AGrADC: Uma Arquitetura para Implantação e Configuração Autônomas de Aplicações em Grades Computacionais*

Sidnei Roberto Selzler Franco[1], **Luciano Paschoal Gaspary**[2], **Marinho Pilla Barcellos**[1], **Gerson Geraldo Homrich Cavalheiro**[3]

[1]Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos (UNISINOS)

sidneif@turing.unisinos.br, marinho@acm.org

[2]Instituto de Informática – Departamento de Informática Aplicada
Universidade Federal do Rio Grande do Sul (UFRGS)

paschoal@inf.ufrgs.br

[3]Instituto de Física e Matemática – Departamento de Informática
Universidade Federal de Pelotas (UFPel)

gerson.cavalheiro@ufpel.edu.br

Abstract. *Deployment and configuration of grid computing applications are exhaustive and error-prone tasks, representing a weak link of the lifecycle of grid applications. To address the problem, this paper proposes AGrADC, an architecture to instantiate grid applications on demand, which incorporates features from the Autonomic Computing paradigm. This architecture improves the grid applications development process, providing tools to define (a) a deployment flow, respecting dependencies among components that compose the application, (b) configuration parameters and (c) actions to be executed when adverse situations like faults arise. The result of this process, materialized in the form of a set of documents, is delivered to an instantiation engine, which starts to autonomously conduct and manage the deployment and configuration process.*

Resumo. *A implantação e a configuração de aplicações em grades computacionais são tarefas exaustivas e sujeitas a erros, ainda representando elo fraco do ciclo de vida de aplicações desta natureza. Para lidar com o problema, este artigo propõe AGrADC, uma arquitetura para instanciação sob demanda de aplicações em grades que incorpora características da Computação Autônoma. Esta arquitetura instrumenta o processo de desenvolvimento de aplicações para grades computacionais, oferecendo ferramentas para definir (a) um fluxo de implantação, respeitando dependências entre componentes que compõem a aplicação, (b) parâmetros de configuração e (c) ações a serem executadas diante de situações adversas tais como falhas. O resultado desse processo, materializado na forma de um conjunto de documentos, é repassado a um motor de instanciação, que passa a autonomamente conduzir e gerenciar o processo de implantação e configuração.*

1. Introdução

O uso de grades computacionais tem se expandido de forma gradual e consistente nos últimos anos [Simmons e Lutfiyya 2005]. As infra-estruturas de grades computacionais são construídas a partir do compartilhamento de recursos distribuídos. Dentre seus

*Este trabalho foi desenvolvido em colaboração com a HP Brasil P&D.

objetivos, incluem-se: (a) reduzir a necessidade de atualização de equipamentos aproveitando computadores geograficamente dispersos; (b) fornecer desempenho superior na solução de problemas que exigem processamento intensivo; (c) tirar melhor proveito de recursos explorando capacidade ociosa [Kesselman e Foster 1998]; (d) prover transparência no uso de recursos, considerando-os como um único e poderoso computador [Nemeth e Sunderam 2002]; (e) oferecer disponibilidade de acesso apenas a recursos específicos de um nodo [Nemeth e Sunderam 2002]. Observa-se que com a evolução das tecnologias de grade, elas incorporam novas funcionalidades e melhoram os serviços oferecidos. Entre os avanços perseguidos, destaca-se a necessidade de automatizar tarefas associadas ao *middleware* de grades, possibilitando a implantação, a configuração e o gerenciamento de recursos e serviços de forma facilitada.

Executar uma aplicação de larga escala – com grande número de recursos e serviços distribuídos – requer que o ambiente seja devidamente implantado e configurado, atendendo, assim, às necessidades da aplicação. Esta tarefa tende a ser exaustiva e sujeita a erros, se apresentando como o elo fraco do ciclo de vida do processo de desenvolvimento de aplicações desta natureza. O problema é agravado pelo desejo de se executar aplicações cada vez mais específicas, que exigem a implantação e a configuração de arcabouços bastante peculiares. Nitidamente, a abordagem manual para realizar tais processos já não é suficiente.

