5
Arquitetura de Tratamento de Exceções Sensível ao Contexto

Este capítulo apresenta uma arquitetura de software que tem por objetivo modularizar os interesses de tratamento de exceções considerando a sensibilidade ao contexto. Neste sentido, os componentes desta arquitetura separam os interesses de tratamento de exceções sensível ao contexto dos componentes correspondentes às funcionalidades básicas e aos outros interesses de aplicações sensíveis ao contexto. As aplicações sensíveis ao contexto podem reusar as facilidades fornecidas pelos componentes da arquitetura a fim de tratar seus contextos excepcionais.

Nossa arquitetura de software para tratamento de exceções é baseada no paradigma *publish-subscribe*. A principal justificativa para a seleção deste paradigma está relacionada à própria natureza das aplicações móveis. Mais especificamente, o paradigma *publish-subscribe* se apresenta como uma alternativa interessante, pois possui 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. Além disso, o fato do paradigma *publish-subscribe* ser adotado por um número crescente de *middlewares* sensíveis ao contexto (Pietzuch & Bacon, 2002; Meier & Cahill, 2002; Sacramento et al., 2004; Muthusamy et al., 2005; Cugola & Cote, 2005) possibilita a utilização mais ampla desta arquitetura de tratamento de exceções por aplicações sensíveis ao contexto.

Neste capítulo, apresentamos inicialmente uma visão geral de uma arquitetura de software para tratamento de exceções. Posteriormente, descrevemos a arquitetura para tratamento de exceções proposta neste trabalho. Finalmente, apresentamos uma análise crítica de suas características bem como de suas contribuições para o desenvolvimento de aplicações móveis.

5.1.
Uma Arquitetura de Software para Tratamento de Exceções

Uma *arquitetura de software* é uma descrição em alto nível da organização de um sistema em termos dos seus componentes e relacionamentos existentes entre estes componentes (Shaw, 1996). Cada componente desta arquitetura corresponde a um interesse específico no sistema e obedece a um conjunto de responsabilidades. A fim de cumprir suas responsabilidades, cada componente fornece um conjunto de interfaces, as quais permitem disponibilizar os serviços que são implementados por ele.

O interesse por arquiteturas de software e reuso de projeto pode motivar o desenvolvimento de uma arquitetura de software para tratamento de exceções. O uso da arquitetura permite que decisões mais detalhadas relacionadas ao tratamento de erros possam ser realizadas em fases posteriores do processo de desenvolvimento de software. Uma arquitetura de software para tratamento de exceções é também independente de linguagens de programação ou plataformas. Isto minimiza a complexidade ocasionada por problemas de entrelaçamento e espalhamento de interesses de tratamento de exceções nas aplicações.

Por outro lado, uma arquitetura de software para tratamento de exceções sensível ao contexto também pode ser implementada de modo a não induzir modificações intrusivas nas aplicações e nos elementos das APIs de *middlewares*. Por esta razão, entendemos que arquiteturas de software que utilizam o conceito de aspectos[4] (Kiczales et al., 2001) podem ser benéficas para o alcance efetivo dos requisitos de reusabilidade e manutenibilidade de aplicações sensíveis ao contexto. A noção de arquitetura orientada a aspectos proposta por Garcia (2004) introduz o conceito de *componente aspectual*.

Um componente aspectual separa os interesses transversais associados a este componente dos demais componentes arquiteturais do sistema. Cada um dos componentes aspectuais se relaciona com mais de um componente arquitetural, o que decorre de sua natureza transversal. Tais relacionamentos estão associados com as interfaces do componente e são denominados relacionamentos transversais[5]. As interfaces são categorizadas em dois grupos: (i) interfaces

[4] Do inglês, *aspects*.

[5] Do inglês, *crosscutting*.

normais, e (ii) interfaces transversais. Uma interface transversal é diferente de uma interface normal. Uma interface normal apenas especifica os serviços fornecidos aos outros componentes. As interfaces transversais, além de fornecerem serviços ao sistema, também especificam quando um aspecto arquitetural afeta outros componentes.

Um componente aspectual pode realizar mais de uma interface transversal uma vez que pode afetar múltiplos componentes de diferentes maneiras. Uma interface de aspecto entrecorta ou elementos internos de um componente ou elementos de outras interfaces. O primeiro caso significa que o aspecto arquitetural entrecorta a estrutura ou comportamento interno do componente. O segundo caso significa que o aspecto afeta um ou mais serviços disponibilizados pelas interfaces do outro componente. Uma notação para os elementos da arquitetura orientada a aspectos acima é introduzida na próxima seção.

