Tratamento de Exceções no Desenvolvimento de Software Confiável baseado em Componentes

Gisele Rodrigues Mesquita Ferreira
Cecília Mary Fischer Rubira
{gisele.ferreira, cmrubira}@ic.unicamp.br
IC-Unicamp

Resumo

Este trabalho apresenta uma abordagem sistemática para a construção de sistemas confiáveis baseados em componentes de software reutilizáveis e robustos. O tratamento de situações excepcionais do sistema é incorporado de forma disciplinada durante todo o seu ciclo de vida, isto é, durante as fases de identificação de requisitos, análise, projeto e implementação do processo de desenvolvimento. Propomos ainda a confecção de uma ferramenta de suporte ao processo de produção de sistemas confiáveis. Esta ferramenta deverá apoiar as diferentes fases do processo, fornecendo informações sobre cada uma delas e auxiliando na identificação das situações excepcionais.

1. Introdução

Sistemas de software são intrinsecamente complexos e esta complexidade é agravada pelos requisitos de qualidade impostos pelas aplicações modernas como, por exemplo, confiabilidade, desempenho, segurança e disponibilidade. Desta forma, a qualidade, o custo de desenvolvimento e manutenção destes sistemas têm preocupado os desenvolvedores de software. A chave para o sucesso no desenvolvimento de software está no controle da sua complexidade através de reuso de soluções já prontas e amadurecidas e na efetiva utilização de processos de desenvolvimentos voltados para a garantia da confiabilidade dos sistemas.

Hoje em dia, existe muita pesquisa em arquitetura de software, padrões de projeto, *frameworks* e desenvolvimento baseado em componentes. Todas estas técnicas têm como principal objetivo permitir a reutilização de soluções já existentes. A reutilização de componentes tem sido uma das abordagens mais promissoras exatamente pelos benefícios que ela proporciona como, por exemplo, aumento da qualidade do sistema e redução de custos e prazos para o desenvolvimento do software.

Por outro lado, pouca importância tem sido dada aos processos de desenvolvimento de sistemas confiáveis[1], isto é, sistemas que se mantêm disponíveis e funcionando corretamente mesmo na presença de falhas. Podemos citar apenas [Lemos99], [Avizienes97] e [Lemos00] como trabalhos que apresentam guias para os projetistas tratarem a confiabilidade dos sistemas de forma integrada e sistemática durante todo o processo de desenvolvimento de software.

A dependência da sociedade moderna dos sistemas computacionais faz com que falhas nestes sistemas possam resultar em conseqüências desastrosas. Técnicas de tolerância a falhas são empregadas para garantir a confiabilidade, disponibilidade e segurança dos sistemas de computadores. Entretanto, existem particularidades na construção de um sistema onde técnicas de tolerância a falhas são consideradas e uma abordagem bem estruturada para projeto e inclusão destas falhas é um pré-requisito para seu sucesso. Uma abordagem não estruturada pode facilmente reduzir a confiabilidade dos sistemas introduzindo mais falhas do que as previstas [LeeAnderson90].

Tratamento de exceções é uma técnica bem estruturada utilizada para incorporar tolerância a falhas em sistemas computacionais. Esta técnica procura retomar o sistema para um estado livre de erros após a identificação da ocorrência da falha pelo levantamento de uma exceção. Vários mecanismos para tratamento de exceções já foram propostos na literatura mas existem ainda problemas em aplicá-los na prática, tal como a pouca disponibilidade de metodologias e ferramentas adequadas para dar apoio ao emprego apropriado do tratamento da atividade excepcional de um sistema.

[1] do ingles Dependable

O objetivo deste trabalho é a proposta de uma abordagem sistemática para a construção de sistemas baseados em componentes de software reutilizáveis e robustos onde o tratamento de situações excepcionais seja incorporado de forma disciplinada nas diferentes fases de desenvolvimento, isto é, fases de identificação de requisitos, análise, projeto e implementação. Propomos ainda a confecção de uma ferramenta de suporte ao processo de produção de sistemas confiáveis.

