

# Um Estudo sobre Implantação de Sistemas Distribuídos Baseados em Componentes de Software

Amadeu Andrade Barbosa Júnior<sup>1</sup>

ajunior@inf.puc-rio.br

<sup>1</sup>Departamento de Informática  
Pontifícia Universidade Católica do Rio de Janeiro

## 1. Introdução

Componentes de software [18] também são definidos como *unidades de implantação* (do inglês *deployment*), *versionamento e substituição* [19]. Por uma unidade de implantação entende-se que um componente pode ser instalado e colocado em execução sem intervenção humana, dado um descritor. Enquanto uma unidade de versionamento, entende-se que um componente pode ser substituído por outra versão ou coexistir com outras versões. Essa definição sugere a separação entre as interfaces funcionais dos serviços e as atividades de manutenção do ciclo de vida, como implantação e configuração.

Em sistemas distribuídos, essa separação ajuda a definir melhor os artefatos de software a serem distribuídos e como implantá-los remotamente. Porém, nesse contexto é comum que os sistemas precisem rodar em diferentes plataformas e/ou integrando softwares implementados em diferentes linguagens. Como mostrou [19], os descritores de componentes ajudam na definição do que precisa ser feito para ter os artefatos de software em execução. Entretanto, essa tarefa pode ser mais complexa já que diferenças entre plataformas, tecnologias de rede e linguagem podem exigir ações específicas para cada combinação dessas. Assim, é importante que os *middlewares* de componentes para sistemas distribuídos incorporem estratégias automatizadas de implantação.

Por outro lado, tais demandas não são específicas das tecnologias de componentes. Em computação em grade (do inglês *grid computing*), as tecnologias para componentes de software podem ajudar [11] ao desenvolvimento das aplicações científicas contanto que permitam fases automáticas da implantação para lidar com a grande escala de distribuição dos recursos. Ainda em [11] observa-se que, para um usuário da grade, a implantação automática deve permitir o monitoramento, suspensão, reinício e término da execução dos componentes, invariante aos nós onde os componentes serão executados. Nesse sentido [1], este autor, indica desafios compartilhados e ferramentas interessantes.

Dessa forma, este trabalho tem por objetivos: (i) continuar o estudo sobre ferramentas relacionadas à implantação distribuída de componentes de software; (ii) eleger critérios para avaliar essas ferramentas e (iii) comparar as abordagens. A partir dessas informações espera-se identificar estratégias (ou soluções) interessantes para implantar componentes de softwares de forma distribuída em cenários com diversas plataformas e linguagens.

Este texto está organizado de forma que na seção 2 apresenta-se uma visão geral das ferramentas estudadas. Em seguida, estabelece-se critérios, na seção 3, para avaliar as tecnologias e comparar as abordagens, na seção 4. Por fim, na seção 5, discute-se sobre algumas dessas abordagens indicando possíveis trabalhos futuros.

## 2. Ferramentas estudadas

Em [1], este autor apresenta uma visão geral sobre as tecnologias CCM, EJB, Fractal e OpenCom, discutindo suas provisões para implantação distribuída. Adicionalmente, considera-se as ferramentas ADAGE e SMARTFROG, que embora voltadas ao contexto de computação em grade, destinam-se a tratar o problema da implantação distribuída. Dessa forma, nesta seção apresenta-se outras ferramentas a fim de complementar o levantamento iniciado anteriormente. O entendimento dessas ferramentas é importante para a compreensão das comparações e comentários que constam nas próximas seções.

### 2.1. CoRDAGE

Conforme [3], é difícil prever as reais necessidades de recursos físicos para executar aplicações científicas (numéricas de longa execução) ou serviços de compartilhamento de dados, num contexto de computação em grade. Nesses cenários torna-se interessante a dispor de facilidades para **expandir** e **retrair** a topologia da rede em tempo de execução. Isso traz implicações diretas nas metodologias de implantação distribuída, uma vez que estratégias estáticas<sup>1</sup> de implantação [10, 14, 15, 7, 4] não conseguem prover tais facilidades. Nessa condição, os usuários são obrigados a manualmente reconfigurar o plano e reiniciar toda execução da aplicação.

Esses autores introduzem os conceitos de re-implantação (do inglês *re-deployment*) e co-implantação (do inglês *co-deployment*), bem como uma ferramenta, o CoRDAGE, que implementa-os e funciona junto ao ADAGE. No primeiro conceito, o sistema de implantação deve estar preparado para concretizar mudanças (expansão ou retração) no plano de implantação assim que a aplicação esteja implantada e em execução. No segundo, esse sistema precisa implantar e gerenciar aplicações que possuam restrições<sup>2</sup> de funcionamento entre si, possibilitando que várias tecnologias sejam implantadas juntas. Quanto ao modelo de restrições, essas podem ser temporais - qual deve ser implantado antes - ou de co-alocação - em quais máquinas ambos devem estar.