Portanto, se uma arquitetura de software para tratamento de exceções for considerada no desenvolvimento de aplicações sensíveis ao contexto, há maior probabilidade de alcançar a modularização dos interesses do tratamento de exceções bem como de funcionalidades básicas e de outros interesses em aplicações sensíveis ao contexto. Particularmente, o uso de aspectos no nível arquitetural também pode ser usado para o alcance efetivo dos objetivos de reusabilidade e manutenibilidade do tratamento de exceções.

5.2. Componentes da Arquitetura

A arquitetura proposta possui os seguintes componentes: (i) componente `ContextualException`, (ii) componente `Handler`, (iii) componente `ExceptionHandlingStrategy`, (iv) componente `PropagationManager`, (v) componente `ExceptionPropagation`, e (vi) componente `HandlingScope`. As responsabilidades dos componentes podem ser classificadas em *dependentes* de aplicação (DA) ou *independentes* de aplicação (IA).

As responsabilidades dependentes de aplicação são diretamente relacionadas às funcionalidades de aplicação e incluem, por exemplo, as facilidades para especificação de exceções contextuais, tratadores e escopos. Assim, a fim de utilizar nossa arquitetura proposta, os desenvolvedores podem invocar serviços já

fornecidos pelas interfaces dos componentes da arquitetura ou podem refinar o projeto dos componentes de acordo com as necessidades de suas aplicações.

As responsabilidades independentes de aplicação incluem, por exemplo, facilidades para o gerenciamento de informação extra, a invocação de um tratador, o desvio do fluxo de controle e a busca de tratadores. Estas responsabilidades estão relacionadas ao gerenciamento das atividades de tratamento de exceções. Os componentes executam as atividades de gerenciamento de uma forma transparente à aplicação. Os componentes interagem entre si de acordo com as especificações da arquitetura a fim de cumprir as responsabilidades independentes da aplicação. A Tabela 1 resume os componentes e suas principais responsabilidades.

Tabela 1. Componentes e suas Responsabilidades na Arquitetura

ContextualException	• Especificação de exceções contextuais (DA) ; • Gerenciamento de informações da exceção, como nome, descrição, contexto excepcional, etc. (IA); • Subscrição de contextos excepcionais (IA); • Interceptação da ocorrência de contextos excepcionais e atualização da informação extra (IA); • Levantamento de exceções contextuais através da ocorrência de contextos excepcionais (IA), propagação de exceções contextuais (IA), ou levantamento explícito de exceções contextuais (DA).
Handler	• Especificação de tratadores sensíveis ao contexto (DA); • Verificação das condições de contexto dos tratadores e invocação de tratadores (IA).
PropagationManager	• Manutenção da infra-estrutura para propagação de exceções contextuais (DA).
ExceptionPropagation	• Recebimento de exceções contextuais que foram propagadas por outros dispositivos e atualização da informação extra (IA); • Recuperação dos novos tratadores associados localmente com a exceção recebida (IA); • Interceptação da execução dos tratadores, para posterior verificação se deve ocorrer propagação automática ou, em caso de insucesso, repetição da busca para maior nível de granularidade (IA) .
ExceptionHandlingStrategy	• Interceptação das associações entre exceções contextuais e tratadores sensíveis ao contexto, além do gerenciamento destas informações em uma tabela de referências de exceções e possíveis tratadores (IA); • Especificação de uma seqüência para os escopos (níveis de granularidade) e especificação de uma estratégia de busca de tratadores (DA); • Busca de tratadores para uma exceção contextual de acordo com estratégia sensível ao contexto especificada (IA).
HandlingScope	• Especificação dos escopos de tratamento (DA); • Gerenciamento dos tratadores sensíveis ao contexto do escopo (IA); • Recuperação dos dispositivos que estão envolvidos em um escopo (IA).