O restante deste trabalho está organizado da seguinte forma: na seção 2 apresentamos o modelo de componentes que compõem os sistemas confiáveis. Na seção 3 apresentamos o processo de desenvolvimento de sistemas confiáveis e terminamos com conclusões e trabalhos futuros apresentados na seção 4 .

2. Modelo de Componentes para Construção de Sistemas Confiáveis

Sistemas computacionais são definidos em termos de componentes que podem ser adaptados e conectados para propiciarem a funcionalidade desejada. Existem certas características particulares de sistemas confiáveis que influenciam as suas arquiteturas, como por exemplo, a interação entre os componentes que o compõem.

2.1. Componentes

Os componentes que compõem a aplicação devem ser robustos, isto é, devem incorporar atividades de tratamento de erros de forma a se comportarem adequadamente mesmo na presença de falhas.

Segundo Lee e Anderson [LeeAnderson90], um sistema é definido em termos de componentes que se interagem para fornecerem a funcionalidade desejada. A definição destes sistemas é recursiva, ou seja, um componente é um outro sistema, ou um sub-sistema. O termo componente é empregado em um sentido genérico, isto é, um modelo abstrato que se refere tanto a componentes de software quanto componentes de hardware.

Figura 1 - Componente Ideal Tolerante a Falhas

Ainda seguindo [LeeAnderson90], um componente ideal tolerante a falhas (Figura 1) recebe entradas, processa o serviço, e produz respostas que podem ser de dois tipos: respostas normais e respostas excepcionais. Respostas normais são produzidas quando o componente consegue realizar seu serviço de maneira satisfatória. Respostas excepcionais são produzidas quando ocorreu alguma falha no sistema. Este componente deve manter separado sua atividade que implementa os serviços normais do componente e a atividade implementa os tratadores de situações excepcionais. Esta separação facilita o entendimento, manutenção e adaptação destes componentes. Exceções podem ser levantadas nos componentes do sistema devido a falhas no projeto de seus componentes *(exceções internas)*, pela incapacidade de fornecer um serviço específico *(exceções de defeito)* ou em alguma interação, acidental ou intencional, entre o sistema e seus usuários *(exceções de interface)*.

No modelo do componente ideal, após o levantamento de uma exceção, um mecanismo de tratamento de exceções é responsável pela interrupção do processamento normal do componente. Em seguida, o mecanismo realiza a busca de um tratador de exceção adequado para lidar com a exceção detectada. A busca pelo tratador pode resultar em dois cenários diferentes. No primeiro cenário, o tratador adequado para a exceção é encontrado e o componente será capaz de tratá-la localmente encapsulando a ocorrência da falha em seu interior. No segundo cenário, o componente não estará apto a tratar a falha localmente, ou por falta de um tratador adequado ou por um mau projeto deste tratador. Neste caso, a exceção será propagada para um nível superior da hierarquia de componentes.

2.1.1. Localização dos Tratadores

A manifestação de uma falha pode ocasionar erros no sistema que se não forem tratados adequadamente podem levar a um defeito. A manifestação de uma mesma falha pode ocasionar erros diferentes e necessitar de tratadores apropriados dependendo do contexto de ocorrência. Erros podem estar associados a uma instância de classe, a um método, a uma classe, a uma exceção ou a um subsistema. Desta forma, os tratadores das exceções devem estar associados ao contexto específico de acordo com a necessidade.

2.1.2. Propagação de Exceções

A propagação das falhas pode ser feita de forma automática ou explicita [Garcia99]. No primeiro caso, se nenhum tratador para a exceção for encontrado em quem a recebeu, esta exceção pode ser propagada automaticamente para o nível superior da hierarquia de componentes até que um tratador seja encontrado. No segundo caso, o componente deve manter um tratador que apenas sinaliza a exceção para hierarquia superior. O fato de existir tratadores para todas as exceções, mesmo que ele seja usado apenas para propagar a exceção, é útil para a identificação do caminho que esta exceção percorreu e conseqüentemente identificação do contexto de manifestação da falha e escolha do tratador adequado. Além disto, esta informação é muito útil para identificação dos componentes que provavelmente foram afetados pelo erro e que necessitam ser reparados.