Para lidar com o problema, este artigo propõe AGrADC (*Autonomic Grid Application Deployment & Configuration Architecture*), uma arquitetura para instanciação de aplicações de grade permitindo que a infra-estrutura necessária para sua execução seja implantada, configurada e gerenciada sob demanda. O objetivo é propiciar que o arcabouço de software necessário para a execução de uma aplicação de grade seja instanciado no momento de sua invocação. Na arquitetura proposta, as ferramentas disponibilizadas permitem ao desenvolvedor definir (a) um fluxo de implantação, respeitando dependências entre componentes que compõem a aplicação, (b) parâmetros de configuração e (c) ações a serem executadas diante de situações adversas tais como falhas. O resultado desse processo, materializado na forma de um conjunto de documentos, é repassado a um *motor de instanciação*, que passa a autonomamente conduzir e gerenciar o processo de implantação e configuração.

O presente trabalho possui duas contribuições principais. Primeiro, definiu-se um serviço *inédito* para conduzir o processo de instanciação de aplicações em grades – o motor de instanciação – que incorpora características da Computação Autônoma, sendo capaz de executar procedimentos de contorno para lidar com problemas enfrentados ao longo do processo. Segundo, projetou-se e desenvolveu-se uma arquitetura de software plenamente alinhada com especificação proposta para a área, a CDDLM (*Configuration Description, Deployment, and Lifecycle Management*) [Bell et al. 2005], na qual o referido serviço foi acomodado.

O restante do artigo está organizado da seguinte forma. A Seção 2 aborda os trabalhos relacionados, enquanto a Seção 3 revisa princípios da Computação Autônoma e características de CDDLM. Na Seção 4 apresenta-se uma visão conceitual da AGrADC e descreve-se os elementos que a compõem. Como prova de conceito, a Seção 5 aborda o protótipo desenvolvido. Na Seção 6 é apresentada uma avaliação experimental da arquitetura. O artigo é encerrado na Seção 7 com considerações finais e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

A área de implantação e configuração dinâmicas de serviços e aplicações tem sido alvo de pesquisas recentes, gerando como resultado diversas propostas para lidar com o problema.

Nesta seção apresenta-se trabalhos relacionados, abordando-se, primeiro, implantação e configuração dinâmicas de serviços em geral e na seqüência, de aplicações de grade.

Em [Talwar et al. 2005] é apresentada uma análise comparativa de soluções manuais, baseadas em *scripts*, em linguagens e em modelos para implantação de serviços, constituindo uma referência-chave para mapear a área. Os autores avaliam escalabilidade, complexidade e expressividade das possíveis soluções e concluem que as baseadas em modelos são as mais promissoras por lidarem bem com escalabilidade e complexidade, aspectos estes cruciais para implantação de aplicações distribuídas de grande porte. A implantação dinâmica de serviços também foi investigada em contextos outros que não grades computacionais, como J2EE [Reverbel et al. 2004] e Web Services [Benatallah et al. 2002]. Em [Rauch et al. 2000] foi proposta a clonagem de partições de sistema para implantação de software, abordagem reconhecidamente *cara* computacionalmente por exigir substituição de toda a imagem do sistema operacional. Seguindo na mesma direção, em [Keahey et al. 2005] é proposto o emprego da tecnologia de máquinas virtuais para implantar ambientes com diferentes serviços e configurações.

Os principais trabalhos correlatos focados em grades computacionais são os apresentados em [Sun et al. 2005], [Weissman et al. 2005] e [Qi et al. 2007]. O primeiro propôs uma solução segura para implantação dinâmica de serviços WSRF (*Web Services Resource Framework*). O segundo introduziu uma arquitetura baseada no Tomcat para incorporar a funcionalidade de implantação dinâmica de serviços ao Globus Toolkit versão 3 (GT3), com a restrição de operar em nível de *container*. O terceiro trabalho propôs mudanças no núcleo do Tomcat para tornar sua implementação mais leve e propiciar implantação em nível de *serviço*. Ao ampliar a granularidade de implantação, os autores obtiveram como resultado uma infra-estrutura capaz de lidar com ambientes altamente dinâmicos sem comprometer disponibilidade e capacidade de atendimento de requisições. Em paralelo a esses trabalhos, o GGF (*Global Grid Forum*) tem despendido esforços para padronizar a especificação CDDLM (*Configuration Description, Deployment, and Lifecycle Management*), que constitui uma abordagem baseada em modelos para configuração, implantação e gerenciamento do ciclo de vida de aplicações de grades (descrita na Seção 3.2).

