5
Infraestrutura de Implantação para Componentes Distribuídos

Em cenários distribuídos, o processo de implantação para uma tecnologia de componentes precisa enfrentar desafios como heterogeneidade, complexidade da gestão e do monitoramento dos recursos e escalabilidade do próprio processo de implantação [6]. Contudo, conforme visto no Capítulo 3, o SCS delega ao administrador a responsabilidade de gerir diretamente todas entidades da infraestrutura de execução. Como consequências dessa abordagem, aumenta-se o volume de tarefas administrativas necessárias que precisam ser exercidas por atores humanos, levando a soluções pouco automatizáveis e que não incentivam a adoção da tecnologia de componentes. Logo, faz-se necessário abstrações de mais alto nível para gerir a implantação distribuída.

A solução tecnológica apresentada no Capítulo 4 representa a estruturação de algumas etapas da implantação a partir do uso de estratégias de empacotamento que automatizam a instalação dos componentes e suas dependências estáticas por linguagem ou plataforma. Contudo o administrador continua sendo obrigado a gerir as dependências paramétricas e a alocação de recursos diretamente no ambiente de execução.

Este capítulo apresenta serviços que auxiliam a implantação distribuída dos componentes SCS. Uma visão geral desses serviços é apresentada na Seção 5.1. Esses serviços foram projetados para permitir o planejamento da implantação e sua gestão em tempo de execução, como apresentado na Seção 5.2. Para tanto, a interface de programação oferecida flexibiliza o mapeamento dos componentes nos recursos físicos por níveis graduais de detalhamento, como apresentado na Seção 5.3. Dessa forma, viabilizamos a implantação automática de componentes SCS e, ao mesmo tempo, garantimos ao administrador total controle sobre o mapeamento caso seja necessário. Adicionalmente, esses serviços apresentam facilidades que estimulam o reuso de repositórios, conforme indicado na Seção 5.4. Por fim, por serem serviços distribuídos, permitimos que a implantação possa acontecer de forma a balancear a carga da própria implantação, conforme comentado na Seção 5.5. Todas as definições das interfaces e descritores em OMG IDL estão apresentadas no Apêndice A.

5.1 Visão Geral dos Serviços

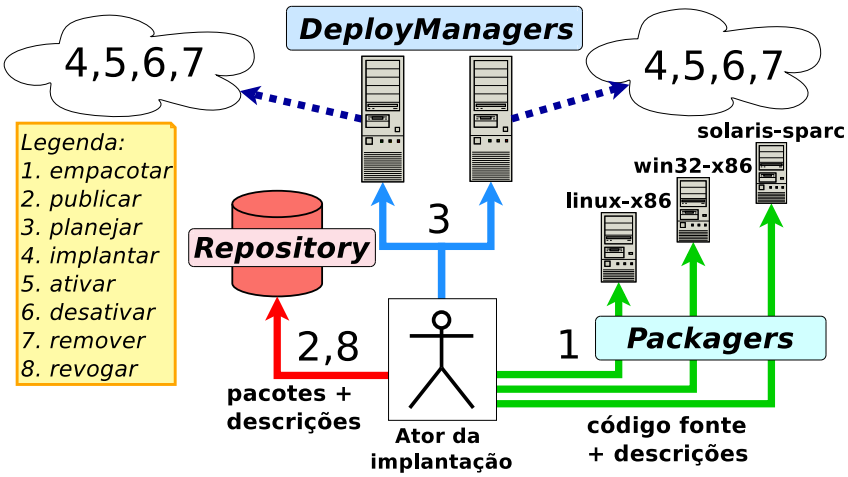

Figura 5.1: Visão geral do uso da infraestrutura de implantação distribuída

Este trabalho oferece os serviços *Packager* e *DeployManager* que implementam um processo de implantação com 8 etapas, ilustradas na Figura 5.1.