2.2. Redundância de Componentes

Tolerância a falhas se baseia em detecção de erros, recuperação do sistema para um estado livre de erros e redundância de elementos do sistema, isto é, componentes. A estratégia de replicação de componentes visa aumentar a disponibilidade e qualidade do serviço fornecido. Replicação pode ser tanto de componentes de hardware quanto de software. Se os componentes forem replicados e distribuídos, de forma que falhas física afetem os elementos independentemente, a disponibilidade do sistema estaria garantida. Entretanto, se o sistema possuir falhas de projeto, esta se manifestará em todas as cópias do sistema, e neste caso, é importante diversidade de projeto dos elementos replicados para aumento da probabilidade de que falhas não ocorram simultaneamente [Avizienes97].

3. Um Processo de Desenvolvimento de Software para Sistemas Confiáveis

Esta seção apresenta um processo para apoiar a construção de sistemas confiáveis. O processo aqui apresentado é uma extensão do Processo Unificado [Jacobson99] que se caracteriza por ser dirigido por casos de uso, iterativo, incremental e centrado na arquitetura. O desenvolvimento é visto como uma série de iterações que abrangem todo o sistema e cada iteração consiste de levantamento dos requisitos, análise, projeto, implementação e testes. A definição dos casos de uso, na fase de levantamento de requisitos, serve como guia para a realização das fases subseqüentes. Esta abordagem proporciona um relacionamento explícito entre os elementos dos modelos do sistema facilitando seu entendimento e manutenção. No processo proposto, os requisitos funcionais e os requisitos de qualidade de um sistema devem ser definidos inicialmente e influenciarão todo o ciclo de vida deste sistema.

A garantia de confiabilidade e disponibilidade de um sistema é um dos requisitos de qualidade que tradicionalmente é considerado apenas na fase de implementação. No entanto, um processo de desenvolvimento de sistemas confiáveis deve manter a preocupação com a tolerância a falhas paralelamente à definição da funcionalidade deste. O objetivo é a identificação da atividade excepcional de um sistema o mais cedo possível de forma a obter melhores resultados, distribuindo os esforços de garantia de confiabilidade durante todo o processo e tornando o trabalho de confecção destes sistemas mais garantido [Avizienes97], [LeeAnderson90], [Lemos00].

3.1. Identificação de Requisitos

Durante esta etapa, os requisitos devem ser levantados a nível tão detalhado quanto necessário para que o cliente, usuário e desenvolvedores entrem em um acordo quanto a funcionalidade disponibilizada. Uma definição precisa destes requisitos é um passo essencial para o desenvolvimento de um produto de qualidade. Também faz parte desta etapa a identificação dos requisitos de qualidade como desempenho, portabilidade, manutenibilidade e confiabilidade. As atividades destas etapa estão apresentadas na **Figura 2.**