Como pôde ser observado, os avanços realizados na área se concentraram em prover mecanismos para interagir individualmente com estações-alvo, provendo uma forma sistemática (e interoperável, no caso de CDDLM) para instanciar componentes de uma aplicação de grade. Contudo, não abordaram como orquestrar a implantação de uma aplicação completa, envolvendo diversos recursos e serviços distribuídos, tampouco se preocuparam em lidar com situações adversas tais como indisponibilidade de estações selecionadas para implantação e erros de configuração – foco da arquitetura apresentada neste artigo.

3. Referencial Teórico

Esta seção revisa princípios da Computação Autônoma, introduzidos por Horn [Horn 2001]. Na seqüência, aborda-se características da especificação CDDLM.

3.1. Computação Autônoma

A necessidade de gerenciamento de recursos computacionais não é um problema novo para a Ciência da Computação. Por décadas componentes e sistemas têm evoluído muito e, com eles, a complexidade dos sistemas de controle, de compartilhamento de recursos e de gerenciamento das operações. Computação Autônoma surge neste contexto como uma alternativa promissora para reduzir tal complexidade, consistindo na habilidade de um sistema ser mais auto-gerenciável [Ganek e Corbi 2003].

Com conotação biológica, inspirado pelas funcionalidades do sistema nervoso humano, um sistema computacional autônomo incorpora um sub-conjunto de características introduzidas em [Ganek e Corbi 2003, White et al. 2004, Salehie e Tahvildari 2005], que são brevemente descritas a seguir.

1. *Auto-Definição:* capacidade do sistema conhecer a si próprio, podendo ser formado por componentes que possuem essa mesma capacidade.
2. *Auto-Configuração:* capacidade do sistema se adaptar automática e dinamicamente a mudanças do ambiente. Provê características que permitem a inserção de novos componentes com a mínima intervenção humana. Auto-configuração consiste, ainda, na habilidade de sistemas inteiros se auto-configurarem.
3. *Auto-Otimização:* capacidade do sistema maximizar a alocação e a utilização dos recursos para satisfazer as requisições do usuário. Para tal, precisa ser capaz de ajustar seus parâmetros e monitorar seu desempenho.
4. *Auto-Recuperação:* capacidade do sistema detectar, diagnosticar e reparar problemas localizados, resultantes de erros ou falhas, tanto de hardware quanto de software.
5. *Auto-Proteção:* capacidade do sistema antecipar, detectar e recuperar-se de ataques. O objetivo é defender o sistema contra ataques maliciosos ou falhas em cascata ocorridas na tentativa de auto-recuperação.
6. *Sensibilidade ao Contexto:* capacidade do sistema de se adaptar às condições do ambiente onde ele está inserido. É projetado de maneira que possa controlar, gerenciar e alocar seus componentes e recursos de rede de forma a melhor atingir seus objetivos (ex: alto desempenho ou baixo custo).
7. *Interoperabilidade:* capacidade do sistema ser altamente portável para múltiplas plataformas e grande variedade de infra-estruturas de componentes. Para que isso seja possível, independente de desenvolvedor e tecnologia, é necessário que interfaces padrão sejam definidas e utilizadas. Assim, elementos autônomos podem ser contactados e/ou relacionarem-se.
8. *Antecipação:* capacidade do sistema otimizar o uso e a busca por recursos de forma que usuários e aplicações possam utilizá-los em qualquer ambiente, sem a necessidade de requisições específicas e sem aumentar o nível de complexidade para o usuário.

3.2. Configuration Description, Deployment, and Lifecycle Management

A especificação CDDLM define mecanismos para encapsular informações de uma aplicação de grade, que podem ser utilizadas para a sua implantação em uma ou mais estações-alvo de uma infra-estrutura de grade. Dois documentos se destacam nesse contexto: (a) linguagem para descrição da configuração dos componentes de uma aplicação [Bell et al. 2005] e (b) interface para instanciação e gerenciamento destes componentes [Loughran 2005].