A Figura 1 ilustra como o CoRDAGE altera a interação com as ferramentas de implantação na perspectiva do usuário. Diante dessa nova forma de interação, o usuário não lida diretamente com a ferramenta real de implantação, no caso o ADAGE [10]. Agora o usuário usa o CoRDAGE que vai se responsabilizar por selecionar os recursos (da *reservation tool*) e atualizar os planos de implantação (na *deployment tool*). O CoRDAGE adota um protocolo **xmlrpc**<sup>3</sup> para interagir diretamente com as aplicações e concretizar as operações relacionadas à re-implantação e co-implantação<sup>4</sup>.

Internamente, o CoRDAGE adota uma representação própria da aplicação em forma de árvores lógicas (do inglês *logical tree*) para gerir a aplicação, expandindo ou retraindo seu uso de recursos. Cada tipo de aplicação deve disponibilizar um tratador capaz de construir tais árvores. Esse deve ser escrito em C++ estendendo<sup>5</sup> o *framework* do CoRDAGE. Assim o núcleo do CoRDAGE consegue realizar certas operações da implantação como: *register*, *deploy*, *add\_sub\_app*, *discard*, *terminate*. Os recursos da

<sup>1</sup>Na bibliografia encontra-se a nomenclatura *one-shot deployments*

<sup>2</sup>Na ferramenta ADAGE já havia o conceito de restrições (do inglês *constraints*), embora estáticos.

<sup>3</sup><http://www.xmlrpc.com/spec>

<sup>4</sup>Assume-se que a *deployment tool* não dispõe dessas funcionalidades.

<sup>5</sup><http://cordage.gforge.inria.fr>

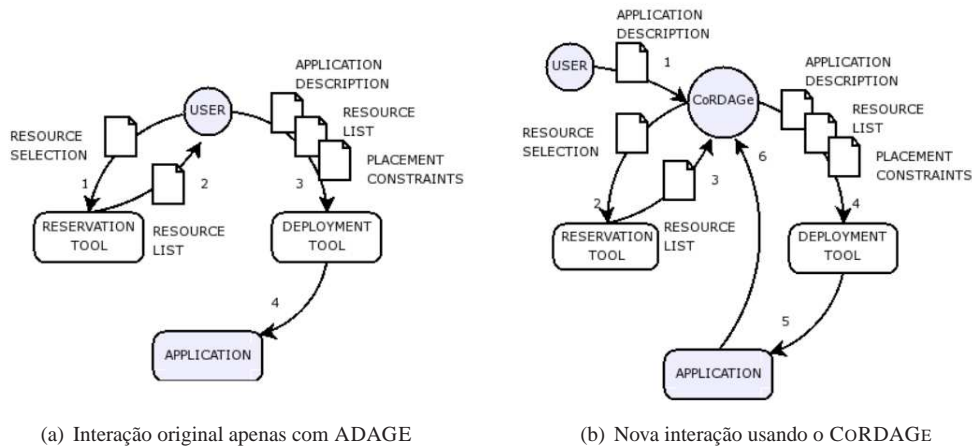


Figura 1. Alteração da perspectiva do usuário: antes e depois do CoRDAGE

grade também possuem uma representação hierárquica baseada em proximidade na rede e informações de serviços de informação da grade<sup>6</sup>.

## 2.2. DeployWare

Ainda no cenário da computação em grade, em [7] os autores apresentam motivações semelhantes a [10] que os motivam a criar um **metamodelo** que representa conceitos da implantação distribuída, na tentativa de mascarar a heterogeneidade dos softwares do ponto de vista do usuário. Dessa forma, os autores entendem que podem simplificar a descrição de dependências entre os softwares, adotar uma etapa de validação estática sobre o metamodelo e implantar diferentes *middlewares* de computação distribuída de uma forma autônoma. Além do metamodelo, os autores apresentam o *framework* DEPLOYWARE construído na forma de componentes Fractal [2]. O DEPLOYWARE agrega: (i) uma linguagem específica a domínio (do inglês *DSML*) para associar as entidades concretas da implantação ao metamodelo, (ii) uma máquina virtual que interpreta o metamodelo, (iii) uma ferramenta gráfica para monitorar e gerenciar, em tempo de execução, as aplicações implantadas.

O metamodelo usado no DEPLOYWARE possui diversas entidades. As principais, conforme Figura 2, são as *personalities*, compostas por *software types* que, por sua vez, representam os artefatos a serem implantados e podem indicar *dependencies* com outros *software types*. Cada *software type* possui um ou mais *procedures* que definem ações úteis durante a implantação. O usuário da implantação usa as diversas *personalities* disponibilizadas pela máquina virtual do DEPLOYWARE para descrever um plano, semelhante à Listagem 1. A máquina virtual [8], também chamada de FDF (*Fractal Deployment Framework*), instancia os componentes Fractal, semelhante ao da Figura 3, e executa o plano de implantação.

