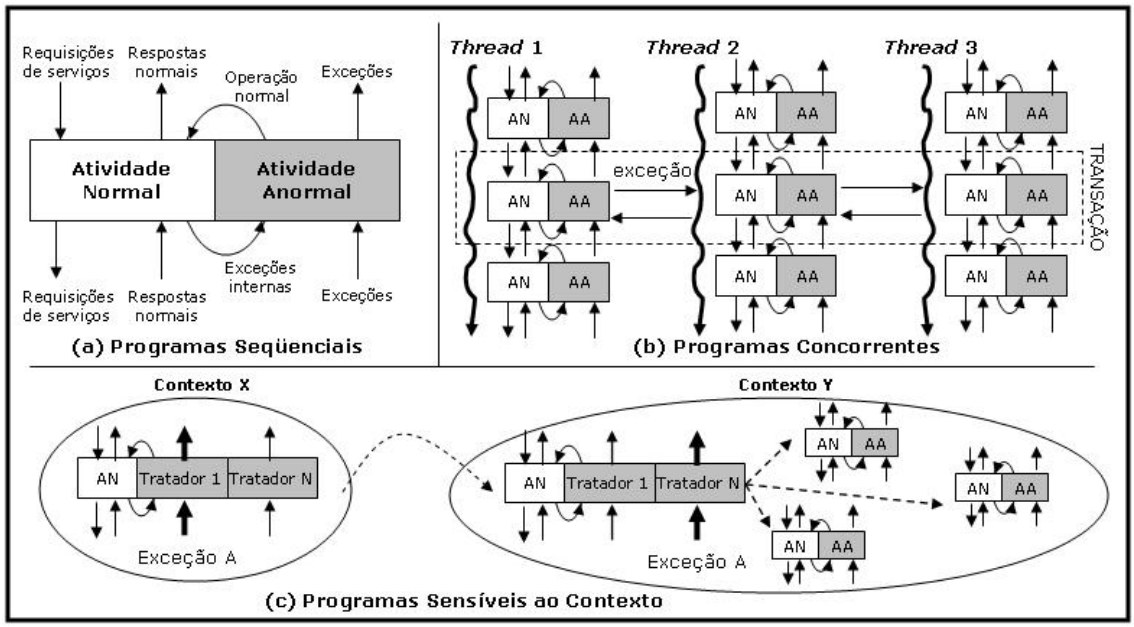

Figura 1. A evolução dos mecanismos de tratamento de exceções

Devido suas características de dinamicidade e mobilidade, as aplicações usualmente devem considerar os contextos onde estão atuando e quais usuários do sistema estão inseridos (Figura 1c). Assim como informações de contexto devem ser consideradas na execução das atividades normais do sistema, as variações de contextos também precisam ser consideradas nas atividades relacionadas ao tratamento de exceções. Além disso, diversas outras abstrações precisam ser consideradas, como contexto, localização, múltiplos tratadores, exceções dependentes de contexto, diferentes estratégias de propagação dependendo do contexto onde uma exceção é levantada, e assim por diante.

2.3. Exemplo de Aplicação Sensível ao Contexto

Virtual Lines é uma aplicação móvel sensível ao contexto cujo protótipo foi desenvolvido com a 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 e aproveitem melhor 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 para que ele retorne e participe da atração. Quando o usuário não retorna a tempo, o sistema emite um alerta avisando que perdeu a vez. Esta aplicação envolve contextos excepcionais que serão discutidos e analisados na próxima seção de tal forma a ilustrar as soluções de tratamento de exceções sendo propostas.

3. Modelo de Tratamento de Exceções Sensível ao Contexto

O nosso modelo de tratamento de exceções para aplicações móveis sensível ao contexto é constituído pelas características sendo apresentadas nas seções a seguir. Cada característica do modelo é apresentada da seguinte forma. Primeiro, discute-se a problemática envolvendo cada elemento do modelo, o qual é exemplificado com a

aplicação *Virtual Lines* (Seção 2.3). Quando apropriado, também argumenta-se por que soluções existentes (Seções 2.1 e 2.2), tais como *middleware publish-subscribe*, não provêem mecanismos adequados para cada um dos elementos apresentados.

3.1. Especificação de Contextos Excepcionais

De modo similar a definição de contexto proposta por Dey e Abowd (1999) (Seção 2.1), um "contexto excepcional" pode ser visto como um contexto que caracteriza uma situação excepcional de uma entidade, onde "excepcional" pode variar de acordo com os requisitos da aplicação. Um contexto excepcional corresponde então a um conjunto indesejável de condições que podem estar relacionadas a diferentes tipos de informação tais como uma região geográfica específica, a temperatura de uma sala e batimentos cardíacos de um paciente. O objetivo de definir um contexto excepcional é facilitar a definição de situações excepcionais, pois a ocorrência de um contexto excepcional está diretamente relacionada à ocorrência de uma exceção.