A linguagem de descrição de configuração (CDL – *Configuration Description Language*) provê construtores para descrever, seguindo uma estrutura hierárquica, os componentes que compõem uma aplicação e suas propriedades (parâmetros de configuração). CDL permite expressar, também, a ordem em que os componentes devem ser implantandos e configurados. Já a interface de implantação, denominada *Deployment* API, especifica métodos a serem invocados para implantar e configurar os componentes, bem como para gerenciar seu ciclo de vida. *Create(), initialize()* e *run()* são exemplos de métodos fornecidos pela API e que precisam ser implementados por serviços disponíveis em estações da grade habilitadas com o padrão CDDLM.

À medida em que os métodos vão sendo invocados por uma aplicação cliente, o componente sendo implantado muda seu estado, respeitando o fluxo ilustrado na Figura

1. Instanciação e inicialização representam a implantação (*upload*) e a configuração do componente, respectivamente. Quando no estado *running*, o componente encontra-se funcional; ao ocorrer uma falha em qualquer momento do ciclo de vida do componente, seu estado passa a ser *failed*. Por fim, *terminated* é o estado final de um componente. Mudanças bem sucedidas de estado ou problemas enfrentados ao longo do processo dão origem a mensagens de notificação, seguindo um padrão como o WS-Notification [Graham et al. 2005].

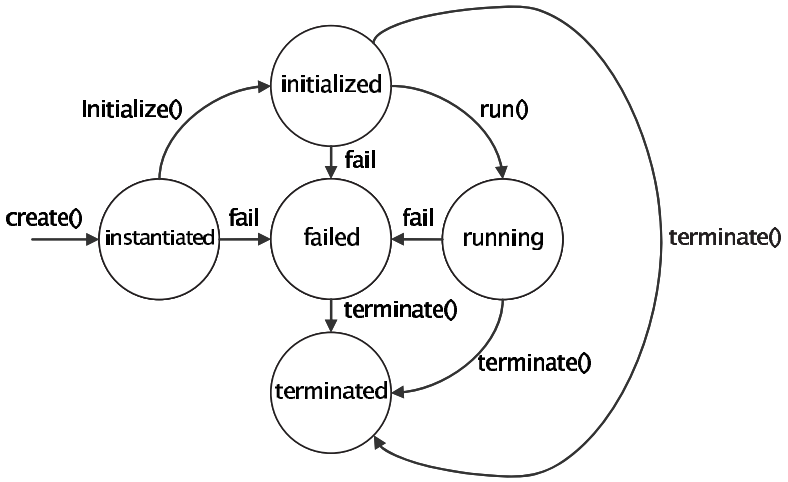

Figura 1. Máquina de estados do ciclo de vida de um componente

4. Arquitetura AGrADC

Esta seção descreve a arquitetura proposta para a instanciação de aplicações de grades computacionais. Inicialmente, aborda-se os elementos da arquitetura e os fluxos de informações existentes entre eles. Em seguida, é apresentada a linguagem de descrição de cenários de instanciação. Por fim, detalha-se o funcionamento do motor de instanciação, enfatizando as funcionalidades fornecidas por ele para prover auto-configuração e auto-recuperação ao processo de instanciação de aplicações.

4.1. Elementos e suas Interações

A AGrADC (*Autonomic Grid Application Deployment & Configuration Architecture*) é composta por quatro elementos: (a) *aplicação de gerenciamento*, (b) *repositório de componentes*, (c) *motor de instanciação* e (d) *serviços de instanciação*. A Figura 2 ilustra uma visão geral da arquitetura instanciada em uma infra-estrutura de grade composta por três domínios administrativos A, B e C.

A aplicação de gerenciamento permite que o desenvolvedor defina os componentes da aplicação, especifique o fluxo de implantação e configuração, bem como requisite a instanciação da aplicação na infra-estrutura de grade. Os componentes e os roteiros de implantação são armazenados em um repositório. O motor de instanciação tem por função receber invocações da aplicação de gerenciamento e orquestrar a instanciação do ambiente de execução solicitado. Por fim, os serviços de instanciação – hospedados em todas as estações da grade – fornecem interfaces que as tornam aptas a executar implantação, configuração e gerenciamento de componentes.