Considerando que, para implantar uma aplicação, a máquina virtual do FDF pode precisar gerenciar uma enorme quantidade de conexões de rede, o DEPLOYWARE permite o lançamento de várias máquinas virtuais do FDF na tentativa de escalar o próprio procedimento de implantação. Essa preocupação com a escalabilidade da própria implantação é fundamental e não foi comentada nos outros trabalhos estudados, até então.

<sup>6</sup>Em *middlewares* para computação em grade é comum haver um *Grid Information Service* agrupando informações sobre os nós.

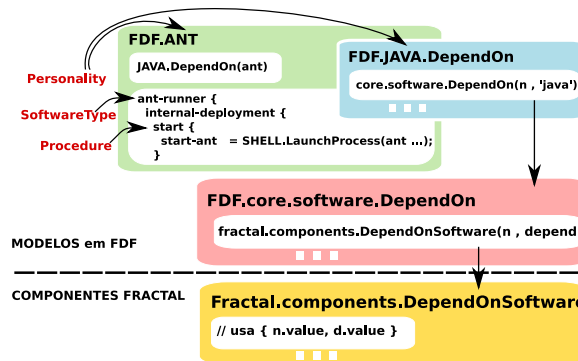


Figura 2. Relações entre metamodelos, modelos FDF e componentes Fractal

```

MyDESC.SC = FDF. Software {
  properties = {
    file = PARAM.ARCHIVE;
    home = PARAM.HOME;
    port = HTTP.PORT(9000);
  }
  install = {
    upload = TRANSFER.Upload(#[ file ]);
    make = SHELL.Execute(make);
  }
  start = {
    ping = SHELL.Execute(ping,#[ host ])
    ...
  }
  dependencies = {
    MyDESC.SA; MyDESC.SB;
  }
}

```

Listing 1. Descrição em FDF

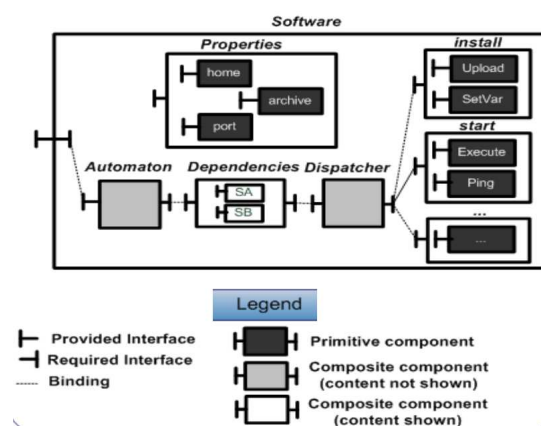


Figura 3. Componente Fractal gerado após a interpretação da descrição FDF

### 2.3. DAnCE

Diferente das abordagens em computação em grade, em [4] os autores focam seu estudo sobre sistemas distribuídos embarcados e de tempo-real (do inglês *DRE - Distributed Real-Time and Embedded*). Argumenta-se que *middlewares* como J2EE e .NET não se adequam a esse cenário, uma vez que, não incorporam tratamento de qualidade de serviço (QoS). Para tal considera-se o uso do CIAO<sup>7</sup> e da predictabilidade que a especificação RT-CORBA [17] permite. Nesse sentido, os autores indicam que a adoção de um *middleware* com suporte a QoS demanda desafios sobre: (i) adotar métodos eficientes de armazenar e obter implementações de componentes; (ii) prover tarefas de ativação, suspensão e desativação dos componentes automaticamente; (iii) configurar QoS no servidor de componentes para garantias de QoS fim-a-fim; (iv) simplificar a configuração e implantação de serviços comumente usados (*RT Event Service, Trade, Notification, Naming, etc*).

Para atender essas demandas, [4] apresenta o *framework* DANCE que implementa a especificação OMG D&C (*Deployment and Configuration*) [16], usando apenas o modelo de objetos CORBA 2.x<sup>8</sup>. Essa especificação (substitui o capítulo *Packaging and Deployment* contido na especificação OMG CCM 3.0 [15]) define uma arquitetura própria para implantação que inclui a gerência do repositório de componentes e permite etapas de configuração e instalação tanto da infra-estrutura quanto da aplicação. A Figura 4 ilustra

<sup>7</sup><http://www.dre.vanderbilt.edu/CIAO>

<sup>8</sup>Segundo os autores isso evita dependências cíclicas, embora tenhamos outra visão desse ponto que é discutida à frente.