Entretanto, a especificação de uma situação de contexto excepcional possui uma semântica diferente e, como tal, precisa lidar com diferentes elementos em sua especificação, incluindo o escopo de tratamento, tratadores gerais alternativos, tipos de informação contextual que devem ou não ser propagadas junto com a ocorrência de exceção, e assim por diante. Para realizar o tratamento de contextos excepcionais é necessário prover suporte ao "fluxo de controle excepcional", que é inerentemente diferente do fluxo normal; este último consiste simplesmente em reações baseadas em notificação, enquanto o primeiro requer a propagação de contextos excepcionais aos tratadores apropriados, que também podem ou não ser selecionados dependendo de sua localização física. Por outro lado, a tarefa de propagação de um contexto excepcional pode exigir o envolvimento entre várias entidades. No caso da aplicação *Virtual Lines* (Seção 2.3), tais entidades podem ser o grupo de dispositivos registrado em uma atração, os dispositivos na mesma localização da atração defeituosa e a equipe responsável pelo suporte no parque. Denominamos tal envolvimento entre entidades como uma "colaboração excepcional".

O mecanismo *publish-subscribe* não provê suporte direto e efetivo para a implementação de um fluxo de controle excepcional sensível ao contexto. Para sistemas *publish-subscribe,* existe a necessidade de subscrições que devem registrar explicitamente o interesse em um ou mais contextos. Além disso, é importante que dispositivos recebam notificações de contextos excepcionais, independente de terem registrado interesse nesta informação. No caso da *Virtual Lines*, um contexto excepcional poderia especificar a situação em que uma atração deixa de funcionar por algum defeito ou manutenção urgente.

3.2. Níveis de Escopo: Dispositivo, Servidor, Regiões ou Grupos

Há muitas situações em que o tratamento de exceções requer que diversos dispositivos estejam envolvidos dependendo da região física e outros tipos de informação contextual. Por exemplo, para a aplicação *Virtual Lines* o tratamento dos contextos excepcionais que se referem a parada de uma atração pode envolver diferentes conjuntos de dispositivos, como o servidor do parque, os dispositivos presentes na região física onde se encontra a atração, o grupo responsável pela manutenção no parque, ou os dispositivos registrados na fila desta atração.

Além disso, deve ser possível que dependendo da gravidade de uma situação excepcional, a exceção seja propagada para todas as regiões e grupos envolvidos. Conseqüentemente, as regiões físicas ou um determinado grupo de dispositivos são também exemplos de escopos de tratamento contextual que devem ser suportados pelo *middleware* subjacente. Contudo, os sistemas de *middleware* não suportam tais escopos de tratamento de exceções sensível ao contexto, diminuindo a modularidade do sistema na presença de contextos excepcionais.

A fim de suportar uma abordagem sensível ao contexto para o tratamento adequado de erros, exceções podem ser capturadas através de escopos em quatro níveis diferentes: um dispositivo, um grupo de dispositivos, um servidor e uma região. Escopos de dispositivo e servidor compreendem as unidades operacionais básicas do sistema sensível ao contexto, o que permite que a funcionalidade do tratador de exceções seja encapsulada dentro do escopo do próprio dispositivo ou servidor. Tais escopos estão relacionados a abstrações próprias dos sistemas *publish-subscribe*.

Um escopo de grupo envolve um conjunto de dispositivos definidos pela aplicação para suportar o tratamento cooperativo de uma exceção entre os dispositivos que pertencem ao grupo. É possível inserir ou remover elementos do grupo de acordo com a necessidade da aplicação. Por exemplo, para a aplicação *Virtual Lines*, agentes que representam a equipe de manutenção podem formar um grupo específico, assim quando ocorrem exceções relacionadas a problemas em uma atração, todos os membros do grupo são notificados. Diferentemente dos outros três escopos, um escopo de região tem um comportamento mais dinâmico para identificar os dispositivos que estão participando deste escopo. Um escopo de região está diretamente relacionado a uma localização física do ambiente onde estão os dispositivos móveis.

3.3. Busca Sensível ao Contexto por Tratadores

Para ser sensível ao contexto, a busca dos tratadores apropriados para uma ocorrência de exceção deve ser realizada considerando os escopos e tratadores sensíveis ao contexto associados a exceção. Entretanto, tais elementos não são considerados pelos mecanismos convencionais, não sendo possível utilizar informações de contexto para realizar a busca. Por exemplo, dependendo das informações de contexto, a busca pode ser feita primeiramente entre os tratadores associados a regiões. Esta ordem pode ser previamente definida por uma aplicação, entretanto, se nenhuma ordem for explicitamente definida, o mecanismo de tratamento de exceções deve fornecer uma seqüência crescente pré-estabelecida para os diversos tipos de escopo, como exemplo, dispositivo < grupo < região < servidor. Esta seqüência estabelecida entre os diversos escopos é utilizada também para executar a propagação sensível ao contexto.