A Figura 6 apresenta a nossa arquitetura para tratamento de exceções sensível ao contexto. A modelagem da arquitetura é baseada em uma extensão da linguagem de modelagem ASideML (Chavez & Lucena, 2001; Chavez, 2004), usada neste trabalho. Esta linguagem estende a UML (Booch et al., 1999) com notações para representar aspectos. Os componentes aspectuais são representados como diamantes. As interfaces normais são representadas como círculos brancos; as interfaces transversais são representadas como círculos cinzas. A visão de um aspecto arquitetural suprime todas as informações sobre seus elementos internos, exceto os nomes de suas interfaces transversais. Cada interface transversal é ilustrada como um pequeno círculo com o nome da interface localizado próximo ao círculo. O círculo está ligado por uma linha sólida ao diamante representando o aspecto arquitetural. O relacionamento transversal é representado através de uma seta pontilhada no sentido do componente aspectual para o componente afetado.

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 recuperar as informações de contexto.

O componente `ContextualException` é responsável pela especificação de exceções, bem como gerenciamento das informações relacionadas a ela, tais como nome, descrição, contexto excepcional, etc. 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 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 dada região, uma informação útil associada a ocorrência deste contexto excepcional é o dispositivo que entrou na região especificada. Para manter estas informações atualizadas, este componente precisa interceptar todas as ocorrências de contextos excepcionais enviadas pelo *middleware* orientado a contexto. 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.

