1
Introdução

Há alguns anos podemos observar um crescimento no número de estudos e projetos relativos a infra-estruturas de apoio à execução de aplicações distribuídas, e a barramentos comuns para a integração entre aplicações [1, 2, 3]. Isto deve-se ao fato de que cada vez mais necessitamos da integração entre diferentes plataformas, principalmente em sistemas complexos e de larga escala. Vemos esta necessidade em outros casos como em situações de integração de aplicações já existentes, onde não houve uma preocupação durante sua concepção em torná-las capazes de interagir entre si posteriormente. Devido a essa necessidade de integração, podemos observar também uma mudança na forma de desenvolvimento de aplicações distribuídas, que agora frequentemente são implementadas na forma de serviços que contam com interfaces bem definidas. Esta definição formal permite a interação com outros serviços ou aplicações sem que ambos tenham conhecimento prévio de suas existências.

Dentre os desafios primários para o desenvolvimento de uma infra-estrutura de execução de aplicações ou serviços distribuídos, podemos citar a instalação, a configuração e a execução desses serviços nas diversas máquinas do domínio distribuído. Realizar o trabalho manualmente pode ser inviável, tanto por questões de tempo como de sincronização. Pode-se criar *scripts* de configuração e execução, mas estes tendem a tornar-se excessivamente complexos e numerosos. Outra opção é criar um serviço adicional que realize este tipo de trabalho a partir de descrições de mais alto nível da configuração de uma aplicação distribuída. Ainda assim, mesmo esse serviço de mais alto nível precisa do suporte de serviços mais básicos que permitam ao menos a instalação e execução remotas de outros serviços.

Tipicamente, sistemas distribuídos mais complexos envolvem serviços implementados em diferentes linguagens. Por conta disso, uma infra-estrutura para a execução de tais sistemas deve permitir a configuração de propriedades específicas a determinadas linguagens. Por exemplo, para linguagens baseadas

em máquinas virtuais como Java ou Lua, os serviços podem contar com requisitos específicos relativos à configuração da máquina virtual na qual estará executando, por motivos de desempenho ou mesmo compatibilidade com diferentes plataformas. Para atender a estes requisitos, são necessárias ferramentas adicionais que permitam a criação de um ambiente que seja configurado previamente à carga do serviço.

Frequentemente, aplicações distribuídas se dividem em serviços menores que colaboram entre si e possivelmente ocupam diferentes recursos do ambiente distribuído, geralmente em diferentes nós da rede. Esta colaboração entre os serviços faz com que uns se tornem dependentes dos outros para seu funcionamento. Assim, além de terem suas funcionalidades providas bem definidas através de suas interfaces, é importante que tenham bem definidas também suas funcionalidades requeridas, que na verdade são suas dependências de *software*.

Uma técnica que vem sendo bastante utilizada em modelos de comunicação distribuída é o paradigma de programação orientada a componentes [4, 5]. Este paradigma prega a extensão do conceito de modularidade [6]: a divisão e construção em blocos pequenos, mas também bem definidos e conhecidos, com interfaces providas e requeridas bem especificadas. Um exemplo de tecnologia baseada em componentes para sistemas distribuídos é o Modelo de Componentes CORBA [7], que atua como uma evolução do modelo de objetos distribuídos precedente. *Middlewares* baseados em componentes geralmente fornecem também ferramentas que embutem outras facilidades, como suporte nativo à comunicação distribuída e independência de linguagem.

Este paradigma de programação traz ainda outras vantagens. A maioria dos *softwares* atuais são fortemente acoplados e permitem apenas que certos módulos, geralmente bibliotecas, sejam carregados em tempo de *link*-edição ou de carga da aplicação. Para substituir um módulo de uma aplicação em execução, é necessário acabar com o processo, reiniciá-lo e carregar outra biblioteca, levando à falta de disponibilidade do *software*, que pode ser de uso crítico. Também pode ser necessário modificar e recompilar o *software* antes de reiniciá-lo, o que demanda tempo, esforço, e possivelmente a presença de um desenvolvedor ou administrador. É desejável, portanto, uma granularidade menor e um controle maior nestes aspectos para se obter uma maior flexibilidade. Aplicações baseadas em componentes de *software* geralmente possibilitam este maior controle pois atingem um baixo grau de acoplamento, devido à especificação de suas interfaces providas e requeridas. Isto torna possível a configuração e reconfiguração dinâmica das dependências dos componentes.