A interação entre os elementos da arquitetura se dá da seguinte forma. Inicialmente, o desenvolvedor define – usando a linguagem CDL – os componentes que fazem parte de sua aplicação (ex: banco de dados, servidor http e *grid service*), a seqüência de implantação

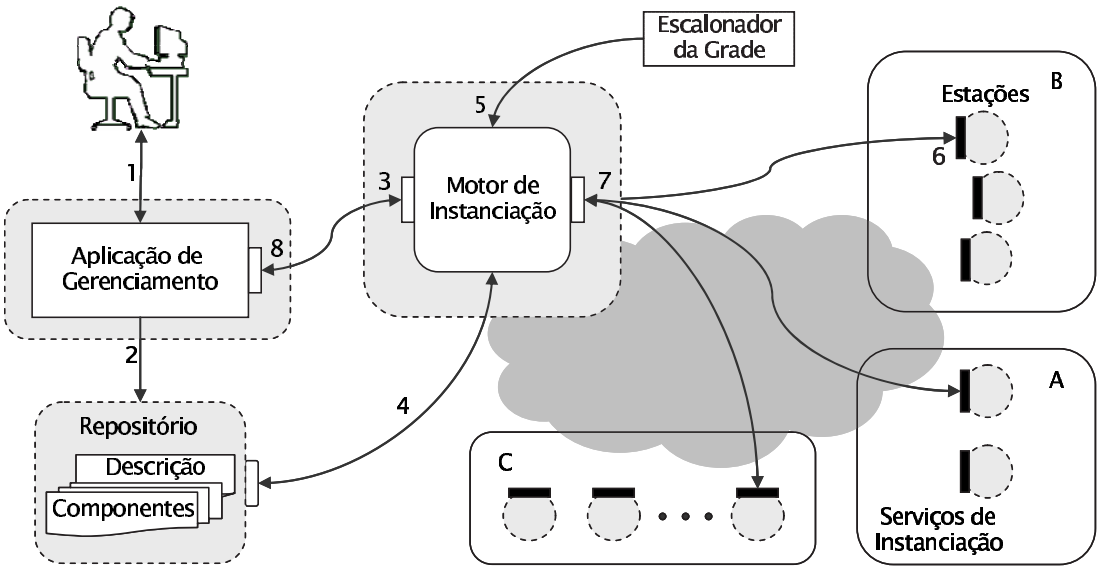

Figura 2. Arquitetura AGrADC e interação entre seus elementos

a ser respeitada e os parâmetros de configuração (fluxo 1 na Figura 2). O resultado desta etapa é a geração de um conjunto de arquivos CDL e componentes, que são armazenados no repositório (fluxo 2). A próxima etapa consiste na solicitação de instanciação da referida aplicação ao motor de instanciação (3), que é acompanhada do identificador da localização do arquivo de descrição da aplicação. Ao receber a solicitação, o motor de instanciação recupera este arquivo (4), interpreta-o e inicia o processo de instanciação.

À medida que o motor identifica os componentes no arquivo de descrição, os mesmos vão sendo recuperados do repositório (fluxo 4 na Figura 2). Com base nas informações fornecidas pelo escalonador da grade (5) em relação aos recursos disponíveis, o motor determina em que estações cada componente será instanciado e interage com os serviços de instanciação das estações escolhidas (6). A interação prevê operações para implantar, configurar e gerenciar os componentes. O resultado dessas operações – sucesso ou falha – é informado ao motor mediante notificações geradas pelos serviços de instanciação (7). A partir de políticas expressas junto ao motor (explicadas na Seção 4.3), o mesmo reage autonomamente avançando o processo de instanciação ou executando um procedimento de contorno. Por fim, ao término do processo de instanciação, o motor notifica a aplicação de gerenciamento sobre o resultado (8). Neste momento, se o processo resultou bem sucedido, o ambiente de execução da grade está pronto para executar a aplicação.