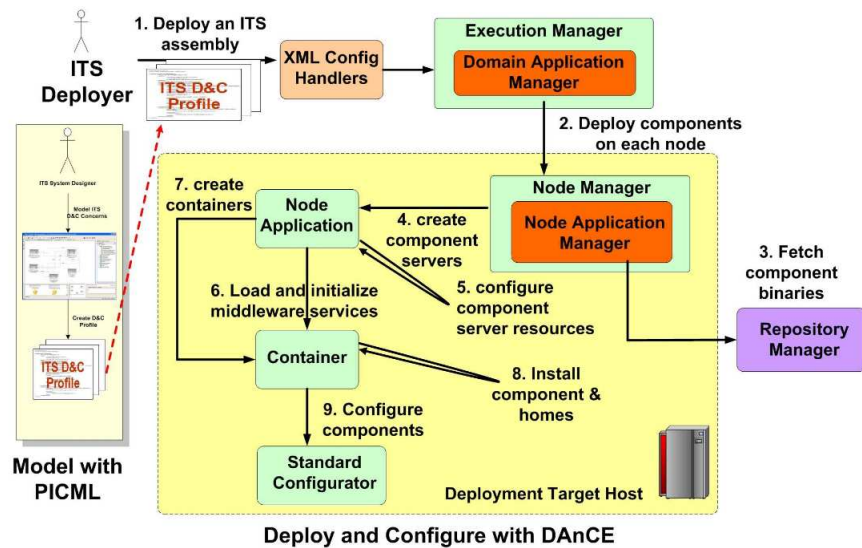


Figura 4. Arquitetura interna e sequência de eventos no DANCE

a arquitetura interna e o funcionamento do DANCE.

Nessa implementação da OMG D&C dada pelo DANCE, o *ExecutionManager* é um serviço (*daemon*) usado para gerenciar a implantação em vários domínios (*DomainApplication*) que são compostos por *nodes*, *interconnects*, *bridges* e *resources*, definindo um *TargetEnvironment*. Assim é tarefa do *DomainApplicationManager* (colocado junto ao *ExecutionManager*) instanciar cada nó do domínio. Considerando que o *NodeManager* é um serviço em cada máquina remota, ele é o responsável por concretizar a implantação dos componentes remotos. Para isso ele cria o *NodeApplicationManager* que lança os diversos *NodeApplications*, que agem como servidores de componentes. Esses servidores lançam contêineres e carregam os componentes conforme os meta-dados da implantação. Em especial, o *RepositoryManager* é dedicado para cada domínio (*DomainApplication*).

A fim de cumprir as demandas identificadas pelos seus autores:

1. O repositório no DANCE pode atuar como um cliente HTTP para se relacionar com outros repositórios e a obtenção dos componentes versionados é otimizada pelo uso de cache local no *NodeApplicationManager*.
2. O gerente de domínio (*DomainApplicationManager*) coordena o ciclo de vida dos componentes montados (do inglês *assemblies*), mantendo os estados **pré-ativo**, **ativo**, **passivo**, **inativo** sobre cada componente. Assim é possível garantir só conectar e ativar os componentes quando todos outros estiverem em **pré-ativo**. Além disso garante-se (pela tabela de despacho do POA e interceptadores do ORB) que as invocações remotas se concretizem antes do componente entrar num estado **passivo** ou **inativo**.
3. Estende-se a OMG D&C para permitir configurações no servidor de componentes (*NodeApplication*) de forma a (i) configurar opções de linha de comando no ORB e (ii) configurar opções da configuração dos serviços do ORB TAO<sup>9</sup>, pois há fábricas especializadas para restrições de tempo real como modelos de concorrência, filas de prioridades de conexões e despacho.

<sup>9</sup>ORB CORBA 2.x. <http://www.dre.vanderbilt.edu/TAO>

4. Implementa-se um serviço configurador em cada servidor (*NodeApplication*) que utiliza as interfaces padrões de cada serviço CORBA (eventos, nomes, trade, etc) através de fachadas do TAO para o CIAO<sup>10</sup>, mantendo a compatibilidade. Esse permite configurar os serviços CORBA através de um XML próprio.

### 3. Critérios da avaliação

Tendo em vista o reconhecimento de conceitos e trabalhos relacionados à implantação distribuída iniciado em [1] e complementado com a seção 2 deste, faz-se necessário a elicitacão de critérios qualitativos para comparar as diferentes abordagens. Embora esses trabalhos mostrem diferentes cenários de aplicacão, entende-se que as motivações são semelhantes. Todas tentam adotar estratégias alto-nível (e da forma mais autônoma possível) para desonerar seus administradores e desenvolvedores na árdua tarefa de gerenciar recursos físicos (máquinas e redes) e implantar suas aplicacões e/ou *middlewares* distribuídos nesses. Compartilhando dessa visão, a seguir são esclarecidos os critérios que nortearão a discussão da seção 4.

#### 3.1. Modelo de composicão e distribuicão

Em componentes de software, a COMPOSICÃO deve possibilitar a integração de vários componentes em blocos de componentes mais complexos. Além das linguagens de descrição de interfaces (IDL) é comum o uso de linguagens de mais alto nível para descrever a arquitetura dos componentes em termos de suas dependências e composicões, conhecidas como linguagens de descrição de arquiteturas (ADL). Cada tecnologia de componentes define uma forma própria de compor seus artefatos.

