7
Conclusões

Nesta dissertação apresentamos uma arquitetura de monitoração do ambiente de execução de componentes distribuídos, e também realizamos experimentos com o objetivo de verificar a aplicabilidade da solução. Embora questões de instrumentação de sistemas de componentes distribuídos não seja um tema novo, são poucas as aplicações e *middlewares* que fornecem suporte a esse tipo de funcionalidade. Dessa forma, existe uma demanda para que os trabalhos relacionados com esse tema desenvolvam-se visando atender a essas necessidades.

Nessa seção apresentamos uma avaliação do trabalho realizado durante o desenvolvimento desta dissertação. Também são apresentados trabalhos futuros que podem ser realizados a partir dos resultados obtidos.

7.1
Avaliação do Estudo

Sistemas distribuídos baseados em componentes de software ao evoluirem em tamanho e complexidade tornam-se difíceis de administrar. Isso ocorre devido ao envolvimento de diversas máquinas, versões de componentes com comportamentos distintos e usuários. Monitorar o comportamento das diversas máquinas, e as relações existentes entre os diversos componentes que compõem a aplicação, é uma das formas desses sistemas continuarem a evoluir de forma controlada.

Neste trabalho estudamos formas de monitorar esse tipo de aplicação e apresentamos uma solução que se propõe a auxiliar a tarefa de administração desses ambientes. Uma das principais características que buscamos foi montar uma arquitetura transparente para o desenvolvedor dos componentes e que

impusesse uma baixa sobrecarga ao sistema monitorado.

Ao escolher CORBA como *middleware* para implementação dos componentes encontramos algumas soluções possíveis para a monitoração das relações entre os componentes. Uma delas seria a inserção de código nos artefatos gerados pelo *middleware* usado. Essa solução não se mostrou adequada pois podemos lidar com diversas implementações do padrão CORBA, e o código gerado por elas não segue um padrão, o que nos levaria a diferentes abordagens de acordo com o *middleware* que esteja sendo utilizado. Assim, optamos pelo uso dos interceptadores (seção 5.3.1) por se tratar de um recurso definido pela arquitetura CORBA e ao qual vários fabricantes oferecem suporte. No entanto a documentação encontrada sobre o uso de interceptadores se mostrou escassa, o que dificultou seu uso inicialmente.

Independente das dificuldades com documentação, os interceptadores se mostraram um poderoso recurso para auxiliar o processo de monitoração, devido à transparência oferecida ao desenvolvedor e aos diversos pontos de interceptação que este mecanismo disponibiliza, permitindo a coleta de informações relevantes para o administrador da aplicação.

Além da coleta de informações sobre a interação entre os componentes, realizadas pelos interceptadores, sentimos necessidade de coletar informações sobre a carga de cada *container*. Para isso utilizamos um *daemon*, seção 5.3, disparado pelo *container*. No entanto, as características do processo que devem ser coletadas são dependentes do tipo de aplicação que está sendo monitorada, o que nos levou a construir um mecanismo extensível que possibilitasse ao administrador acrescentar métricas dinamicamente na arquitetura.

Durante a construção de um mecanismo de monitoração, a forma de publicação das informações se mostrou um outro desafio. Manter as informações em cada nó poderia gerar um alto custo para armazená-las e em seguida para coletá-las. A solução encontrada foi retirar as informações do nó imediatamente após a coleta, de forma que clientes interessados nessas informações possam recebê-las e tratá-las em nós que estejam fora do ambiente da aplicação. Canais de eventos foram escolhidos para essa tarefa, seção 5.4.

Monitorar o ambiente de execução do sistema possui uma importante relação entre acurácia das informações coletadas e a sobrecarga imposta. Dessa forma, para obtermos as informações exatas temos que realizar coletas e publicações em intervalos pequenos de tempo, o que irá aumentar a sobrecarga sobre a aplicação. Contornamos esse problema montando uma arquitetura flexível onde o usuário pode ajustar o tempo entre as coletas, e construir

políticas definindo o momento em que os dados devem ser publicados.

No capítulo 6, os experimentos realizados ajudaram a verificação da aplicabilidade da solução implementada e a correção de *bugs* tanto na arquitetura de monitoração como no SCS. Esse capítulo também reforçou a importância do mecanismo de monitoração possuir uma forma flexível de configuração, permitindo que o administrador do sistema possa customizar o mecanismo de acordo com as características que estão sendo monitoradas.

7.2 Trabalhos Futuros

Após a finalização desta dissertação identificamos alguns pontos que podem ser abordados em novos trabalhos, que agregariam valor a arquitetura construída. Essa seção irá tratar alguns pontos que foram identificados.

Atualmente somente as métricas coletadas a partir do *daemon* do *container* podem ser estendidas. Logo, fornecer um mecanismo que permita ao administrador do sistema adicionar métricas para serem coletadas pelos interceptadores tornaria a arquitetura ainda mais flexível. Porém, adicionar métricas nos interceptadores envolve uma estrutura mais cuidadosa pois o administrador deverá ter uma forma de informar qual métrica coletar, em quais pontos de interceptação e caso exista um relacionamento entre a coleta realizada em diferentes pontos, como correlacionar as informações obtidas.

Iniciamos a implementação do mecanismo de coleta para componentes SCS/Lua. A implementação realizada verificou que todos os recursos implementados em Java podem ser implementados em Lua, que atualmente já possui uma versão do *container* SCS. Terminar essa implementação pode permitir comparações entre o uso da instrumentação em diferentes linguagens, assim como permitir a monitoração de aplicações com componentes heterogêneos.

As políticas definidas neste trabalho são relacionadas somente à publicação de métricas. Seria interessante montar um mecanismo similar para definição do tempo de espera entre a coleta das informações. Assim o administrador poderia fornecer uma estratégia que definisse dinamicamente a periodicidade da coleta de métricas.

Na mesma linha das estratégias de agrupamento de informações proposta em [1], poderiam ser criadas estratégias para o agrupamento das informações

publicadas, que é realizado no *StatsCollection* (seção 5.4.2), que irão fornecer métricas mais apropriadas aos clientes.

Para aumentar o número de informações sobre a interação entre os componentes pode ser montado um mecanismo que permita a inspeção dos parâmetros passados durante as requisições. Isso poderia ser feito com o uso de *proxys* dinâmicos defidos pela arquitetura CORBA(DII e DSI) ou utilizando a abordagem de *proxy* proposta em [4].

Para tornar mais simples o mecanismo de extensão das métricas e políticas, os métodos responsáveis por essas atividades poderiam ser preparados para receber diretamente o código que deve ser executado. O administrador ficaria responsável somente por construir as métricas e não por instalá-las.

A construção de um mecanismo para controlar o fluxo completo da uma determinada chamada, indicando todos os componentes que foram invocados a partir de uma requisição até que a resposta seja retornada ao cliente; e uma forma de manter informações sobre o usuário que está acessando cada serviço, seriam de grande valia para o administrador do sistema.

Por fim seria interessante construir uma ferramenta gráfica capaz de realizar o processo de configuração dos mecanismos de instrumentação, e que também possibilitasse ao administrador realizar uma auditoria dos serviços utilizados. A ferramenta monitora apresentada na seção 5.4.3 pode ser usada como ponto de partida para essa implementação.