A representação de cenários de instanciação é crucial para a arquitetura, considerando a diversidade de aplicações e, sobretudo, de requisitos de execução demandados pelas mesmas. Sem os cenários (e a infra-estrutura para instanciá-los dinamicamente), as aplicações ficariam restritas aos serviços disponíveis previamente nas estações da grade ou exigiriam um grande esforço operacional (e *ad-hoc*) para preparar o ambiente desejado. A próxima seção aborda as funcionalidades providas pela linguagem CDL para especificação dos referidos cenários.

4.2. Representação de Cenários de Instanciação

A descrição de um cenário de instanciação é realizada com o uso da linguagem de descrição de configuração (CDL) e consiste na identificação dos componentes e seus parâmetros de configuração, bem como na determinação do fluxo de implantação a ser respeitado.

Antes de detalhar as funcionalidades da linguagem, um exemplo de cenário é apresentado gráfica e textualmente nas Figuras 3 e 4, respectivamente. O cenário é composto por três componentes: (a) base de dados (*Hsqldb*); (b) servidor de aplicações Java para *web* (*Tomcat*); e (c) aplicação de grade (*GridApp*). O exemplo ilustra as dependências entre esses componentes, indicando que *Hsqldb* e *Tomcat* podem ser instanciados em paralelo, e somente ao término desse passo *GridApp* pode ser instanciada.

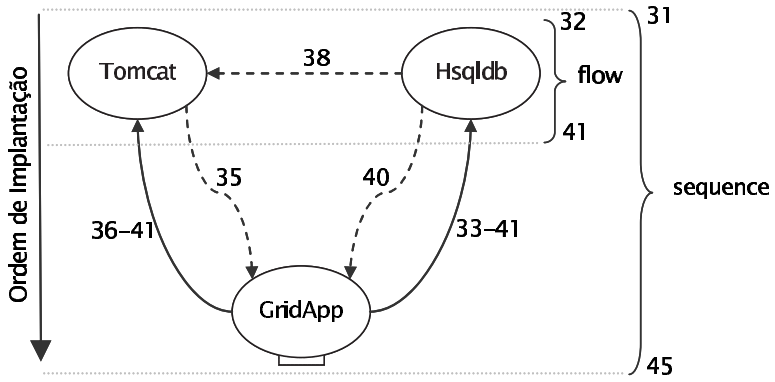

Figura 3. Representação gráfica de um cenário de instanciação

A representação gráfica ilustra o grafo de dependências entre os componentes, onde as setas contínuas indicam a ordem de implantação e as pontilhadas indicam eventos ou referências a parâmetros de configuração. A especificação textual, em CDL, define os componentes que compõem a aplicação de grade, os seus parâmetros e o fluxo de instanciação do cenário. A especificação é organizada em dois blocos: (a) definição e configuração de componentes (`cdl:configuration`) e (b) definição da ordem de instanciação dos componentes (`cdl:system`), explicados a seguir.

No bloco de definição e configuração (Figura 4, linhas 4 a 28) são definidos os componentes da aplicação de grade, os quais serão associados, posteriormente, aos recursos disponíveis. Os componentes são definidos em blocos XML e a cada um deles são associados os parâmetros de configuração. Além de representar serviços (ex: *Tomcat*, linhas 10 a 15, e *Hsqldb*, linhas 16 a 21), os componentes podem representar parâmetros a serem compartilhados por outros componentes, como é o caso de *DBConnection*, linhas 5 a 9.

O número de parâmetros de cada um dos componentes é peculiar às suas necessidades. São três as maneiras de associar valores para cada um dos parâmetros dos componentes: (a) definir o valor no momento da declaração do parâmetro (ex: linha 13, `<port> 8080 </port>`); (b) informar que o valor será definido no momento da instanciação (ex: linha 12, `<hostname cdl:lazy="true"/>`); ou (c) determinar que o valor do parâmetro deve ser aquele associado à referência indicada (ex: linha 38, `<dbport cdl:ref="Hsqldb:/port"/>`). Neste último exemplo, determina-se que o componente *Tomcat* herdará do componente *Hsqldb* a configuração da porta em que o banco de dados estará disponível.