Para o suporte à configuração e reconfiguração dinâmica de dependências, componentes devem poder ser facilmente substituídos. Idealmente, isto pode ocorrer a qualquer momento e se possível em tempo de execução [8]. Para projetos grandes, complexos, possivelmente distribuídos e críticos, este requisito se mostra fundamental [9]. A granularidade pode variar de acordo com a necessidade da aplicação, desde que cada componente mantenha uma identidade e propósito bem definidos.

No entanto, o suporte a este tipo de facilidade torna-se mais um desafio para infra-estruturas de execução, assim como o suporte à inspeção das interfaces providas e requeridas pelos componentes. Este tipo de inspeção é essencial para facilitar a descoberta de serviços e recursos no ambiente distribuído. Outros desafios existem, alguns não sendo tão fáceis de se prever inicialmente. Um exemplo é a possível necessidade de suspensão das comunicações para pequenas manutenções ou mudanças na configuração de componentes. Outro exemplo é a configuração dos próprios serviços e facilidades oferecidas pela infra-estrutura, que podem exigir tratamentos adicionais em certos ambientes ou plataformas, ou ainda novas funcionalidades que precisem ser adicionadas sem terminar a execução. Para facilitar a solução destes desafios, pode-se prover o apoio a outros paradigmas de programação, como a programação orientada a aspectos [10, 11, 12, 13, 14].

Atualmente, diversos sistemas com tais funcionalidades foram desenvolvidos e alguns até mesmo usados amplamente, apesar de nenhum ter emergido como padrão. Foram propostas diversas soluções e modelos [15, 16, 9, 17, 1, 2, 3, 14, 18, 19, 20] com bons resultados para a maioria dos problemas comuns. Alguns *middlewares*, além do suporte a componentes, oferecem serviços ligados à execução, localização, segurança e persistência. Para nosso estudo, temos especial interesse em prover serviços ligados à execução de componentes, procurando tratar dos desafios mencionados anteriormente e ainda alguns relacionados à implantação de serviços. No escopo deste trabalho, nos referiremos ao processo de implantação como as ações necessárias para que um componente ou serviço seja instalado em um nó, possa ser carregado e possa ter suas dependências externas supridas.

Existem diversos cenários de uso possíveis para uma infra-estrutura de execução de componentes de *software*. Um dos cenários mais interessantes para nós é o de criação de arcabouços de teste [21] para aplicações de grade ou distribuídas em geral. Normalmente, aplicações distribuídas são difíceis e custosas de se testar, pois exigem esforços como a replicação da aplicação por todos os nós, a geração de *scripts* para a inicialização de cada nó, código extra

para a comunicação, entre outros detalhes [22, 23, 24]. Sem uma estrutura de apoio, com facilidades como a independência de linguagens, a reflexão computacional, a (re)configuração dinâmica de dependências e serviços para facilitar a implantação e execução, aumenta-se o custo de desenvolvimento e o trabalho extra pode potencialmente gerar mais problemas do que ajudar a resolver os existentes.