Primeiro, o ator da implantação (usuário) deve (1) **empacotar** seus componentes usando o *Packager* que automatiza a criação de pacotes e garante o cumprimento das regras do sistema de pacotes. Em seguida, o usuário deve (2) **publicar** os pacotes e as descrições dos seus componentes num *Repository* previamente conhecido. Assim torna-se possível (3) **planejar** a implantação através do *DeployManager* que encapsula o uso dos componentes da infraestrutura de execução e permite decisões de mais alto nível sobre sua configuração. A interface de programação do *DeployManager* permite (4) **implantar** os planos em um conjunto de máquinas físicas, (5) **ativar** os componentes, (6) **desativar** ao término da execução e (7) **remover** os seus componentes. Por fim, o usuário pode (8) **revogar** a publicação dos pacotes no *Repository* tornando seus componentes indisponíveis.

Os serviços foram implementados em Lua como objetos CORBA versão 2, possibilitando seu uso a partir de diferentes linguagens e plataformas.

A descoberta remota desses serviços pode ser feita por um serviço de nomes ou pelo prévio conhecimento da URI no formato *corbaloc* [60]. Para isso, cada instância do serviço usa um apelido (no Inglês, *nickname*) que também é a chave no mapa de objetos interno ao ORB, tornando-as acessíveis por endereços *corbaloc*. Por exemplo, a primeira instância do *DeployManager* na máquina *n1.tecgraf* usando a porta 2501 seria acessível pela URI *corbaloc::n1.tecgraf:2501/Deployer_1*. Já uma outra instância do mesmo serviço na mesma máquina, mas em outro processo que usa outra porta (3500, por exem-

plo) teria a URI *corbaloc::n1.tecgraf:3500/Deployer_1*. No caso do serviço *Packager*, a formação da URI é semelhante: *corbaloc::n1.tecgraf:2501/Packager_1*.

5.1.1 Serviço Packager

O serviço *Packager* encapsula o processo de empacotamento apresentado no Capítulo 4, simplificando a geração dos pacotes de componentes de forma distribuída. Recomenda-se que esse serviço seja executado em, ao menos, uma máquina de cada plataforma para usá-las como servidores de empacotamento. Assim diferentes desenvolvedores poderão gerar pacotes válidos nessas plataformas e depois publicá-los no repositório.

O Código 5.1 apresenta as operações do serviço *Packager*. Caso o usuário não esteja na mesma máquina onde o *Packager* executa, ele deve usar o método *create* para criar um pacote de componente a partir dos arquivo com o descritor do pacote (*rockspec*) e com o código fonte. Nesse caso, o conteúdo desses arquivos devem ser enviados como parâmetro. Por outro lado, caso o usuário esteja na mesma máquina onde o serviço *Packager* executa, basta criar o pacote de componente usando o método *create_from_dir* informando apenas a localização do arquivo do descritor do pacote e a pasta com o código fonte.

O Código 5.2 apresenta um exemplo programado em Lua do uso do *Packager* para a criação de pacotes, seja considerando-o localmente (linha 4) ou remotamente (linha 8). Nesse último caso é preciso fazer a leitura binária dos arquivos a serem enviados. Ambos métodos retornam o pacote do componente e o descritor do modelo do componente, que serão usados, posteriormente, durante a etapa de publicação.

Código 5.1: Interface OMG IDL implementada pelo *Packager*

```
1 module scs { module deployer {
2   interface Packager {
3     Package create(in OctetSeq rockspec, in OctetSeq srczip,
4       out ComponentTemplate tmpl)     raises (InvalidPackage, BuildError);
5     Package create_from_dir(in string rockspec_filename, in string dirname,
6       out ComponentTemplate tmpl)     raises (InvalidPackage, BuildError);
7   };
8 }; };
```

Código 5.2: Exemplo em Lua do uso do serviço *Packager* para empacotamento

```
1 packager = require("scs.deployer.Packager")()
2 dir            = "/home/amadeu/sources/hello-dev"
3 rockspec       = dir.."/hellochecked-1.0-0.rockspec"
4 pkg, template = packager:create_from_dir( rockspec, dir )
5
6 rockspec_stream = io.open(rockspec, "rb"):read("*a")
7 zip_stream      = io.open(dir.."/Hello.zip", "rb"):read("*a")
8 pkg, template   = packager:create( rockspec_stream, zip_stream )
```

5.1.2 Serviço DeployManager

O serviço *DeployManager* encapsula o uso da infraestrutura de execução do SCS provendo facilidades para o usuário. Dentre as facilidades, é possível criar planos (roteiros) de implantação por onde define-se toda a configuração inicial da aplicação distribuída. Entretanto, não é obrigatório definir o mapeamento físico nas máquinas ou na infraestrutura de execução do SCS.