No bloco de definição da ordem de instanciação dos componentes, a ordenação é determinada pela hierarquia com que os componentes são declarados no bloco `cdl:system` (Figura 4, linhas 30 a 51). A hierarquia entre os componentes é estabelecida através das seguintes regras.

- *Sequence*: indica que os componentes devem ser instanciados de acordo com a ordem léxica. Por exemplo, nas linhas 31 a 45 é especificado que a instanciação dos

componentes *Tomcat* e *Hsqldb* deve ser finalizada antes do processo de instanciação de *GridApp* ser iniciado.

- *Inverse*: operação inversa da seqüência, é empregada para indicar a ordem de remoção dos componentes em um processo de *Undeploy* (linhas 46 a 50).
- *Flow*: é utilizada quando não existe necessidade de ordem para a instanciação entre componentes, ou seja, não existem dependências. Na Figura 4, linhas 32 a 41, *Tomcat* e *Hsqldb* podem ser instanciados em qualquer ordem.
- *Switch*: permite que o fluxo de instanciação seja alterado, dependendo de valores associados a variáveis de condição (não usado no exemplo).

```
1  <?xml verson="1.0" encoding="UTF-8"?>
2  <cdl:cdl xmlns="http://cddlm.org/gridapp">
3      ...
4      <cdl:configuration>
5          <DBConnection>
6              <hostname />
7              <dbport />
8              . . .
9          </DBConnection>
10         <Tomcat cdl:extends="c:Component">
11             <data>http://cddlm.unisinos.br/repository/Tomcat-5.0.gar</data>
12             <hostname cdl:lazy="true" />
13             <port>8080</port>
14             <dbconnection cdl:extends="DBConnection" />
15         </Tomcat>
16         <Hsqldb cdl:extends="c:Component">
17             <data>http://cddlm.unisinos.br/repository/Hsqldb.gar</data>
18             <hostname cdl:lazy="true" />
19             <port>3306</port>
20             . . .
21         </Hsqldb>
22         <GridApp>
23             <t:application>http://cddlm.unisinos.br/repository/GridApp.gar</t:application>
24             <t:applicationPath>/GridApp</t:applicationPath>
25             <t:dbname>jdbc/Hsqldb</t:dbname>
26             <t:hostname />
27         </GridApp>
28     </cdl:configuration>
29
30     <cdl:system>
31         <cmp:sequence lifecycle="initialization">
32             <cmp:flow lifecycle="initialization">
33                 <Hsqldb>
34                     <hostname cdl:lazy="true" />
35                 </Hsqldb>
36                 <Tomcat>
37                     <hostname cdl:lazy="true" />
38                     <dbport cdl:ref="Hsqldb:/port" />
39                     . . .
40                 </Tomcat>
41             </cmp:flow>
42             <GridApp>
43                 <hostname cdl:lazy="true" />
44             </GridApp>
45         </cmp:sequence>
46         <cmp:reverse lifecycle="terminate">
47             <GridApp />
48             <Tomcat />
49             <Hsqldb />
50         </cmp:reverse>
51     </cdl:system>
52 </cdl:cdl>
```

Figura 4. Representação textual de um cenário de instanciação

4.3. Motor de Instanciação

O motor de instanciação consiste em um serviço responsável por orquestrar a instanciação da infra-estrutura de software necessária para executar aplicações de grade. O motor

dispensa a intervenção humana no processo de implantação, configuração e gerenciamento dos componentes que compõem as aplicações, suprindo uma lacuna crítica até então não abordada em trabalhos anteriores.

Invocado pela aplicação de gerenciamento, o motor oferece dois métodos: *Deploy* e *Undeploy*. Acompanha a invocação do primeiro o identificador da localização do arquivo de descrição da aplicação a ser implantada, enquanto o segundo método é seguido da EPR (*End-Point Reference*) da aplicação a ser removida. No outro extremo, o motor de instanciação lança mão dos métodos fornecidos pela *Deployment* API, introduzida na Seção 3.2, para interagir com os serviços de instanciação.