3.4. Tratadores Sensíveis ao Contexto

A seleção dos tratadores de exceções apropriados depende de condições de contexto dos dispositivos envolvidos no tratamento de exceções. Para a mesma exceção, é necessário criar tratadores específicos para diferentes condições de contexto e garantir que eles sejam corretamente executados. Na aplicação *Virtual Lines* por exemplo, para a situação excepcional que indica a parada de uma atração, pode ser necessário utilizar informações de contexto sobre a localização dos dispositivos e o tipo de problema ocorrido. No caso da parada na atração indicar uma situação excepcional do ambiente,

como fogo, os tratadores associados são executados nos dispositivos localizados na mesma região da atração. Novamente, temos que implementar tal controle de tratadores sensíveis ao contexto como parte da aplicação móvel, uma vez que não existe uma facilidade que dê suporte a este requisito nos sistemas de *middleware*.

Antes de disparar a execução de um tratador, é interessante que sejam feitas algumas verificações de contexto de acordo com as necessidades específicas da aplicação. Assim, de acordo com o contexto de execução, pode ser apropriado executar um tratador, ao passo que para a mesma exceção em outro contexto, o tratador não deva ser utilizado. Neste caso, a pesquisa pelo tratador apropriado deve continuar até encontrar outro tratador que satisfaça as condições de contexto. O propósito desta abordagem é promover flexibilidade extra que ofereça suporte a definição de tratadores sensíveis ao contexto. Além disso, após a execução dos tratadores, deve ser feita uma checagem para verificar a necessidade de realizar a propagação automática de exceções.

3.5. Propagação Sensível ao Contexto

A propagação é considerada sensível ao contexto, pois as informações de contexto e escopo são utilizadas para que o mecanismo decida quando deve ocorrer a propagação das exceções. A verificação da necessidade de realizar a propagação automática deve ser realizada após busca do tratador e sua posterior execução. Três situações podem ocorrer após a execução do tratador: (i) o tratador foi executado com sucesso e está associado a um escopo de execução local; (ii) o tratador foi executado com sucesso e está associado a um escopo de execução remota; (iii) o tratador não foi executado com sucesso. No primeiro caso não é necessário realizar a propagação automática. No segundo caso, após a execução local do tratador, a exceção deve ser propagada para o escopo associado a ele. Finalmente, no terceiro caso, deve ser realizada uma nova busca por tratadores que estejam associados a escopos no próximo nível de granularidade.

Além da propagação automática, o mecanismo pode permitir que a propagação seja solicitada explicitamente pelas aplicações sensíveis ao contexto.

3.6. Tratamento de Exceções Proativo

Em uma aplicação móvel aberta, não podemos esperar que todos os dispositivos, nos quais os agentes de software são desenvolvidos por diferentes projetistas, sejam capazes de prever todos os possíveis contextos excepcionais. No caso de *Virtual Lines*, por exemplo, durante a ocorrência de um incêndio em uma atração, se por alguma razão a equipe de manutenção do parque não estiver disponível, é útil que a exceção seja propagada para outros dispositivos realizarem o suporte emergencial, os quais podem ser selecionados a partir de informações de perfil dos usuários, como profissão, idade e sexo. Entretanto, estes usuários podem não ser capazes de tratar tal notificação. Adicionalmente, mesmo que os dispositivos em uma região não tenham registrado interesse em uma exceção, é fundamental notificar a ocorrência de uma exceção grave, aos dispositivos que pertencem a mesma região.

Em outras palavras, a exceção contextual deve ser proativamente levantada em outros agentes colaborativos e/ou dispositivos móveis pertencentes à mesma região ou escopo. Isto pode ser feito explorando a infra-estrutura móvel de colaboração fornecida pelo *middleware publish-subscribe* para desenvolver aplicações sensíveis ao contexto, na qual um dispositivo pode colaborar, por exemplo, com os dispositivos que estão na

mesma localização que ele. A proatividade precisa ser considerada sob dois pontos de vista (i) a notificação de exceções contextuais deve ser proativa: os dispositivos precisam tomar conhecimento da ocorrência de exceções mesmo que não tenham registrado interesse em obter informações sobre o contexto excepcional relacionado à exceção, e (ii) o tratamento de exceções precisa ser proativo: deve ser possível que os dispositivos colaborem para adquirir os possíveis tratadores para uma exceção contextual que eles não são capazes de tratar.