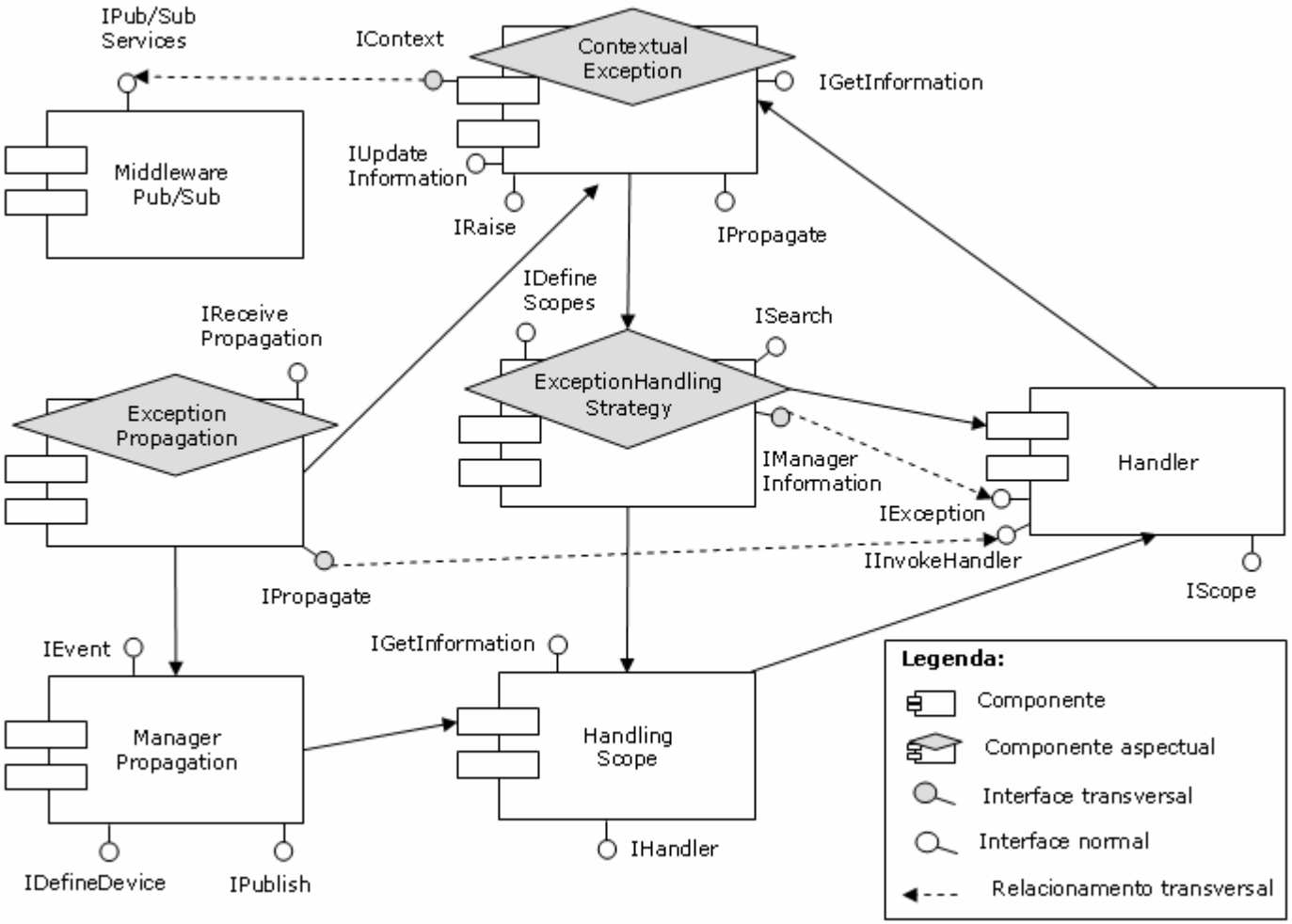

Figura 6. Arquitetura do Mecanismo de Tratamento de Exceções

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. Neste sentido, este componente permite informar se a aplicação executará o papel consumidor ou publicador de eventos. Esta distinção é necessária pois permite responder a uma série de questões referentes às diferenças de comportamento entre estes dispositivos, decorrentes do paradigma *publish-subscribe* adotado. Entre estas diferenças de comportamento pode-se citar, por exemplo: os parâmetros utilizados para a configuração, a inicialização do componente, o tipo de escopo local que deve ser associado ao dispositivo, a publicação de exceções realizada pelo dispositivo, etc.

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, atualização das informações de contexto, além de solicitar o levantamento da exceção recebida ao componente `ContextualException`. Além disso, este componente possibilita que seja realizada a propagação das exceções, através da interceptação do final da execução dos tratadores. Considerando esta execução, existem duas possibilidades de realizar a propagação: (i) no caso de insucesso, repetição da busca em um nível de granularidade 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 escopos 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.

5.3. Interfaces da Arquitetura

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 7 apresenta o projeto do componente `ContextualException` com suas interfaces.

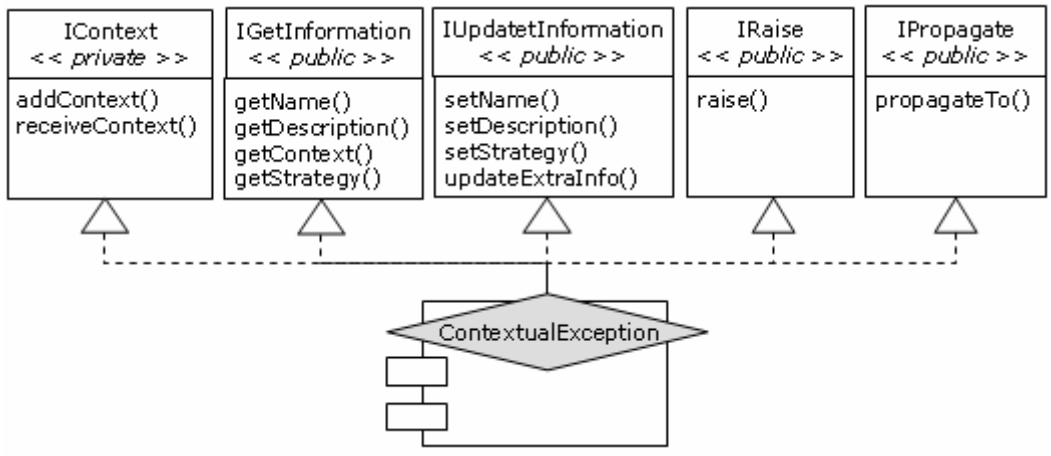

Figura 7. Componente ContextualException e Interfaces

O componente `ContextualException` implementa quatro interfaces públicas e uma interface privada. A interface privada é `IContext` e as interfaces públicas são (i) `IGetInformation`, (ii) `IUpdateInformation`, (iii) `IRaise`, e (iv) `IPropagate`. As interfaces `IRaise` e `IPropagate` possibilitam que a aplicação levante e propague as exceções diretamente através dos métodos `raise` e `propagateTo`. A interface `IGetInformation` permite que a aplicação e os outros componentes da arquitetura obtenham informações sobre as ocorrências de exceções e seus contextos excepcionais e a interface `IUpdateInformation` permite que sejam feitas atualizações nestas informações. A natureza ortogonal deste componente é especificada através de sua interface privada `IContext`. Esta interface entrecorta a associação de exceções e seus respectivos contextos excepcionais, e subscreve no *middleware publish-subscribe* o interesse em receber estas informações. Além disso, esta interface permite o recebimento dos contextos excepcionais que forem enviados pelo *middleware*.

A Figura 8 apresenta o componente `HandlingScope` com suas interfaces.