Conforme a taxonomia definida por [12], a composicão<sup>11</sup> pode ocorrer em diferentes fases do ciclo de vida dos componentes. Na **fase de projeto** é possível compor os componentes em nível de arquitetura e código produzindo unidades de código binário já compostos. Durante a **fase de implantacão**, entende-se por composicão a ligacão (lógica na forma de *assemblies*) entre componentes binários antes de disponibilizá-los para execução<sup>12</sup>. Por fim na **fase de execução** a composicão significa criar conexões entre componentes permitindo o encapsulamento. Ainda em [12], apresenta-se um ciclo de vida ideal para componentes, ilustrado na Figura 5, a fim de potencializar a reusabilidade através da composicão tanto na fase de projeto quanto na de implantacão.

Considerando os *middlewares* distribuídos de componentes, um elemento se mostra importante: o REPOSITÓRIO. Embora um repositório normalmente só tenha a função de **depósito**<sup>13</sup>, ele tem uma relacão direta com as possibilidades de composicão, podendo ajudar na reusabilidade dos componentes. Ainda na Figura 5, o repositório surge como uma entidade **atuante** no reuso ainda em fase de projeto (não apenas durante execução).

No cenário distribuído, também é fundamental adotar uma metodologia de DISTRIBUIÇÃO da aplicacão dado um conjunto de nós remotos. Quando a escala é grande faz sentido adotar um **alto nível** de abstração e mapear automaticamente<sup>14</sup> as aplicacões nos recursos físicos. Por outro lado, certas aplicacões demandam a execução em um

<sup>10</sup>Extensão ao TAO que implementa a especificacão de componentes CORBA 3.0

<sup>11</sup>Kung-Kiu Lau alerta que as teorias de composicão são marginalizadas em favor de abordagens por chamadas de métodos na maioria dos *middlewares* de componentes

<sup>12</sup>Normalmente, nomeia-se uma fase de montagem que faz parte da implantacão

<sup>13</sup>Entenda-se que serve para submeter e obter componentes apenas

<sup>14</sup>Normalmente através de planejadores/orquestradores

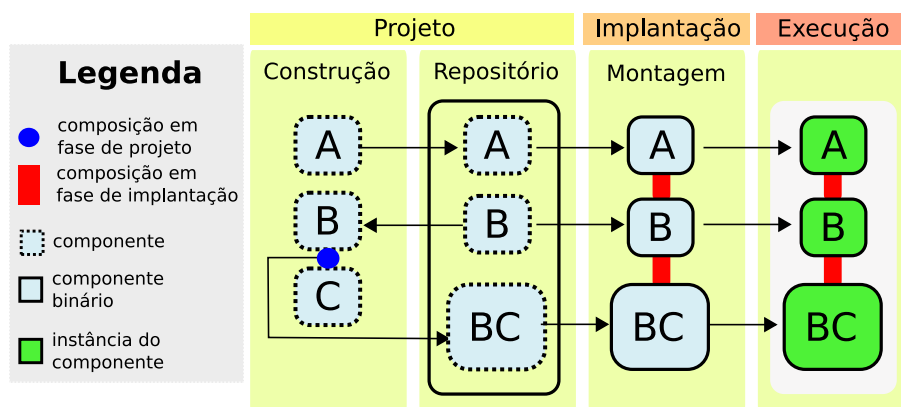


Figura 5. Um ciclo de vida ideal para um componente ilustrando as fases de composição

subconjunto específico de recursos. Ferramentas com bom nível de abstração, em geral, permitem uma forma de incluir restrições (do inglês *constraints*) sobre localização. Classifica-se como um **nível concreto** de abstração (da distribuição), aquele em que é possível apenas preencher descritores informando onde os artefatos devem ser implantados. Por completude, entende-se como um **baixo nível** de abstração aquele em que o programador precisa programaticamente definir como distribuir tais artefatos.

### 3.2. Abrangência tecnológica

Ao construir uma ferramenta para implantação um dos desafios é definir quais TECNOLOGIAS serão possíveis de ser implantadas. Abordagens de implantação de componentes mais puristas destinam-se a implantar seus **próprios modelos** de componentes. Diferentemente, há ferramentas que permitem descrições mais simples possibilitando a implantação de **quaisquer** softwares, muitas vezes em diferentes linguagens.

Uma outra questão tecnológica, intimamente relacionada ao quê se implanta, é quanto à LINGUAGEM em que os componentes ou softwares devem ter sido desenvolvidos para que a ferramenta implante-o no *middleware*. Essa é uma questão fundamental para cenários distribuídos interessados em suportar multi-plataforma e multi-linguagem e não pode passar despercebida ao tratar de implantação distribuída.

Sabendo-se o quê implantar e em qual linguagem o código está baseado, só falta identificar como ter ACESSO REMOTO até a máquina destino para então concretizar a implantação. Certas ferramentas, principalmente as baseadas em grade, podem optar entre os utilitários de **submissão de tarefas**, enquanto outras podem considerar lançar seus **próprios serviços** como estratégia de *bootstrap*.