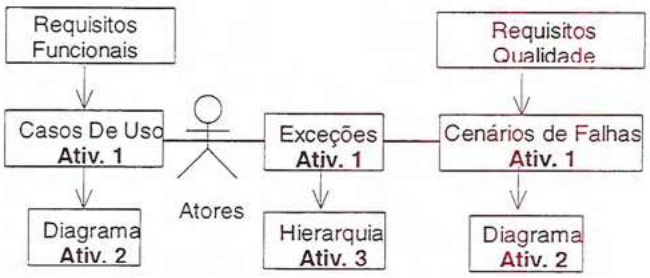

Figura 2 - Atividades da Identificação de Requisitos

A definição dos requisitos funcionais se dá a partir da criação de cenário de casos de uso que descrevem a atividade básica do sistema. Seguindo o modelo do componente ideal (seção 2.1), toda requisição de serviço devolve respostas que podem ser normais ou excepcionais. Portanto, para cada caso de uso, elabore uma tabela contendo sua descrição detalhada, os atores envolvidos, os cenários de falhas e possíveis exceções levantadas pela manifestação de falhas (**Ativ. 1**). Os diagramas de casos de uso, que mostram o relacionamento entre cada cenário, deverão incorporar os cenários de falhas identificados até o momento (**Ativ. 2**). As exceções devem ser representadas como classes e organizadas em forma de hierarquia de exceções(**Ativ. 3**). Através da hierarquia de exceções pode-se definir tratadores genéricos ou tratadores específicos. Tratadores genéricos se aplicam a classes bases, e conseqüentemente às suas subclasses, enquanto que tratadores específicos são destinados apenas às exceções de classes derivadas.

3.2. Fase de Análise

Na fase de análise, os desenvolvedores se deparam com o desafio de identificar os subsistemas de uma aplicação, objetivando a obtenção de um modelo bem estruturado, que seja de fácil entendimento, modificação e adaptação. As situações excepcionais e as possíveis soluções descritas na etapa anterior devem ser consideradas durante a modelagem e novas situações de falhas e exceções serão incorporadas. As atividades desta etapa estão apresentadas na **Figura 3**.

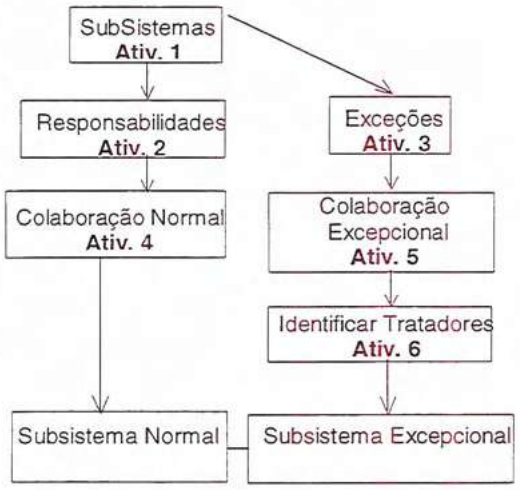

Figura 3 - Atividades da Análise e Projeto

A atividade inicial da análise é o desenho da estrutura principal do produto em subsistemas (**Ativ. 1**). Para cada caso de uso, definidos na identificação de requisitos, existem cenários de falhas associados. Os casos de uso auxiliam na identificação das responsabilidades de cada subsistema (**Ativ. 2**) e os cenários de falhas auxiliam na definição das exceções que este subsistema possa sinalizar (**Ativ. 3**). Baseado na interface de cada módulo, o analista deve definir a colaboração das atividades normais (**Ativ 4**) e baseado nas exceções que cada módulo sinaliza, é importante a construção de um modelo de falhas que represente a propagação das exceções entre os subsistemas (**Ativ. 5**). Este modelo de falhas auxilia na identificação dos tratadores que cada subsistema deve incorporar(**Ativ. 6**). De acordo com o modelo de propagação de exceções (seção 2.1.2) e localização dos tratadores (seção 2.1.1) deve existir um tratador para toda falha que afete o subsistema, mesmo que este tratador só sinalize a exceção.

Em suma, a fase de análise consiste na identificação dos subsistemas que mantêm explícita a separação de suas partes normais e excepcionais estruturadas em duas hierarquias ortogonais: (i) hierarquia de subsistemas normais, que incorporam as atividades funcionais, (ii) hierarquia de subsistemas excepcionais, que incorporam os tratadores das exceções levantadas por este subsistema. Esta hierarquia de subsistemas é o primeiro esboço da arquitetura do sistema seguindo o modelo do componente ideal.