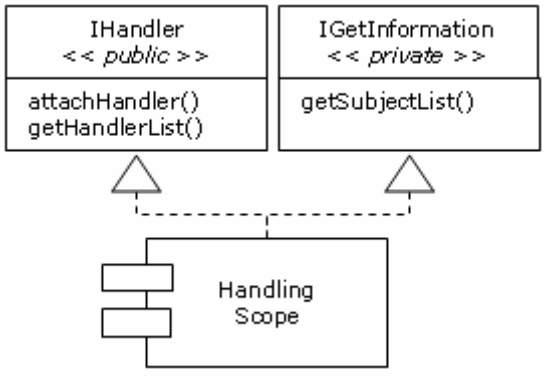

Figura 8. Componente HandlingScope e Interfaces

O componente `HandlingScope` implementa a interface pública `IHandler` e a interface privada `IGetInformation`. A primeira permite que a aplicação associe seus tratadores com um dado escopo de tratamento e recupere todos os tratadores existentes para um dado escopo. A segunda permite recuperar dinamicamente, através do *middleware* orientado a contexto, os dispositivos que estão relacionados a um dado contexto.

A Figura 9 apresenta o projeto do componente `Handler` com suas interfaces.

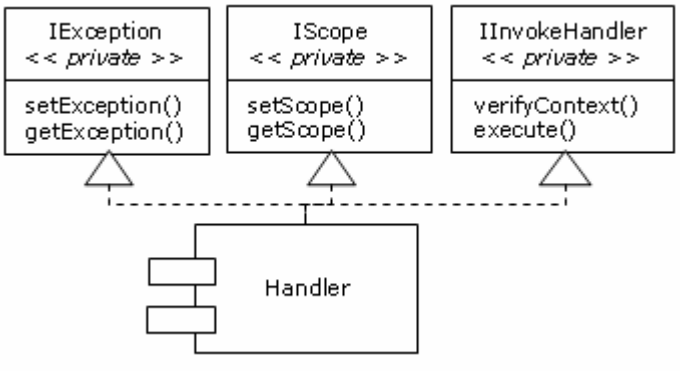

Figura 9. Componente Handler e Interfaces

O componente `Handler` implementa as interfaces privadas (i) `IInvokeHanlder`, (ii) `IException` e (iii) `IScope`. As interfaces `IException` e `IScope` permitem a manutenção das informações sobre a exceção e o escopo que estão associados 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 que forem encontrados.

A Figura 10 apresenta o componente `ExceptionHandlingStrategy` com suas interfaces.

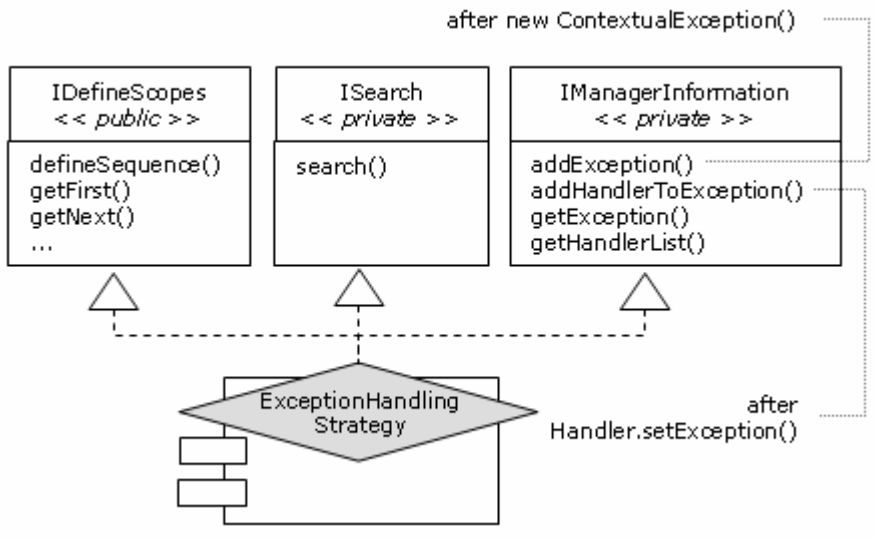

Figura 10. Componente ExceptionHandlingStrategy e Interfaces

O componente `ExceptionHandlingStrategy` implementa a interface pública `IDefineScopes` e as interfaces privadas `ISearcher` e `IManagerInformation`. A interface pública permite que a aplicação especifique a seqüência desejada dos escopos que deve ser utilizada na busca de tratadores. A interface `ISearcher` permite realizar a busca de tratadores. A interface `IManagerInformation` permite gerenciar uma tabela de referências com a relação existente entre uma exceção e seus possíveis tratadores. Isto é feito entrecortando 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.

