1
Introdução

Quando o programador ou analista não participou da fase de projeto e desenvolvimento de um sistema, ele não tem o conhecimento geral do seu comportamento, de seus componentes e da interação entre eles. Nesse caso a manutenção do sistema e o desenvolvimento de novas funcionalidades se tornam uma tarefa mais difícil. Para auxiliá-lo durante a manutenção e o desenvolvimento, o programador necessita de um conjunto de ferramentas que permitam analisar o comportamento do sistema em tempo de execução, para ajudá-lo a diagnosticar a origem e a razão de uma falha ocorrida.

O problema do entendimento do comportamento de um sistema se agrava ainda mais quanto este é um sistema distribuído. Isso porque descobrir as causas de um comportamento indevido nesse tipo de sistema pode ser uma tarefa difícil, já que a ordem de ocorrência dos eventos não é, necessariamente, determinística. Essa ordenação pode sofrer influência de vários fatores. Um deles é a variação na qualidade do serviço provido pela rede, que vai alterar de forma aleatória e independente o tempo de transmissão dos eventos entre os diversos componentes do sistema distribuído. Outro fator é a variação da carga de processamento da máquina, que influencia o tempo de execução, e assim, uma linha de execução (*thread*) pode levar mais ou menos tempo para atingir o ponto de execução na qual um dado evento é enviado. Como resultado, é difícil garantir que um componente receberá ou fará requisições remotas sempre em uma determinada ordem.

A cenário apesentado acima gera uma complicação a mais na análise do sistema, já que a reprodução de um cenário que ocasionou um dado comportamento indevido no sistema pode se tornar uma tarefa muito complexa [1, 2], o que dificulta a obtenção de informações necessárias para entender, identificar e diagnosticar o problema. Em muitos casos, os problemas e *bugs* são sutis, intermitentes e esporádicos, causados por condições de corrida ou por falta de sincronização entre os processos.

Os *logs* de execução do sistema fornecem informações relevantes para que o desenvolvedor possa tentar localizar a origem e a razão de uma falha. Porém, os *logs* são geralmente muito extensos, o que dificulta sua manipulação

e análise manual para extrair informações úteis e visualizar as relações de causalidade entre os eventos. Nesse sentido, há diversos trabalhos [3] que investigam ferramentas para auxiliar o programador na tarefa de entender os traços de execução de um sistema.

Ao mesmo tempo, outros trabalhos [4, 5, 6, 7] propõem ferramentas de depuração distribuída que seguem a mesma filosofia dos depuradores simbólicos tradicionais. Essas ferramentas têm a vantagem de fazer o acompanhamento das linhas de execução distribuídas levando em conta a abstração conceitual oferecida pela ferramenta de programação. No modo passo a passo desses depuradores, a chamada de um método de um *proxy* leva diretamente ao código da função chamada e não ao código do *middleware*. Assim, o programador não é confrontado com esse código que ele não quer ou não precisa entender e que pode até mesmo não estar disponível. Por exemplo, Giuliano [5] propôs uma arquitetura para um depurador distribuído que utiliza a GUI extensível do Eclipse.

Entendemos que essa abordagem, embora vantajosa, porque mantém a mesma filosofia usada para depurar sistemas centralizados, tem algumas limitações. Por exemplo, ela elimina a concorrência e não permite a identificação de erros ou comportamentos inesperados que advêm dessa inerente característica dos sistemas distribuídos, que está intimamente ligada a sua natureza não determinística. Outros trabalhos [8] também salientam o importante papel da concorrência na depuração e análise do comportamento de um sistema distribuído.

Entender o comportamento dos sistemas distribuídos ainda é um desafio e a implementação e manutenção de tais sistemas ainda são tidas como tarefas complexas. Schwarz e Mattern [8] chegam, inclusive, a dizer que a implementação desse tipo de sistema *"ainda é uma arte e não uma questão de engenharia"*.

1.1 Objetivos e contribuições

O objetivo deste trabalho é permitir que o desenvolvedor possa analisar as sequências de interações entre os componentes, as relações de causalidade e os dados de desempenho. Essa análise pode ser feita dinamicamente durante a execução do sistema, ou *post mortem*, através de consultas realizadas nas informações armazenadas em uma base de dados. Dessa forma, o programador pode realizar consultas para auxiliá-lo a identificar a causa de um erro ou comportamento indevido do sistema, de forma que ele possa restringir pouco a pouco o escopo da busca.