### 3.3. Tratamento de dependências

A maioria dos modelos de componentes de software disponibilizam formas de especificar dependências INTERNAS entre si, através das conexões (ou ligações) entre diferentes tipos de **portas** para provisão e consumo de seus serviços. Há modelos que não apresentam a noção de ligações entre portas, logo não implementam dependências internas. Contudo para a implantação distribuída, o importante é indicar de alguma forma que há dependências entre os softwares. Assim, considera-se que as ferramentas (que não usam *middlewares* de componentes) podem ter dependências internas na forma de **descritores**, normalmente XML, que indicam as dependências entre softwares.

Para um processo de implantação distribuída, principalmente para diferentes plataformas e linguagens, é interessante reconhecer se o componente (ou software) a ser implantado possui alguma dependência EXTERNA, que deva estar previamente instalada no nó remoto. Esse critério baseia-se na discussão [1] (seção 2.3) sobre a dificuldade de tratar a instabilidade [13] de software. Em geral soluções práticas, mas não completas, existem nos sistemas de pacotes para distribuições de software livre [5, 13]. Para avaliar esse item, apenas identifica-se se a ferramenta provém alguma facilidade nesse sentido.

Para melhorar o entendimento, um cenário que ilustra a demanda por dependências externas é dado a seguir. Um desenvolvedor deseja prover um componente **AccessControl** que depende de uma biblioteca externa (de sistema) que implementa um serviço de autenticação (como LDAP). Esse desenvolvedor não quer ter o trabalho de criar um componente extra que entenda todas as combinações de plataforma e linguagem para permitir que seu componente **AccessControl** seja usado. De forma similar, o administrador do parque de máquinas quer poder garantir que essa biblioteca (e suas dependências consequentes) estejam instaladas apropriadamente em todas as plataformas onde o componente **AccessControl** precisa ser implantado. Essas tarefas normalmente são muito custosas no dia-a-dia de administradores e desenvolvedores, e são casos de uso típicos de um sistema de pacotes nos moldes que as distribuições de software livre disponibilizam.

### 3.4. Flexibilidade da implantação

Visto que a área de implantação distribuída de componentes (e software em geral) não é uma linha de pesquisa recente, espera-se que as ferramentas possuam interessantes recursos adicionais como **monitoração, reconfiguração, adaptação, suspensão, re-implantação e co-implantação**. Embora normalmente tais recursos estejam ligados à características não-funcionais das aplicações, alguns como re-implantação são importantes para diminuir o esforço manual de desenvolvedores e administradores. Assim, observa-se se qual recurso a ferramenta dispõe e sua forma de uso, se manual ou automatizada.

## 4. Comparação e comentários

**Modelos de composição e distribuição** A Tabela 1 mostra que não foi observado nenhuma forma de compor componente em fase de implantação<sup>15</sup>. Isso significa que nenhuma tecnologia permite, em tempo de implantação, agrupar dois ou mais componentes de forma binária. Um recurso desse gênero pode aumentar a reusabilidade, já que, por exemplo, faz sentido que o componente composto na implantação seja publicado num repositório e possa ser reusado na fase de projeto. Embora, o repositório seja importante para facilitar a obtenção dos componentes, só está presente na especificação CCM, e na ferramenta DANCE, mesmo assim, apenas com provisões de depósito. Logo, conforme seção 3.1, é interessante, para o reuso em projeto, criar um repositório atuante.

**Abrangência tecnológica** A Tabela 2 mostra que as ferramentas ADAGE+CORDAGE e DEPLOYWARE permitem a programação em quaisquer linguagens. Em detalhe, o ADAGE necessita da existência de um estilo XML e um *parser* escrito em C++, mas isso não impede do software a ser implantado esteja em qualquer outra linguagem. De forma análoga, o DEPLOYWARE também permite tal recurso, só que descrevendo componentes FRACTAL em Java. À primeira vista, incluir o DEPLOYWARE nessa categoria

<sup>15</sup>Em [12] comenta-se que apenas *JavaBeans* permite, mas não EJB

	<b>composição</b>	<b>repositório</b>	<b>distribuição</b>
ADAGE+CORDAGE	–	–	alto nível
SMARTFROG	–	–	baixo nível
EJB	projeto	–	baixo nível
OPENCOM	projeto, execução	–	baixo nível
CCM+DANCE	projeto, execução	depósito	nível concreto
FRACTAL+DEPLOYWARE	projeto, execução	–	alto nível

Tabela 1. Modelo de composição e distribuição

	<b>tecnologias</b>	<b>linguagem</b>	<b>acesso remoto</b>
ADAGE+CORDAGE	quaisquer	quaisquer	submissão tarefas
SMARTFROG	SmartFrog	Java	serviço próprio
EJB	JavaBeans	Java	serviço próprio
OPENCOM	OpenCom	CORBA 2.x	serviço próprio
CCM+DANCE	CCM	CORBA 2.x	serviço próprio
FRACTAL+DEPLOYWARE	quaisquer	Java	serviço próprio

Tabela 2. Abrangência tecnológica

parece injusto, mas considera-se que usar o *framework* FDF para descrever componentes é muito simples, conforme Listagem 1 seção 2.2.