A manipulação dos planos de implantação se dá por uma interface de programação mais flexível que as abordagens de descrição de arquitetura (do Inglês, *Architecture Description Language - ADL*). Essa flexibilidade é alcançada porque é possível escrever os planos de implantação em qualquer linguagem que possua um mapeamento CORBA. Assim, permite-se o uso de linguagens com iteradores, estruturas de controle condicional, laços e outras abstrações comumente encontradas em linguagens como Java, C++, Lua e Ruby, por exemplo. Dessa forma, o usuário é livre para usar todos os recursos da sua linguagem preferida. Outro benefício é que não exige-se que o usuário aprenda uma nova linguagem ou use uma linguagem de descrição que, eventualmente, poderia ter limitações.

Ao mesmo tempo, a interface de programação oferecida pelo *DeployManager* facilita a integração desse trabalho com linguagens de descrição de arquitetura. Por exemplo, os compiladores (ou interpretadores) de ADL podem usar os serviços oferecidos neste trabalho para implantar e administrar as aplicações baseadas em componentes, mesmo em tempo de execução.

Os planos de implantação além de padronizar o uso de métodos relacionados às etapas de implantação como *deploy*, *activate*, *deactivate* e *undeploy*, ainda possibilitam gerir os componentes implantados em tempo de execução. Assim, pode-se usar os planos dinamicamente para administrar a execução da aplicação baseada em componentes. Isso é possível porque todas as entidades internas ao plano oferecem o acesso direto às facetas dos componentes implantados através do método *get_facet*, como será ilustrado posteriormente.

O Código A.8 contido no Apêndice A.3 apresenta a interface *Manager* especificada em OMG IDL que é implementada pelo serviço *DeployManager*. A seguir, na Seção 5.2, detalha-se as relações entre a interface *Manager* e os planos de implantação.

Os planos de implantação exemplificados ao longo deste trabalho, para efeito didático, estão programados em Lua porque o ORB OiL [46] permite gerenciar as estruturas de dados especificadas em OMG IDL como tabelas Lua, o que facilita manter os exemplos pequenos e simples de ler.

5.2 Planejamento da Implantação

Durante a implantação de uma aplicação complexa, é comum o projetista necessitar de assistência para validar sua arquitetura antes de realmente ter os componentes instanciados nas máquinas remotas. A partir dessa motivação, o serviço *DeployManager* assume uma fase de planejamento. Um planejamento inicia com a criação de um *plano de implantação* e é concretizado no ato da chamada do método *deploy*. Em particular, pela abordagem de níveis graduais de detalhamento (analisada a seguir na Seção 5.3), o método *deploy* está disponível em todas as entidades relacionadas à implantação. Mas um uso típico é invocar esse método no objeto do plano. Nesse caso, o *DeployManager* procede à implantação de todos os componentes relacionados àquele plano.