De acordo com Schwarz e Mattern [8], a importância da análise das sequências de interações entre componentes se deve ao fato de que um melhor entendimento do comportamento de um sistema distribuído pode ser obtido a partir da análise das relações causais e temporais dos eventos.

A abordagem utilizada neste trabalho para determinar essas relações é a de instrumentar o sistema para coletar informações durante a sua execução e, então, submetê-las a um processo de análise a fim de recuperar a sequência de invocações remotas. No escopo deste trabalho, assumimos que os processos do sistema se comunicam utilizando chamada remota de procedimento (RPC) e que as relações de causalidade estão limitadas a relação entre funções invocadas/invocadoras.

A partir da análise da relação causal entre as chamadas remotas, associadas com as informações sobre a arquitetura do sistema e com dados de desempenho e concorrência, o desenvolvedor pode entender melhor o comportamento do sistema distribuído e o cenário que envolve um dado comportamento inesperado ou uma falha. Assim, ele pode restringir pouco a pouco o escopo da análise do erro e diagnosticar mais facilmente as causas do problema.

Neste trabalho desenvolvemos uma ferramenta para recuperar a relação de causalidade das interações entre os componentes de um sistema distribuído e exibir as sequências de chamadas remotas, correlacionado-as com os componentes do sistema e com dados de desempenho. De acordo com Jerding et al. [9], a visualização das interações entre os objetos de uma aplicação constitui uma forma de analisar e compreender o seu comportamento dinâmico. A identificação de padrões recorrentes de interações, que podem ser usados como abstrações para o processo de compreensão, é importante também para auxiliar tarefas de re-engenharia de Software.

As informações utilizadas pela nossa ferramenta são coletadas durante a execução e então processadas e organizadas em um modelo de dados para serem persistidas ao longo do tempo em um banco de dados relacional, permitindo futuras consultas sobre diversos aspectos do sistema.

Para o desenvolvimento dessa ferramenta criamos um mecanismo para acompanhamento das chamadas remotas que coleta dados necessários para estabelecer a relação de causalidade entre as chamadas. No desenvolvimento desse mecanismo foi necessário a utilização de um esquema para permitir a identificação da relação causa-efeito entre as invocações e para permitir o suporte a requisições assíncronas feitas em múltiplas linhas de execução.

Neste trabalho, as principais contribuições foram: um mecanismo para acompanhamento das sequências de chamadas remotas realizadas pelos componentes de um aplicação que usam o mecanismo de RPC (*Remote Proce-*

dure Call) de CORBA [10]; um modelo de dados que permite correlacionar as chamadas remotas com informações sobre a topologia da aplicação e com informações de desempenho, além de permitir o armazenamento das sequências de chamadas para futuras consultas; a extensão da infraestrutura de monitoramento utilizada para permitir o acompanhamento das sequências de chamadas remotas, implementando o mecanismo desenvolvido.

Procuramos avaliar a ferramenta desenvolvida na análise de uma aplicação, para que pudéssemos verificar o quanto o nosso modelo de dados é eficaz para responder questões elaboradas por desenvolvedores a respeito da execução dessa aplicação. Essas respostas podem nortear o processo de depuração em caso de ocorrência de um problema. Procuramos ainda mensurar o impacto que a nossa ferramenta causa no desempenho da aplicação em análise, tendo em vista a necessidade de instrumentação dessa aplicação.

1.2 Estrutura do documento

No capítulo 2 discutiremos alguns trabalhos relacionados enfatizando como cada um se relaciona com este trabalho. No capítulo 3 apresentaremos o modelo de componentes SCS e suas infra-estruturas de execução e monitoramento. No capitulo 4 apresentamos a definição de transação e descrevemos a ferramenta e o mecanismo utilizado para fazer o acompanhamento de uma transação. No capítulo 5 apresentaremos um experimento da nossa ferramenta em um sistema de testes desenvolvido para reproduzir as interações de uma aplicação real baseada em componentes. Fazemos, também, uma avaliação da nossa ferramenta na questão do impacto causado no desempenho a aplicação analisada. Por fim, no capítulo 6 apresentamos nossas considerações finais e discutimos possíveis extensões deste trabalho.