A Figura 11 apresenta o projeto do componente `PropagationManager` com suas interfaces.

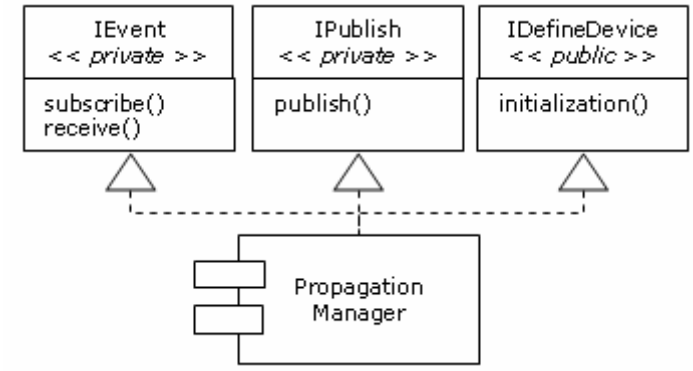

Figura 11. Componente PropagationManager e Interfaces

O componente `PropagationManager` possui duas interfaces privadas `IEvent` e `IPublish`. Estas interfaces são responsáveis, respectivamente, por fornecer eventos com as exceções que foram propagadas por outros dispositivos e permitir a publicação de exceções. Além disso, este componente possui a interface pública `IDefineDevice`, que permite que a aplicação especifique e inicie o papel desempenhado pela aplicação na estrutura de propagação.

A Figura 12 apresenta o projeto do componente `ExceptionPropagation` com suas interfaces.

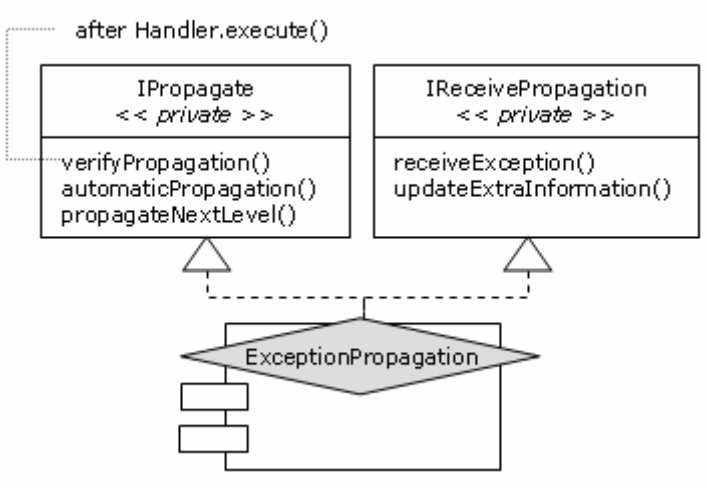

Figura 12. Componente ExceptionPropagation e Interfaces

O componente `ExceptionPropagation` implementa as interfaces privadas `IPropagate` e `IReceivePropagation`. 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 `IReceivePropagation` 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.

5.4. Análise da Arquitetura

Nossa arquitetura de software para tratamento de exceções sensível ao contexto é independente de linguagens de programação ou de *middlewares publish-subscribe*. Isto minimiza a complexidade da tarefa de realizar um efetivo tratamento de exceções sensível ao contexto em aplicações móveis. As diretivas de nossa arquitetura expõem a definição de interfaces, componentes e a interação entre estes que orientam os desenvolvedores no projeto do tratamento de exceções em aplicações móveis. Nossa arquitetura também fornece um contexto no qual decisões de projeto detalhado podem ser adiadas para fases posteriores do processo de desenvolvimento de software. Utilizamos as diretivas de nossa arquitetura durante o desenvolvimento de nosso mecanismo de tratamento de exceções sensível ao contexto (Capítulo 6). Concluímos que a utilização de nossa arquitetura no processo de desenvolvimento de software pode satisfazer os requisitos de sensibilidade ao contexto de aplicações móveis, além de possibilitar o aumento da reusabilidade e da manutenibilidade dos interesses de tratamento de exceções, bem como dos outros interesses nas aplicações.