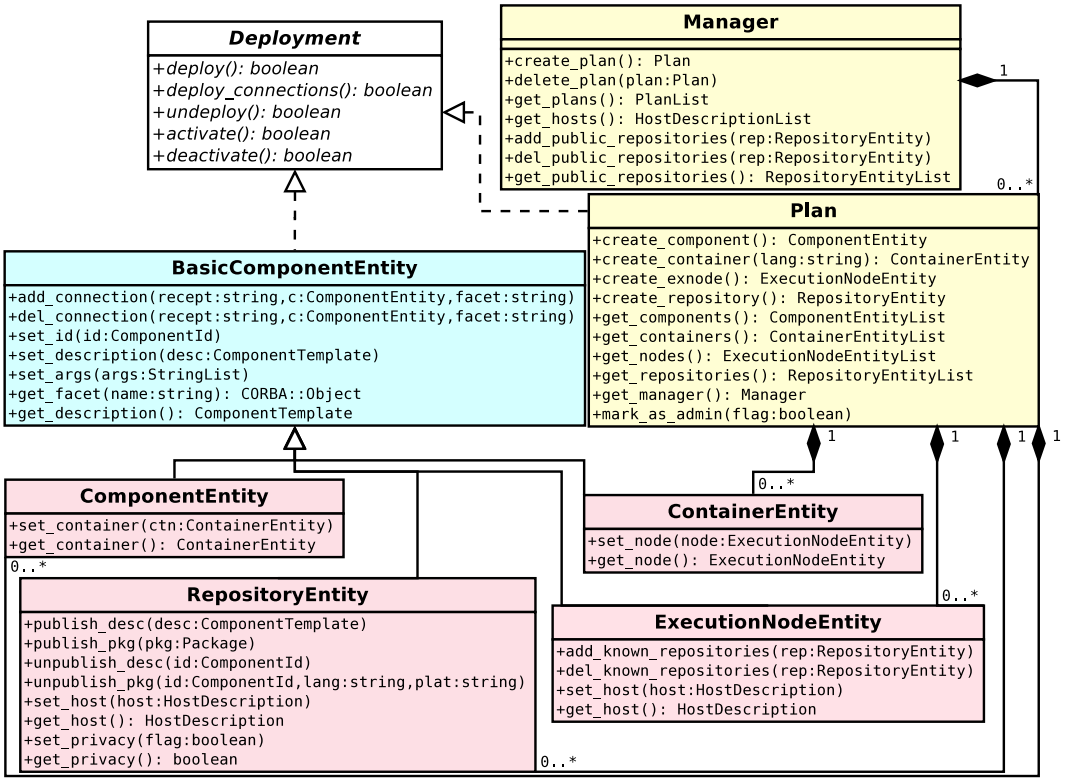

Figura 5.2: Relações entre as entidades virtuais e o DeployManager

A fim de viabilizar o planejamento como uma etapa pré-execução, usa-se o conceito de *entidades virtuais* as quais possuem relação 1:1 com as instâncias reais dos componentes do usuário e da infraestrutura de execução do SCS. A Figura 5.2 apresenta o diagrama de classes entre as entidades virtuais, os planos e o serviço *DeployManager*.

As entidades virtuais mantém um nível de indireção com as instâncias reais e encapsulam a lógica da carga, da conexão entre componentes e do

ordenamento necessário para ativação e desativação dos componentes. Em especial, existem entidades que representam a infraestrutura de execução do SCS (*Container, ExecutionNode e Repository*) e que simplificam o uso dessa infraestrutura, pois oferecem uma interface de programação mais simples, pela qual o projetista pode desconhecer os detalhes da configuração entre aqueles componentes. Todas entidades virtuais são objetos CORBA e, assim, permite-se a comunicação remota entre entidades associadas a diferentes planos ou a diferentes instâncias do *DeployManager*.

A indireção imposta pelas entidades virtuais não causa perdas de desempenho nas aplicações implantadas, pois essas entidades só agem em metanível, ou seja, as instâncias reais dos componentes não manipulam referências para as entidades virtuais. As entidades virtuais apenas facilitam a implantação e a gestão dos componentes implantados mas garantem o acesso direto às instâncias dos componentes uma vez implantados, conforme visto na Seção 5.1.2.

O algoritmo de implantação implementado no serviço *DeployManager* (no método *deploy*) segue uma abordagem semelhante à busca em profundidade, conforme pseudo-código apresentado no Código 5.3. Ao invocar *deploy* num suposto componente A do usuário[1], será solicitada a implantação do *Container*, que obrigará a implantação do *ExecutionNode* e dos componentes *Repository* conhecidos. Para efetivar as conexões aos receptáculos de A, é solicitada a implantação de todos outros componentes com os quais A se conecta. Caso algum desses componentes já tenha sido implantado previamente, a implantação procede sem causar duplicação.