Para investigar problemas de execução de componentes em um ambiente distribuído e possibilitar futuras novas abordagens a esses diversos cenários, estendemos a infra-estrutura de serviços do projeto OpenBus, integrando serviços de apoio à execução. O OpenBus é um projeto desenvolvido pelo TecGraf[25] / PUC-Rio[26] em parceria com a Petrobrás, que atua no cenário de integração de aplicações científicas. O projeto tem como objetivo atenuar ou resolver o problema de integração entre estas aplicações. Através de um barramento comum, aplicações podem se comunicar e interagir de forma mais fácil pois passa a existir um padrão para as interações. Muitas dessas aplicações são sistemas legados que não previram a necessidade de interações posteriores, não sendo assim capazes de se comunicar entre si. Por isso, passam primeiramente por um processo de definição de suas interfaces providas e requeridas. Posteriormente, são transformadas em componentes e passam a utilizar os serviços fornecidos pelo OpenBus para ajudar a resolver esses problemas de integração. O detalhamento de alguns dos serviços já existentes no OpenBus e da modelagem por componentes utilizada será descrito no Capítulo 3.

Algumas ferramentas servirão de base para a implementação dessa infra-estrutura de execução. Tomamos de base as mesmas ferramentas utilizadas pelo projeto OpenBus. Utilizaremos a linguagem de programação Lua[27, 28] para nossas implementações, apesar de almejarmos uma infra-estrutura independente de linguagem. Para o apoio à essa independência, a infra-estrutura será baseada na arquitetura CORBA[16] 2.x, representada em Lua pelo ORB OiL[29]. A infra-estrutura de execução será baseada em componentes de *software*, e para a definição desses componentes utilizaremos uma versão aprimorada e mais nova (em relação à utilizada pelo OpenBus à época do desenvolvimento deste trabalho) do modelo SCS. Este modelo será descrito em poucos detalhes no Capítulo 3.

1.1 Objetivos e Contribuições

A infra-estrutura de execução aqui proposta para o OpenBus almeja dar suporte à instalação, carga e execução de componentes, e à configuração dinâmica das dependências de *softwares* baseados em componentes, em um ambiente distribuído. Nosso foco será o tratamento de problemas de execução, mas desejamos também iniciar o estudo e experimentação de técnicas de implantação. Outro objetivo é avaliar o uso da infra-estrutura em um cenário real e de interesse ao nosso grupo, que é o de criação de arcabouços de teste para aplicações distribuídas.

Como mencionado anteriormente, já existem sistemas que realizam a maior parte dos serviços providos pelo *middleware* OpenBus. Ao utilizarmos tais sistemas, no entanto, geralmente esbarramos em algumas dificuldades, principalmente relativas ao seu uso complexo e grande quantidade de bibliotecas anexadas, gerando módulos pesados. Por isso, temos também como objetivo criar uma arquitetura que reúna apenas as funcionalidades críticas, e que ao mesmo tempo ofereça interfaces pequenas e simples de se utilizar.

Como contribuições, temos como principal a extensão do *middleware* OpenBus para que atue também como uma infra-estrutura de execução. Outras contribuições são o relato das decisões de projeto tomadas, que podem servir de base para projetos futuros, e o estudo de soluções para os problemas encontrados que possam ser utilizadas nestes tipos de arquitetura.

Algumas das soluções que serão abordadas e que se diferenciam da maioria dos sistemas existentes são a carga de diferentes componentes em uma mesma área de memória, chamada de Contêiner de Componentes, e a possibilidade de suspensão das comunicações de todos os componentes que compartilhem um mesmo contêiner. As características e vantagens dessas abordagens serão discutidas ao longo deste trabalho.

1.2 Estrutura do Documento

Os próximos capítulos serão organizados da seguinte forma: o Capítulo 2 trata dos sistemas similares, dando uma visão geral dos seus aspectos similares, e tratando de alguns dos trabalhos mais relevantes separadamente. O Capítulo

3 descreve a infra-estrutura na qual nos baseamos, composta pelo modelo de componentes SCS e os serviços OpenBus que já existiam. A seguir, no Capítulo 4, descrevemos a infra-estrutura de execução, que engloba os principais serviços desenvolvidos para o *middleware* OpenBus. No Capítulo 5, descrevemos os experimentos que realizamos, fortemente ligados ao cenário de uso de testes de sistemas distribuídos. Por fim, no Capítulo 6, trazemos as conclusões e possíveis trabalhos futuros.