3.3. Fases de Projeto e Implementação

A fase de projeto adiciona decisões de projeto para a associação dos subsistemas que levantam exceções (normais) e os que possuem tratadores para estas exceções (excepcionais). Nessa fase, o objetivo é identificação de componentes que possam ser reutilizáveis e a integração transparente dos módulos normais e excepcionais de um componente. As atividades de projeto estão apresentadas na **Figura 4.**

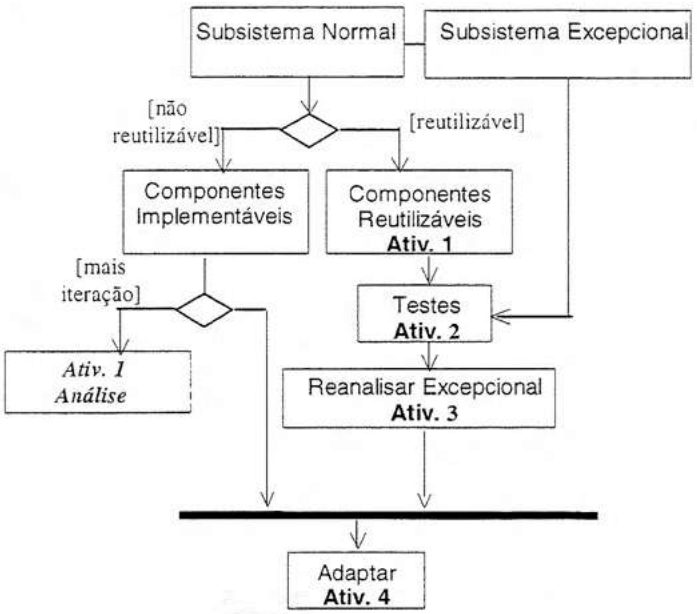

Figura 4 - Atividades de Projeto

O modelo de subsistemas normais e excepcionais gerados na fase de análise irão guiar todas as atividades do projeto. Olhando para os módulos que implementam as funcionalidades dos subsistema (subsistema normal) o projetista pode identificar componentes de software que foram outrora implementados ou projetados e que podem ser reutilizados **(Ativ. 1)**. Os módulos que definem a parte excepcional deste componente reutilizável (subsistema excepcional) auxiliam na elaboração de teste que deverão ser feitos sobre os componentes para avaliar se estes implementam o comportamento excepcional identificado até o momento **(Ativ. 2).** Pode ser necessário remodelar a atividade excepcional do componente caso ele não trate algumas das situações excepcionais previstas **(Ativ. 3)**. Os subsistemas que não serão reutilizados, devem ser refinados até a obtenção de um modelo a nível de classes com seus atributos e operações. Deve-se refinar inclusive os tratadores das exceções, definindo-os a nível de classes, de método ou a nível de instância de classe (objetos), como apresentado na seção 2.1.1.

A separação explícita entre a atividade excepcional e normal de um componente, seguindo o padrão do componente ideal, facilita a reutilização da funcionalidade do componente. Entretanto, os mecanismos de exceções das linguagens de programação usualmente não oferecem suporte direto para a separação entre o código normal e código de tratamento de erros. O código de tratadores é freqüentemente embutido no código normal, tornando complexa a semântica dos componentes da aplicação e menos propenso à reutilização. A atividade de adaptação de componentes (**Ativ. 4**) consiste em associar as classes normais e excepcionais sempre mantendo a separação explícita entre elas. Entretanto, se o mecanismo de tratamento de exceções disponível não permitir esta separação, dois padrões de projeto apresentam soluções bem estruturadas para este problema:

1. O padrão *Meta-Handler* apresentado em [Garcia99] define uma arquitetura de meta-nível que permite que exceções sinalizadas pelos componentes da aplicação, situados no nível base, sejam interceptadas pelo protocolo de meta-objetos e associadas aos seus tratadores.
2. Criação de uma nova classe que realize a composição das atividades normais e anormais de um componente. Considere o seguinte exemplo ilustrado da **Figura 5**: seja BN a classe que implementa a atividade normal e BA a classe que implementa os tratadores das exceções levantadas em BN. Defina uma nova classe B, que realize a composição de BN com BA através da redefinição da interface publica

de BN, associando a chamada dos métodos de BN com os tratadores definidos em BA. As chamadas aos métodos de B são delegadas para serem efetivamente realizadas por BN.