Código 5.3: Pseudo-código do algoritmo de implantação

```
1  função ComponentEntity.deploy {
2    1. implantar recursivamente todos os ComponentEntity dependentes
3    1.1 chamar ComponentEntity.deploy em cada entidade com
4        uma conexão previamente planejada
5    2. implantar o ContainerEntity associado ao componente,
6       caso ainda não tenha sido implantado
7    2.1 implantar o ExecutionNodeEntity associado ao Contêiner,
8        caso ainda não tenha sido implantado
9    2.1.1 implantar o RepositoryEntity associado ao Nó de Execução,
10         caso ainda não tenha sido implantado
11   2.1.2 conectar a faceta ComponentRepository do repositório
12         no Nó de Execução
13   2.2 conectar faceta Installer do Nó de Execução no Contêiner
14   3. solicitar a carga do componente através da
15      faceta ComponentLoader do Contêiner
16 }
```

A mesma estratégia recursiva usada nesse algoritmo também vale para a ativação (método *activate*). Nos casos de desativação (método *deactivate*) e de

[1]Entende-se como componente do usuário aquele que não seja um *Container*, *ExecutionNode* ou *Repository*.

remoção (método *undeploy*) só são desativados ou removidos os componentes que recebem diretamente tais chamadas de métodos. Alternativamente, o usuário também pode solicitar essas operações no plano de implantação ao invés de manipular individualmente cada componente. Nesse caso, a mesma operação requisitada no plano será executada sobre todos os componentes automaticamente.

5.3 Níveis Graduais de Detalhamento

Aproveitando a indireção considerada na fase de planejamento, permite-se que o usuário especifique a implantação dos componentes em níveis gradativos de detalhamento. Em um nível mais alto, o usuário só precisa indicar quais componentes e como eles se conectam, sem a necessidade de decidir sobre o mapeamento na infraestrutura de execução (*Container*, *ExecutionNode* ou *Repository*) ou nos recursos físicos (máquinas). Dessa forma, o *DeployManager* deve se responsabilizar por tal mapeamento.

Um exemplo desse uso em um nível mais alto é apresentado no Código 5.4 e exercita a implantação de 10 componentes *HelloChecked*. Nesse exemplo para implantar os componente basta: (i) criar um plano e uma entidade virtual para cada componente (linhas 7–10), (ii) especificar o identificador do tipo do componente (linha 11), (iii) especificar as conexões entre os componentes (linha 15), e (iv) invocar a implantação (linha 17). Em seguida, é possível ativar todos os componentes do plano (linha 18), programar às ações específicas da aplicação (linhas 20– 22) e, por fim, remover todos os componentes (linha 23).

Código 5.4: Exemplo em Lua que implanta componentes em mais alto nível

```
1  manager = orb:newproxy("corbaloc::n1.tecgraf:2501/Deployer_1",
2                         "IDL:scs/deployer/Manager:1.0")
3  local template_id = {
4    name = "HelloChecked",
5    major_version=1,  minor_version=0,  patch_version=0,
6  }
7  plan = manager:create_plan() -- início do planejamento
8  local hello={} -- instanciação de 10 entidades virtuais que
9  for i = 1,10 do -- representam 10 instâncias de componentes reais
10   hello[i] = plan:create_component()
11   hello[i]:set_id( template_id )
12 end
13 for i = 1,10 do -- configuração entre os componentes do usuário
14   j = (i+1)%10
15   hello[i]:add_connection("OtherHello", hello[j], "Hello")
16 end
17 plan:deploy()   -- implantação de todos componentes do plano
18 plan:activate() -- ativação de todos componentes do plano
19 -- exemplo de ações específicas da aplicação
20 local hello1_facet = orb:narrow(hello[1]:get_facet("Hello"),
21                                  "IDL:scs/demos/hello/Hello:1.0")
22 hello1_facet:sayHello("Hey there, I'm alive!")
23 plan:undeploy() -- remoção de todos componentes do plano
```

Alternativamente, o usuário pode intervir manualmente no mapeamento em 3 níveis graduais:

1. Agrupando os componentes de usuário em um mesmo *Container*: basta associar os componentes a um *Container* através do método *set_container* disponível na entidade virtual do componente.

2. Agrupando os *Containers* em um mesmo *ExecutionNode*: basta associar o *ExecutionNode* a cada *Container* através do método *set_node* disponível na entidade virtual do contêiner.