4. Arquitetura de Software para Tratamento de Exceções Contextuais

Esta seção apresenta uma arquitetura de software que objetiva modularizar os interesses de tratamento de exceções considerando a sensibilidade ao contexto. Neste sentido, os componentes da arquitetura separam os interesses de tratamento de exceções dos componentes correspondentes às funcionalidades básicas e aos outros interesses de aplicações sensíveis ao contexto. Vários componentes da arquitetura proposta são responsáveis por gerenciar o fluxo resultante da ocorrência de contextos excepcionais, não suportados comumente por mecanismos e sistemas de *middleware* existentes (Seção 3). As aplicações sensíveis ao contexto podem reusar facilidades fornecidas pelos componentes (Seção 4.1) e interfaces (Seção 4.2) da arquitetura a fim de tratar seus contextos excepcionais. Além disso, a arquitetura proposta é independente de linguagens de programação ou plataformas específicas, descrevendo um projeto em alto nível de abstração dos componentes, relacionamentos, responsabilidades e interfaces.

4.1. Componentes

A Figura 2 ilustra a arquitetura proposta para tratamento de exceções sensível ao contexto. Ela possui os seguintes componentes: (i) `ContextualException`, (ii) `Handler`, (iii) `ExceptionHandlingStrategy`, (iv) `PropagationManager`, (v) `ExceptionPropagation`, e (vi) `HandlingScope`. O componente `Middleware` é um componente externo que interage com os componentes da arquitetura de tratamento de exceções. Este componente representa a infra-estrutura *publish-subscribe* subjacente utilizada para recuperação das informações de contexto.

O componente `ContextualException` é responsável pela especificação de exceções, bem como gerenciamento das informações relacionadas, tais como nome, descrição, contexto excepcional, e assim por diante. Como esta arquitetura está relacionada à utilização do paradigma *publish-subscribe*, sempre que um contexto excepcional é associado com uma exceção, este componente possui a tarefa de subscrever interesse no recebimento de eventos com aquela informação. Existindo portanto uma relação direta entre este componente e o modelo de contexto adotado pelo *middleware* orientado a contexto. Além disso, este componente é responsável por manter atualizadas as informações sobre ocorrências de contextos excepcionais. Por exemplo, se um contexto excepcional está relacionado à entrada de dispositivos em uma região, uma informação útil associada a ocorrência deste contexto excepcional é qual dispositivo entrou na região especificada. Para manter estas informações atualizadas, este componente precisa interceptar todas as ocorrências de contextos excepcionais enviadas pelo *middleware*. Este componente também possui a tarefa de levantar as exceções contextuais, tão logo sejam sinalizadas as ocorrências de suas condições de contexto, ou quando solicitadas pelo componente de propagação ou aplicação.

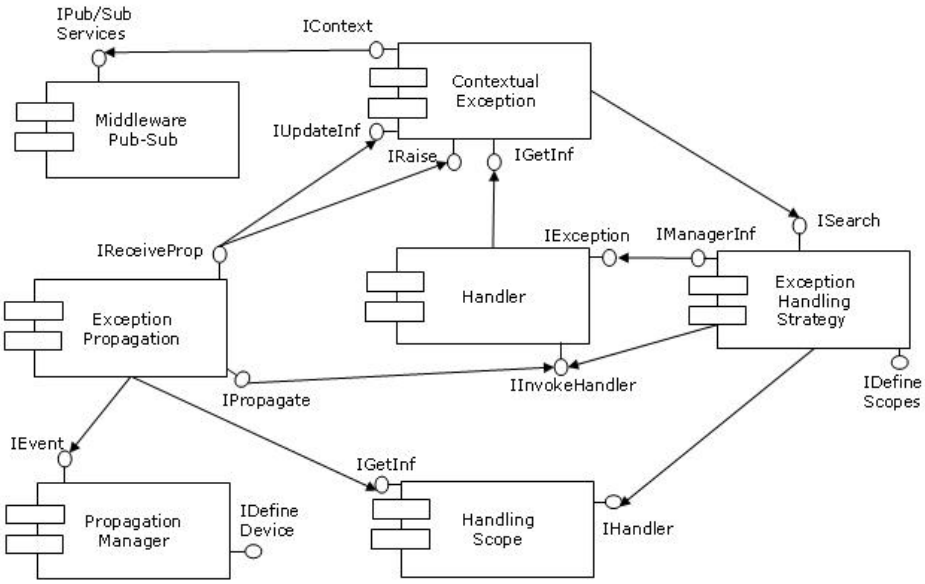

Figura 2. Arquitetura para Tratamento de Exceções Sensível ao Contexto

O componente `Handler` é responsável por especificar os tratadores e realizar as verificações de condições de contexto, necessárias para realizar um tratamento sensível ao contexto. Para isto, solicita as informações de contexto ao componente `ContextualException`. Além disso, este componente possui a tarefa de chamar os tratadores quando suas condições de contexto são satisfeitas. O componente `PropagationManager` tem a função de manter uma infra-estrutura *publish-subscribe* para permitir a propagação das exceções. Este componente permite informar se a aplicação executará o papel consumidor ou publicador de eventos, de acordo com a API do *middleware publish-subscribe* adotado.