As outras tecnologias só conseguem implantar componentes do seu próprio tipo, contudo entende-se que poderia-se encapsular diferentes tipos de softwares em componentes CCM, por exemplo. Contudo considera-se essa abordagem demasiadamente custosa nessas outras tecnologias. Já em relação à linguagem de programação, indica-se CORBA 2.x para referenciar que tais *middlewares* estão definidos sobre IDL CORBA 2.x e permitem a programação em todos mapeamentos CORBA oficiais.

**Tratamento de dependências** A Tabela 3 mostra que as ferramentas ADAGE+CORDAGE e SMARTFROG permitem a referência a dependências via descritores (XML e especializados, respectivamente), a fim de reproduzir o que os *middlewares* de componentes conseguem através da existências de portas de serviços requisitados (comumente chamadas de receptáculos). Em especial o EJB não possui recursos análogos aos receptáculos.

Por outro lado, nenhuma das tecnologias tratam dependências externas. Logo pode ser um interessante trabalho integrar um sistema de pacotes multi-plataforma e multi-linguagem a *middlewares* desse gênero, conforme comentado seção 3.3.

**Flexibilidade da implantação** As ferramentas ADAGE, DEPLOYWARE e EJB possuem facilidades para monitoramento, respectivamente, por comandos do console, ferramenta gráfica e *frameworks* extras (como JMX). Os trabalhos do SMARTFROG, DANCE e OPENCOM não apresentam nenhuma provisão para monitoramento. Em especial, o CORDAGE é o único que endereça re-implantação e co-implantação, embora o DANCE comente a re-implantação como trabalho futuro. Nas outras características como reconfiguração, suspensão e adaptação nenhum dos trabalhos estudados as implementa.

	<b>internas</b>	<b>externas</b>
ADAGE+CORDAGE	sim, descritores	–
SMARTFROG	sim, descritores	–
EJB	–	–
OPENCOM	sim, portas	–
CCM+DANCE	sim, portas	–
FRACTAL+DEPLOYWARE	sim, portas	–

**Tabela 3. Tratamento de dependências**

É importante observar que o processo de implantação, em todos esses trabalhos estudados, apresenta-se de forma simplificada. Conforme [9, 6] a implantação pode ter diversas etapas. Etapas como reconfiguração e atualização, mostram-se importantes para permitir que o plano de implantação evolua com o tempo e/ou esteja consistente com as mudanças que a aplicação pode ter sofrido, enquanto o sistema estava implantado. Logo, interessantes trabalhos seriam: (i) dar suporte a tais etapas de implantação e (ii) poder recuperar o estado do plano de implantação, dado um sistema já implantado.

## 5. Conclusões

Ao final deste estudo entende-se que os *middleware*s de componentes de software trazem interessantes contribuições no reuso, desacoplamento e na forma de implantar e distribuir as aplicações. Nos cenários de computação distribuída, a utilização de um processo sistemático de implantação desonera desenvolvedores e administradores de tarefas repetitivas e propensas a erros. Em geral, há uma tendência à adoção/implementação de processos de implantação versáteis com recursos dinâmicos para monitoramento, reconfiguração e re-implantação, por exemplo. Contudo nesse sentido, torna-se um desafio a validação dos planos de implantação e há indicações de trabalhos futuros [4, 7].

Entre as ferramentas estudadas, o ADAGE e CORDAGE possuem um interessante mapeamento automático entre as aplicações e os recursos físicos. Mesmo em cenários onde espera-se mais controle sobre quais máquinas serão usadas, essa estratégia é convidativa, na medida que, se possa ter restrições (*constraints*) de localização. O DANCE e a arquitetura da OMG D&C apresentam uma boa organização entre os contratos das entidades da implantação, porém, o volume de descrições, que precisam ser escritas junto ao código principal do componente, ainda é grande. Contudo, o próprio DANCE (e a especificação também) considera o uso de ferramentas de modelagem dirigida a modelos para, a partir de modelos mais alto nível, gerar os descritores. De uma forma geral, essa estratégia mostra-se interessante e pode reduzir o esforço em escrever todas descrições.

Em especial, em [4] comenta-se que o DANCE é implementado totalmente em objetos CORBA 2.x (não sendo componentes), a fim de evitar uma definição recursiva que produza dependências cíclicas. Entretanto, em [7] é interessante exatamente observar que o DEPLOYWARE é totalmente construído em componentes e pode inclusive se implantar em máquinas remotas. Essa definição recursiva da ferramenta de implantação, além de ser elegante, sugere-se importante para distribuir o próprio processo de implantação. Tal distribuição é fundamental para a escalabilidade da implantação em grandes parques de máquinas, como mostra [7].