3. Definindo a máquina do *ExecutionNode*: basta associar a descrição da máquina a um *ExecutionNode* através do método *set_host* disponível na entidade virtual do nó de execução.

Portanto, caso essas ações supracitadas não tenham sido tomadas então o algoritmo de mapeamento completará automaticamente o restante da implantação.

Atualmente, o *DeployManager* implementa um algoritmo simples para o mapeamento automático. Esse algoritmo se baseia em *round-robin* para distribuir uniformemente os componentes entre uma quantidade pré-determinada de contêineres e máquinas e só executado durante a etapa de implantação (método *deploy*). Esse algoritmo é parametrizável por três informações: (i) a lista das máquinas gerenciadas por um *DeployManager*, (ii) um número máximo de contêineres por máquina, e (iii) um número máximo de componentes por contêiner. Por padrão, a lista de máquinas é previamente conhecida pela instância do *DeployManager* pois considera-se que cada *DeployManager* gerencia uma rede ou um aglomerado de máquinas. Além disso, os números máximos de contêineres por máquina e componentes por contêiner são definidos por padrão como 5, mas são configuráveis durante em tempo de planejamento.

O algoritmo de mapeamento automático permite a construção de planos com diferentes níveis de detalhamento, conforme é apresentado no Código 5.5. Segundo esse algoritmo, caso o usuário não indique um contêiner associado a um componente então o método *autofit_container* decide qual o tipo de contêiner é compatível com uma das implementações disponíveis para o tipo do componente, conforme detalhado no Código A.9. Se o usuário não indicar um nó de execução associado a um contêiner então o método *autofit_exnode* decide qual o tipo de nó de execução é capaz de hospedar um dos possíveis contêineres (ou escolhido pelo usuário ou pelo *autofit_container*), conforme detalhado no Código A.10. Por fim, caso o usuário não indique uma máquina associada a um nó de execução então o método *autofit_host* decide qual a máquina (entre as conhecidas pelo *DeployManager*) é compatível com as restrições

impostas pela implementação do componente que foi escolhida (ou pelo usuário ou pela execução dos métodos *autofit_container* e *autofit_exnode*), conforme detalhado no Código A.11. A implementação desses métodos encontra-se no Apêndice A.4.

Código 5.5: Trecho de código do algoritmo de mapeamento automático durante a implantação

```
1  function ComponentEntity:deploy()
2    --...
3    --consulta sobre as implementações disponíveis nos repositórios
4    local implementations = self:get_available_implementations()
5    if not self.container then
6      self:set_container( self:autofit_container(implementations) )
7    end
8    --se um contêiner foi escolhido então sabe-se a linguagem
9    local language = self.container:get_property("language")
10   if not self.container:get_node() then
11     self.container:set_node( self:autofit_exnode(implementations,language) )
12   end
13   --se um nó de execução foi escolhido então sabe-se a plataforma
14   local node = self.container:get_node()
15   local platform = node:get_property("platform")
16   if not node:get_host() then
17     node:set_host( self:autofit_host(implementations,language,platform) )
18   end
19   --...
20 end
```

O mapeamento automático por níveis graduais de detalhamento proposto neste trabalho é extensível. Caso se deseje experimentar novas estratégias basta fornecer outras implementações para os métodos *autofit_container*, *autofit_node* e *autofit_host*. Atualmente, esses métodos são módulos isolados no serviço do *DeployManager* sendo simples trocá-los por outras implementações com pouco impacto no código do serviço. A assinatura desses métodos é tal que eles recebem (i) uma lista de implementações disponíveis para o tipo de componente solicitado no plano e (ii) um conjunto de restrições de linguagem e plataforma que eles devem respeitar.

5.4 Acesso aos Repositórios

A fim de difundir o uso e compartilhamento de repositórios de implementações, o *DeployManager* permite o cadastramento de repositórios públicos. Esses repositórios são idênticos a qualquer outro, só que são acessíveis por qualquer plano criado na mesma instância do *DeployManager* e além disso podem estar sendo compartilhado por outros *DeployManager* ou seus planos. Para tornar um repositório público, o administrador que criar o plano para implantação do repositório deve usar o método *set_privacy* (presente na entidade virtual do repositório) passando o parâmetro *false*. Dessa forma a entidade virtual do repositório cadastrará o repositório numa lista de repositórios públicos

que é acessível através da interface do serviço *DeployManager*. Assim, espera-se que os desenvolvedores busquem a lista de repositórios públicos e publiquem suas implementações de componentes neles.