Um componente diretamente relacionado a este é o componente `ExceptionPropagation`, que envia solicitações para realizar a propagação de exceções, seguindo a especificação do componente gerenciador. Este componente é responsável também pelo recebimento de exceções propagadas por outros dispositivos, recuperação local dos tratadores associados com a ocorrência de exceção, além de solicitar o levantamento da exceção recebida ao componente `ContextualException`. Além disso, este componente possibilita que seja realizada a propagação das exceções, assim que o tratador terminar sua execução. Considerando esta execução, existem duas possibilidades de realizar a propagação: (i) no caso de insucesso, repetição da busca por escopos de nível superior ou (ii) quando a execução é bem sucedida, verificação do escopo e propagação automática para o escopo especificado.

O componente `ExceptionHandlingStrategy` é responsável por realizar o gerenciamento das exceções e seus respectivos tratadores, mantendo um controle geral desta informação, o qual pode ser acessado por todos os outros componentes. A possibilidade de múltiplos tratadores associados a uma única exceção permite a associação dinâmica entre uma ocorrência de exceção e seu respectivo tratador, portanto é fundamental para aplicações sensíveis ao contexto. Além disso, este componente permite a especificação de uma estratégia de busca de tratadores e uma ordem de prioridade entre os tipos de escopo da aplicação. Esta seqüência de escopos será utilizada tanto na busca de tratadores, como na propagação das exceções. Este

componente tem a função de realizar as buscas de tratadores, considerando suas informações de contexto. É importante a existência de estratégias gerais, usadas no caso de nenhuma estratégia ser especificada pela aplicação. Após realizar a busca de tratadores, este componente solicita ao componente `Handler` a execução dos tratadores. Finalmente, o componente `HandlingScope` permite especificar os escopos de tratamento do mecanismo, gerenciar os tratadores associados aos escopos e recuperar dinamicamente quais os dispositivos que estão associados a um escopo.

4.2. Interfaces

As interfaces dos componentes da arquitetura podem ser utilizadas por outros componentes da arquitetura ou diretamente por aplicações que utilizem o mecanismo de tratamento de exceções. As interfaces podem ser: (i) privadas, definem serviços visíveis somente para os componentes da arquitetura; e (ii) públicas, definem serviços visíveis tanto pela arquitetura quanto pelas aplicações. A Figura 3 apresenta as interfaces publicas e privadas para cada um dos componentes da arquitetura.

Na Figura 3a o componente `ContextualException` implementa três interfaces públicas e uma privada. A interface `IRaise` possibilita que a aplicação levante e propague as exceções diretamente através dos métodos `raise` e `propagateTo`. A interface `IGetInf` permite que a aplicação e os outros componentes da arquitetura obtenham informações sobre as ocorrências de exceções e seus contextos excepcionais e a interface `IUpdateInf` permite que sejam feitas atualizações nestas informações. A interface `IContext` subscreve o interesse em receber as informações ao *middleware publish-subscribe* e recebe os contextos excepcionais enviados pelo *middleware*.

A Figura 3b ilustra as interfaces do componente `HandlingScope`. A interface pública `IHandler` permite que a aplicação associe seus tratadores com um dado escopo de tratamento e recupere todos os tratadores existentes para um dado escopo. A interface privada `IGetInf` permite recuperar dinamicamente, através do *middleware* orientado a contexto, os dispositivos que estão relacionados a um dado contexto. Já a Figura 3c ilustra as interfaces para o componente `Handler`. A interface `IException` permite a manutenção das informações sobre a exceção que está associada a um tratador. A interface `IInvokeHandler` permite que o componente `ExceptionHandlingStrategy` verifique se as condições de contexto dos tratadores são satisfeitas quando estiver realizando a busca de tratadores, além de disparar a execução dos tratadores.

A Figura 3d apresenta o componente `ExceptionHandlingStrategy`. A interface pública `IDefineScopes` permite que a aplicação especifique a seqüência desejada dos escopos que deve ser utilizada na busca de tratadores. A interface `ISearche` permite realizar a busca de tratadores. A interface `IManagerInf` permite gerenciar uma tabela de referências com a relação existente entre uma exceção e seus possíveis tratadores. Isto é feito sempre que ocorre a definição de uma nova exceção e a associação de uma exceção a um tratador. Depois que uma exceção contextual é levantada, os componentes da arquitetura interagem para realizar as atividades de gerenciamento. A informação extra sobre uma ocorrência da exceção é atualizada implicitamente pelos componentes da arquitetura, entretanto a aplicação também pode adicionar outras informações de acordo com sua necessidade.