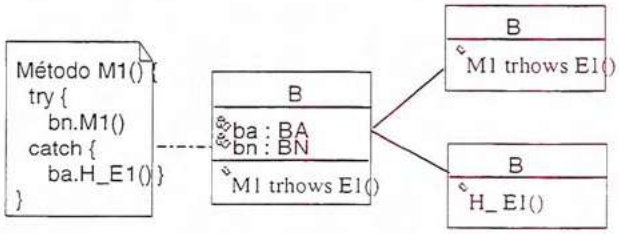

Figura 5 - Padrão para Associação de Classes Normais e Anormais

Durante o projeto, pode-se ainda aprimorar a disponibilidade e qualidade do serviço através de redundância e diversidade de subsistemas e de canais de comunicação e distribuição física dos módulos (seção 2.2).

Na fase de implementação, o projeto detalhado é convertido em código executável das linguagens de implementação. Novas exceções serão identificadas. Estas exceções estarão relacionadas com a implementação, como por exemplo, a linguagem escolhida e estruturas de dados específicas, e com o ambiente aonde a aplicação estará executando, como por exemplo, o sistema operacional e o mecanismo de tratamento de exceções disponível.

4. Conclusões e Trabalhos Futuros

Neste trabalho apresentamos: (i) uma arquitetura de sistemas confiáveis baseado no modelo do componente ideal e na redundância de componentes de sistema; (ii) um processo para a construção de sistemas confiáveis. Neste processo, as situações de falhas e exceções são definidas na fase de identificação de requisitos e auxiliam todo o ciclo de desenvolvimento do sistema . Os tratadores de exceções de uma classe são mantidos em classe separada para facilitar a manutenção e reuso dos componentes.

Este trabalho está em andamento, e realizaremos ainda um refinamento de todo o processo apresentado na seção 3. Além disto, vislumbramos a construção de uma ferramenta de gerência de processo que controle o desenvolvimento de sistemas através do definição de cargos, atribuição de responsabilidades, atribuição e controle de execução de tarefas e fluxo dos documentos. O desenvolvimento de uma ferramenta de apoio à utilização e gerência informatizada do processo propicia agilidade na utilização do processo, aumento da qualidade do sistema gerado, auxílio na personalização do processo e garantia de execução de atividades essenciais para manutenção da qualidade final do sistema.

5. Referências

[Avizienes97] Avizienes, A. "*Toward Systematic Design of Fault-Tolerant Systems*". IEEE Computer - 1997

[Garcia99] Garcia, A.and Beder, D and Rubira, C *"An exception handling mechanism for developing dependable object-oriented software based on meta-level approach"* 10th Symposium on Software Reliability Engineering IEEE 1999

[Jacobson99] Jacobson, I. and Rumbaugh, J. and Booch, G. "*Unified Software Development Process*" Addison-Wesley, Reading -MA, 1999.

[LeeAnderson90] Lee and Anderson "*Fault-Tolerance: Principles and Practice*". Springer-Verlag 2ª Edição, Jan-1990

[Lemos99] Lemos, R. and Romanovsky, A. *"Exception Handling in a Cooperative Objetct-Oriented Approach"* Proceedings of the 2nd IEEE International Symposium of Object-Oriented Real-Time Distributed Computing, ISOCR'99, França,Maio'99

[Lemos00] Lemos, R. and Romanovsky, A. *"Exception Handling in the Software Lifecycle"* Int. Journal of Computer Systems Science and Engineering (Special Issue on Object-Oriented Real-Time Distributed Systems) (a ser publicado)