Além desse mecanismo, é possível controlar o acesso aos repositórios de outras formas. O *middleware* CORBA permite o uso de interceptadores de chamadas que podem implementar estratégias de segurança para bloquear ou liberar o acesso aos objetos. Este trabalho não implementa nenhuma dessas estratégias mas é trivial usar o recurso de interceptadores de CORBA caso deseje-se controlar o acesso remoto para os repositórios ou mesmo para o serviço *DeployManager* como um todo.

5.5 Escalabilidade da Implantação

O processo de implantação precisa estar apto a ser distribuído, já que a escala dos recursos físicos e das entidades remotas pode alcançar níveis impossíveis de gerenciar a partir de um único ponto na rede [6]. Com base nessa preocupação, o *DeployManager* é nativamente um serviço distribuído que pode ser usado de forma a balancear a carga da própria implantação de aplicações em grandes conjuntos de máquinas.

O Código 5.6 exemplifica um planejamento usando 4 instâncias distintas do serviço *DeployManager* (linhas 4–7). Em cada instância do serviço *DeployManager* cria-se um plano (linhas 8–12) e cada plano se responsabiliza por implantar 100 componentes (linhas 13–20), totalizando, assim, 400 componentes distribuídos. Assume-se que cada instância do serviço *DeployManager* gerencia um conjunto de máquinas diferentes. Nesse exemplo, o mapeamento automático foi parametrizado para permitir um máximo de 10 componentes para cada contêiner (linha 10) e 2 contêineres por máquina (linha 11), logo cada plano usará 5 máquinas físicas. Por fim, como as entidades virtuais também são objetos CORBA, simplifica-se a conexão entre os componentes gerenciados por diferentes instâncias do *DeployManager* (linhas 21–26).

É importante ressaltar que, para o exemplo acima funcionar, é preciso existir rotas válidas entre as redes dos diferentes conjuntos de máquinas, pois foram planejadas conexões entre componentes de planos diferentes. Além disso, como visto na Seção 5.2, o algoritmo de implantação permite que a operação *deploy* seja invocada apenas num plano e automaticamente os componentes dependentes também serão implantados, mesmo que mais de um *DeployManager* esteja envolvido na implantação.

Código 5.6: Exemplo em Lua do uso de quatro *DeployManager* distribuídos

```
1  local managers={}; local plans={}; local hello={} --vetores vazios
2  local idltype = "IDL:scs/deployer/Manager:1.0"
3  --referências remotas para os diferentes DeployManagers
4  managers[1] = orb:newproxy("corbaloc::cluster1:2501/Deployer_1",idltype)
5  managers[2] = orb:newproxy("corbaloc::cluster2:2501/Deployer_1",idltype)
6  managers[3] = orb:newproxy("corbaloc::cluster3:2501/Deployer_1",idltype)
7  managers[4] = orb:newproxy("corbaloc::cluster4:2501/Deployer_1",idltype)
8  for i=1,4 do --instanciação de um plano por cluster
9    plans[i] = managers[i]:create_plan()
10   plans[i]:set_max_container_usage( 10 ) -- 10 componentes por contêiner
11   plans[i]:set_max_node_usage( 2 )       -- 2 contêineres por máquina
12 end
13 for i=1,4 do
14   hello[i]={}
15   for j=1,100 do --criação de 100 instâncias por plano
16     hello[i][j] = plans[i]:create_component()
17     hello[i][j]:set_id( {name = "HelloWorld",
18                          major_version=1, minor_version=0, patch_version=0} )
19   end
20 end
21 for i=1,4 do     --componentes do plans[i] se conectam aos do plans[i+1],
22   for j=1,100 do --e aqueles do plans[4] se conectam aos do plans[1]
23     hello[i][j]:add_connection("HelloReceptacle",
24                                hello[(i+1)%4][j], "Hello")
25   end
26 end
27 plans[1]:deploy()   --implantação do plans[1] causará outras implantações
28 plans[1]:activate() --ativação do plans[1] causará outras ativações
```