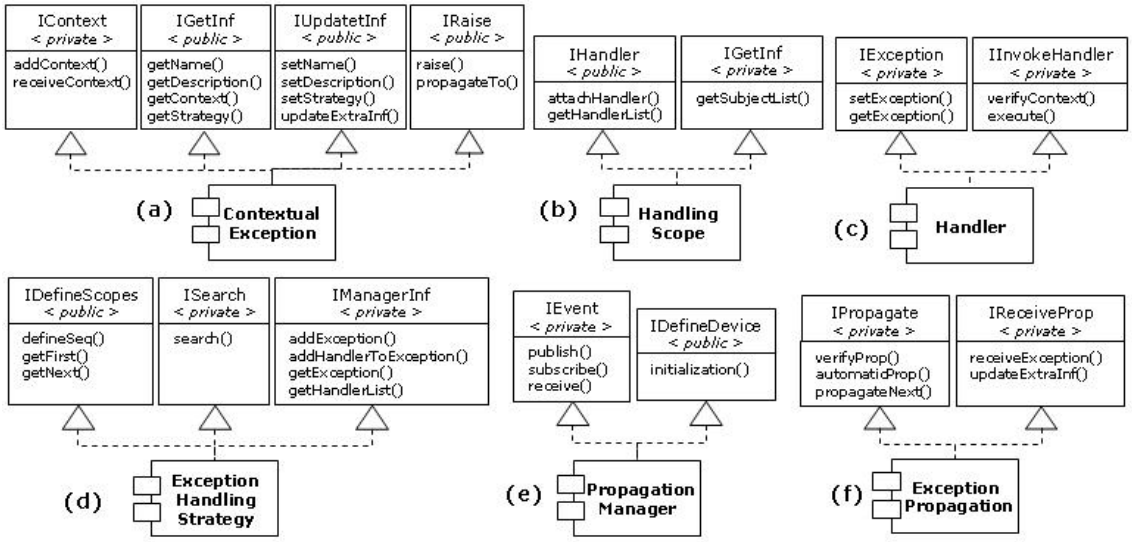

Figura 3. Interfaces dos Componentes da Arquitetura

A Figura 3e apresenta o projeto do componente `PropagationManager`. A interface privada `IEvent` é responsável por fornecer eventos com as exceções que foram propagadas por outros dispositivos e permitir a publicação de exceções. A interface pública `IDefineDevice` permite que a aplicação especifique e inicie o papel desempenhado pela aplicação na estrutura de propagação. Finalmente, a Figura 3f apresenta o projeto do componente `ExceptionPropagation`. Sempre que um tratador termina sua execução, a interface `IPropagate` verifica a necessidade de realizar a propagação automática para o escopo associado ao tratador, ou a propagação para um escopo de maior nível de granularidade, por este motivo esta interface entrecorta o fim da execução dos tratadores. A interface `IReceiveProp` permite que, após o recebimento de uma exceção via propagação, sejam atualizadas as informações de contexto necessárias para realizar o tratamento. Após a atualização destas informações, a exceção é levantada, para que então seja tratada pelo mecanismo.

5. Avaliação do Mecanismo

Esta seção apresenta a avaliação do modelo e arquitetura propostos. Devido a restrições de espaço, este artigo descreve mais detalhadamente apenas o estudo de caso baseado na aplicação *Virtual Lines* (Seção 2.3). Entretanto, outros dois estudos foram realizados, utilizando uma aplicação de *Health Care*, para suporte ao monitoramento de pacientes com doença cardiovascular sob cuidados médicos (Cacho et al., 2006) e uma aplicação de ambiente inteligente (*AmI*), um ambiente inteligente com diversos sensores para controle de temperatura, alarme de incêndio, entre outros (Damasceno et al., 2006).

5.1. Estudos de Caso

Para o estudo de caso da aplicação *Virtual Lines*, foram definidas 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. Por exemplo, a exceção `DeviceNotFound Exception` 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. 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.

No 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`, por exemplo, 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.

Para o segundo estudo de caso, uma aplicação para monitoramento de pacientes chamada *Health Care*, foram definidos como contextos excepcionais possíveis situações relacionadas a um ataque cardíaco. Por exemplo, a inexistência de familiares no mesmo local onde está um paciente pode implicar na necessidade de propagar a exceção para outros dispositivos localizados na mesma região. Além disso, caso não seja possível enviar a equipe de emergência ao local, pode ser necessário solicitar auxílio aos dispositivos móveis de profissionais da saúde localizados em regiões próximas ao paciente. No terceiro estudo, a aplicação AmI, a temperatura de uma região ultrapassando um valor máximo ou mínimo pode indicar uma situação excepcional; adicionalmente, se outras informações de contexto forem consideradas, tais como, a temperatura de outras regiões ou a quantidade de dispositivos, a exceção UnableToHeat pode ser propagada para a equipe de manutenção dos equipamentos.