Adicionalmente, este trabalho identifica potenciais investigações futuras sobre:

- a implementação de um repositório de componentes e de um ambiente de projeto mais dinâmicos, onde componentes compostos em outras fases, como implantação ou mesmo execução, possam ser publicados no repositório e disponibilizados para reuso pelos projetistas;
- a integração de um sistema de pacotes (semelhantes às distribuições de software livre) com suporte multi-plataforma e multi-linguagem que possa ajudar o *middleware*, e a ferramenta de implantação, na tarefa de instalar softwares complexos, sem obrigar ao desenvolvedor em encapsulá-los em componentes<sup>16</sup>;
- a implementação (ou extensão) de uma ferramenta de implantação distribuída com suporte a recursos dinâmicos (como re-implantação, reconfiguração, suspensão e adaptação).

## Referências

- [1] BARBOSA JR, A. A. Implantação de componentes de software e grades computacionais: desafios compartilhados, July 2008. <http://www.inf.puc-rio.br/~ajunior/puc/inf2545/artigos/monografia/monografia.pdf>.
- [2] BRUNETON, E., COUPAYE, T., LECLERCQ, M., QUÉMA, V., AND STEFANI, J.-B. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Software: Practice and Experience* 36, 11-12 (2006), 1257–1284.
- [3] CUDENNEC, L., ANTONIU, G., AND BOUGÉ, L. CoRDAGe: towards transparent management of interactions between applications and resources. In *Proceedings of the International Workshop on Scalable Tools for High-End Computing (STHEC'08)* (Kos Grèce, 2008), Michael Gerndt and Jesus Labarta and Barton Miller, pp. 13–24. This work has been supported by a grant of Sun Microsystems and a grant from the Regional Council of Brittany, France.
- [4] DENG, G., BALASUBRAMANIAN, J., OTTE, W., SCHMIDT, D. C., AND GOKHALE, A. DAnCE: A QoS-enabled component deployment and configuration engine. In *Proceedings of the 3rd Working Conference on Component Deployment (CD'05)* (2005), pp. 67–82.
- [5] DOLSTRA, E. *The Purely Functional Software Deployment Model*. PhD thesis, Universiteit Utrecht, 2006.
- [6] EMMERICH, W. Distributed component technologies and their software engineering implications. In *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)* (New York, NY, USA, 2002), ACM, pp. 537–546.
- [7] FLISSI, A., DUBUS, J., DOLET, N., AND MERLE, P. Deploying on the grid with DeployWare. In *Proceedings of the 8th International Symposium on Cluster Computing and the Grid (CCGRID'08)* (2008), IEEE Computer Society, pp. 177–184.
- [8] FLISSI, A., AND MERLE, P. A generic deployment framework for grid computing and distributed applications. In *Proceedings of the 2nd International OTM Symposium on*

---

<sup>16</sup>O encapsulamento em componentes poderia inclusive ser automático em certos casos

*Grid computing, high-performance and Distributed Applications (GADA'06)* (November 2006), vol. 4279 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1402–1411.

- [9] HALL, R. S., HEIMBIGNER, D., VAN DER HOEK, A., AND WOLF, A. L. An architecture for post-development configuration management in a wide-area network. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)* (Washington, DC, USA, 1997), IEEE Computer Society, p. 269.
- [10] LACOUR, S., PEREZ, C., AND PRIOL, T. Generic application description model: toward automatic deployment of applications on computational grids. *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on* (November 2005), 4 pp.–.
- [11] LACOUR, S., PÉREZ, C., AND PRIOL, T. Deploying CORBA components on a computational grid: General principles and early experiments using the Globus Toolkit. In *Proceedings of the 2nd International Working Conference on Component Deployment (CD'04)* (Edinburgh, Scotland, UK, may 2004), W. Emmerich and A. L. Wolf, Eds., no. 3083 in *Lect. Notes in Comp. Science*, Springer-Verlag, pp. 35–49.
- [12] LAU, K.-K., AND WANG, Z. Software component models. *IEEE Transactions on Software Engineering* 33, 10 (2007), 709–724.
- [13] MANCINELLI, F., BOENDER, J., DI COSMO, R., VOUILLON, J., DURAK, B., LEROY, X., AND TREINEN, R. Managing the complexity of large free and open source package-based software distributions. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 199–208.
- [14] MURRAY, P., AND GOLDSACK, P. Fully distributed service configuration management. In *Proceedings of the 3rd workshop on on Hot Topics in System Dependability (Hot-Dep'07)* (Berkeley, CA, USA, 2007), USENIX Association, p. 4.
- [15] OMG. CORBA Component Model specification. Tech. rep., 2006. <http://www.omg.org/technology/documents/formal/components.htm>.
- [16] OMG. Deployment and configuration of component-based distributed applications specification, v4.0. Tech. rep., 2006. <http://www.omg.org/cgi-bin/doc?formal/06-04-02>.
- [17] SCHMIDT, D., AND KUHN, F. An overview of the real-time corba specification. *Computer* 33, 6 (Jun 2000), 56–63.
- [18] SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [19] SZYPERSKI, C. Component technology: what, where, and how? In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)* (Washington, DC, USA, 2003), IEEE Computer Society, pp. 684–693.