5.6 Limitações da Implementação Atual

As facetas de instalação (faceta *Installer* no *ExecutionNode*) e carga dos componentes (faceta *ComponentLoader* no *Container*) não permitem a distinção de duas implementações de um tipo de componente que: (i) tenham sido desenvolvidas numa mesma linguagem e com suporte a uma mesma plataforma; e (ii) sejam diferentes, por exemplo, apenas pela sua qualidade de serviço. Portanto, atualmente, o suporte a múltiplas implementações de um mesmo tipo de componente limita-se ao tratamento de implementações que diferem entre si apenas pela linguagem em que foram programadas ou pela plataforma que podem executar.

O *DeployManager* permite o acesso aos objetos dos planos de implantação. É importante que os planos possam ser persistidos em disco para posterior recuperação, a fim de promover seu reuso. Atualmente, isso não é feito e todos os planos deixam de existir quando o *DeployManager* para.

A implementação das etapas de implantação, fornecida pelo *DeployManager*, apresenta um problema se forem feitas solicitações concorrentes aos métodos *deploy*, *activate*, *deactivate* e *undeploy* para um mesmo componente ou um mesmo plano. Quando um componente já está implantado esse problema não se manifesta e apenas retorna-se uma exceção indicando que o componente (ou plano) já foi implantado previamente. Mas se o *DeployManager* receber

uma requisição concorrente à execução de um desses métodos no mesmo componente (ou plano), então isso causará uma inconsistência.

Por outro lado, há uma forma simples de resolver esse problema de concorrência. O serviço *DeployManager* está implementado em Lua e utiliza a implementação CORBA fornecida pelo ORB OiL. A linguagem Lua oferece um modelo de *multithreading* cooperativo através do recurso de co-rotinas. As co-rotinas Lua não são preemptivas, assim, enquanto uma co-rotina está executando ela não pode ser interrompida sem uma requisição explícita e voluntária da própria co-rotina pela chamada da função *yield*. No OiL existe um escalonador de co-rotinas que baseia-se no bloqueio por acesso à rede para escolher uma nova co-rotina apta a executar. Dessa forma, o problema da concorrência no *DeployManager* pode ser facilmente resolvido pela adição de variáveis que indiquem um estado interno extra, como por exemplo, "deploying". Esse estado interno deve representar o fato de estar no meio da execução de, por exemplo, uma implantação (*deploy*) para evitar que a nova requisição comece a executar antes do término da requisição antiga (eventualmente bloqueada por alguma chamada remota originada ao longo da primeira requisição). Garantir-se-á, portanto, que não haverá trocas de contexto durante uma atribuição ou comparação de uma variável pois Lua processa tais operações atomicamente uma vez que seu modelo de *multithreading* é cooperativo e não-preemptivo.

5.7 Considerações Finais

Ao fim deste Capítulo é interessante analisar o suporte oferecido por este trabalho para a implantação de componentes e comparar com a abordagem original da infraestrutura de execução do SCS. A Tabela 5.1 apresenta essa comparação de acordo com os critérios propostos na Seção 2.2.

Critério Classificatório	Infraestrutura de Execução Original [42]	Nova Infraestrutura de Implantação para o SCS
Controle da Implantação	dinâmico por acesso direto	dinâmico por interface padrão
Suporte à Composição	execução	implantação e execução
Repositório de Implementações	**acesso:** remoto **uso:** dinâmico **fase:** projeto, implantação, execução	**acesso:** remoto **uso:** dinâmico **fase:** projeto, implantação, execução
Abstração da Distribuição	manual	abstrato
Modelo de Dependências	paramétricas	paramétricas e estáticas
Abrangência Tecnológica	**linguagem:** Lua, Java e C++ **acesso:** serviço próprio	**linguagem:** Lua, Java e C++ **acesso:** serviço próprio

Tabela 5.1: Comparação entre as classificações da infraestrutura de execução original do SCS e a nova infraestrutura de implantação