A avaliação dos estudos levanta questões como a distinção entre fluxo normal e fluxo excepcional de eventos. O fluxo normal é representado pelo envio e recebimento de informações de contexto, enquanto o fluxo excepcional é realizado pela propagação de exceções que foram detectadas a partir das informações de contexto. Esta distinção permite atribuir prioridade de execução para o fluxo excepcional em detrimento ao normal, já que uma vez detectada, a exceção deve ser imediatamente tratada. Um segundo fator trata da possibilidade de se modificar o fluxo excepcional, uma vez que para isso é necessário apenas definir novas estratégias de propagação (Seção 4) que permitam adaptar o mecanismo aos requisitos da aplicação alvo.

5.2. Questões de Implementação, Benefícios e Limitações do Mecanismo Proposto

O modelo e a arquitetura propostos foram implementados usando a arquitetura MoCA (Sacramento et al., 2004). Detalhes de implementação encontram-se descritos em (Damasceno, 2006). Consideramos que a utilização do paradigma *publish-subscribe* e do *middleware* MoCA se mostrou como uma boa alternativa para a implementação do mecanismo de tratamento de exceções. Entretanto, pode ser interessante utilizar outros paradigmas e sistemas de *middleware* especialmente para ampliar o modelo de tratamento de exceções. Para implementar alguns componentes do mecanismo de exceções foi feito uso de programação orientada a aspectos (Kiczales et al., 1997), o que permitiu uma clara separação entre os interesses excepcionais e os outros interesses das aplicações sensíveis ao contexto. Para o estudo de caso da aplicação *Virtual Lines*, as classes e aspectos relacionados ao tratamento de exceções ficaram totalmente separados das outras classes da aplicação.

Como pode ser abstraído da seção anterior e de outros estudos de caso que realizamos (Cacho et al., 2006; Damasceno, 2006; Damasceno et al., 2006), o mecanismo proposto (Seções 3 e 4) alivia o programador de aplicações móveis de uma série de complexidades inerentes a tratamento de erros em tais aplicações. Por exemplo, o programador da aplicação não tem necessidade de implementar toda a complexidade para (i) utilização de diferentes estratégias de propagação de erros, podendo apenas definir a prioridade entre os níveis de escopo ou selecionar uma estratégia pré-definida pelo mecanismo; bem como para (ii) realizar a notificação de contextos excepcionais. Por outro lado, percebemos a necessidade da evolução do mecanismo de exceções (Seção 3) para prover suporte à resolução concorrente de condições excepcionais.

6. Trabalhos Relacionados

Não existem trabalhos na literatura que consideram os requisitos de sensibilidade ao contexto no tratamento de exceções de aplicações móveis. Existem alguns trabalhos que consideram questões gerais relacionadas ao tratamento de exceções em aplicações móveis baseadas em agentes. Entretanto, os mecanismos propostos restringem-se a características específicas de propriedades como mobilidade e autonomia de agentes. Por exemplo, na abordagem de Tripathi e Miller (2000), exceções são propagadas para agentes especiais denominados “guardiões”, os quais implementam reações gerais para as exceções. Esta solução é extremamente restritiva tendo em vista que a maior parte do código de tratamento de exceções em sistemas reais é específico de aplicação (Garcia et al., 2001). Além do tratamento não ser sensível ao contexto, cria-se um gargalo pela sua centralização em um único agente.

A abordagem de Souchon et al. (2004) não possibilita a associação dinâmica entre tratadores e ocorrências de exceção, fundamental para o tratamento de exceções no domínio de sistemas com mobilidade e sensibilidade a contexto. Em Iliasov e Romanovsky (2005), Context-Aware Mobile Agents (CAMA) é um *framework* para o desenvolvimento de aplicações móveis que suporta o conceito de escopos aninhados, que agrupam erros e os tratadores de exceção aos quais estão associados. Contudo, não existe suporte ao tratamento sensível ao contexto. Por exemplo, CAMA não trata a definição de contextos excepcionais e a busca de tratadores sensível ao contexto. Finalmente, alguns de nossos trabalhos nesta linha (Damasceno et al., 2006; Cacho et al., 2006) apresentaram apenas os estudos de caso exploratórios. Este artigo, do contrário, apresenta um modelo e uma arquitetura genéricos para Tratamento de Exceções Sensível ao Contexto.

7. Conclusões e Trabalhos Futuros

Mecanismos especializados de tratamento de exceções tem sido consistentemente uma questão central em Engenharia de Software de tal forma a promover melhor confiabilidade no desenvolvimento de sistemas de software (IEEE TSE, 2000). Este trabalho motivou a necessidade para mecanismos de tratamento de exceções sensível ao contexto que foi detectada: (i) no desenvolvimento de uma série de aplicações com o *middleware* MoCA (Seção 5), (ii) em uma análise extensiva dos vários sistemas de *middleware* existentes (Damasceno, 2006), e (iii) uma avaliação das soluções existentes de tratamento de erros em aplicações móveis (Seção 6). Isto nos permitiu obter um conjunto de requisitos e definir um modelo de tratamento sensível ao contexto com: (i) suporte explícito para especificação de "contextos excepcionais", (ii) busca sensível ao contexto por tratadores de exceção, (iii) escopo de tratamento multi-nível que fornece novas abstrações (tais como grupos), e abstrações relacionadas ao *middleware* sensível ao contexto subjacente, como dispositivos, regiões, e servidores, (iv) propagação de erros sensível ao contexto e (v) tratamento de exceções proativo.

Como trabalho futuro, pretendemos evoluir o mecanismo de exceções proposto, provendo suporte à resolução de exceções contextuais concorrentes e tratando das limitações identificadas na Seção 5. Além disso, estamos planejando avaliar a generalidade do modelo proposto (Seção 3) a partir de aplicações móveis construídas com outros paradigmas de coordenação (Seção 2.1), como o paradigma baseado em espaços de tuplas. Finalmente, pretendemos estender o mecanismo de tratamento de exceções para suportar mobilidade de código em adição a mobilidade física.

Referências Bibliográficas

Cacho, N. et al. Handling Exceptional Conditions in Mobile Collaborative Applications: An Exploratory Case Study. 4th Intl. Workshop on Distributed and Mobile Collaboration, Manchester, Jun 2006.

Capra, L. et al. CARISMA: Context-Aware Reflexive mIddleware System for Mobile Applications. IEEE Transactions on Software Engineering, v.29, n.10, Oct. 2003, p. 929-945.

Coutaz, J. et al. Context is Key. Communications of ACM. v.48, n.3, Mar. 2005, p. 49-53.

Cugola, G.; Cote, J. E. M. On Introducing Location Awareness in Publish-Subscribe Middleware. 4th International Workshop on Distributed Event-Based Systems, 2005.

Damasceno, K. et al. Context-Aware Exception Handling in Mobile Agent Systems: The MoCA Case. 5th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2006) at ICSE 2006, Shanghai, China, May.

Damasceno, K. Tratamento de Exceções Sensível ao Contexto. Março 2006. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil.

Dey, A. K.; Abowd, G. D. Towards a better understanding of context and context-awareness. Conf on Human Factors in Computing Systems, Netherlands, Apr. 2000.

Garcia, A. et al. A Comparative Study of Exception Handling Mechanisms for Building Dependable Object-Oriented Software. Journal of Systems and Software, Elsevier, v.59, n.6, Nov. 2001, p.197-222.

Garcia, A. et al. Software Engineering for Large-Scale Multi-Agent Systems. SELMAS 2005. (Post-Workshop Report) ACM Software Engineering Notes, v. 30.

Goodenough, J. B. Exception handling: issues and a proposed notation. Commun ACM v. 18, n.12, Dec. 1975, p. 683-696.

IEEE TSE (IEEE Transactions on Software Engineering), Special Issue on Current Trends in Exception Handling, v. 26, n. 9, Sep. 2000.

Iliasov, A.; Romanovsky, A. CAMA: Structured Communication Space and Exception Propagation Mechanism for Mobile Agents. ECOOP-EHWS, Glasgow, Jul. 2005.

Kiczales, G. et al. Aspect-Oriented Programming. European Conference on Object-Oriented Programming (ECOOP), LNCS 1241, Springer, Finland, Jun. 1997.

Lee, P.; Anderson, T. Fault Tolerance: Principles & Practice. Springer, 2nd ed, Wien, Austria, Jan. 1990.

Meier, R.; Cahill, V. STEAM: Event-Based Middleware for Wireless Ad Hoc Networks. Intl. Workshop on Distributed Event-Based Systems. Austria, 2002.

MQTT. http://mqtt.org/

Muthusamy, V. et al. Publisher Mobility in Distributed Publish/Subscribe Systems. In Proc. 4th Intl. Workshop on Distributed Event-Based Systems (ICDCSW'05), 2005.

Parnas, D.; Würges, H. Response to Undesired Events in Software Systems. In Proc. 2nd Intl. Conference on Software Engineering. California, USA, p. 437-446, 1976.

Pietzuch, P.; Bacon, J. Hermes: A Distributed Event-Based Middleware Architecture. In Proc. Workshop on Distributed Event-Based Systems, 2002.

Sacramento, V. et al. MoCA: A Middleware for Developing Collaborative Aplications for Mobile Users. IEEE Distributed Systems Online, v.5, n.10, Oct. 2004.

Souchon, F. et al. Improving exception handling in multi-agent systems. In Software engineering for multi-agent systems II, Springer-Verlag, LNCS 2940, Feb. 2004.

Tripathi, A.; Miller, R. Exception Handling in Agent-Oriented Systems. In Proc. Advances in Exception Handling Techniques (ECOOPW'00), Springer-Verlag, LNCS 2022, 2000.

Xu, J. et al. Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery. In Proc. 25th FTCS, 1995.