A instalação de uma aplicação consiste na instanciação dos seus componentes, respeitando a ordem informada no arquivo de descrição. A instanciação de cada componente, por sua vez, é executada através dos passos descritos na Seção 3.2 e ilustrados na Figura 1. Para conduzir, de maneira controlada, esse processo complexo, o motor mantém uma máquina de estados para cada componente, que reflete a situação atual da instanciação de cada um deles.

A Figura 5 ilustra um *snapshot* do processo de instanciação da aplicação introduzida anteriormente na Seção 4.2. Observe que três máquinas de estados são utilizadas, sendo que aquela que controla a instanciação do componente *Hsqldb* aparece em destaque. O referido componente já foi carregado (*uploaded*) na estação-alvo e configurado. No momento o motor acaba de invocar o método *run()* e recebe, de forma assíncrona, uma mensagem de notificação do tipo *RunFault* informando que a operação não foi bem sucedida. Concomitantemente à tentativa de instanciação do componente *Hsqldb*, o motor está executando o mesmo procedimento para o componente *Tomcat*, já que a especificação CDL autoriza tal paralelismo.

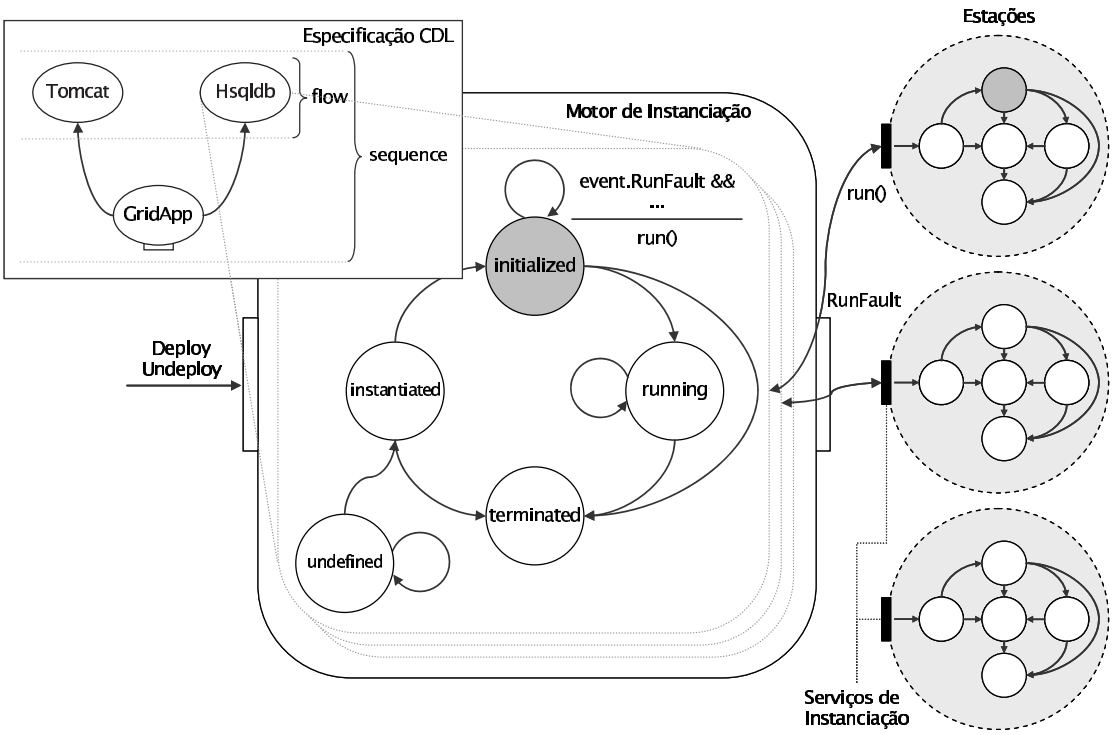

Figura 5. Máquina de estados do motor de instanciação

Os estados em que se encontram os componentes e as mensagens de notificação recebidas podem ser usados para expressar como o motor de instanciação deve se comportar diante de situações adversas. Tal comportamento – especificado pelo desenvolvedor da aplicação ou pelo gerente da infra-estrutura de grade mediante *políticas de ação* [Kephart e Walsh 2004] – confere à arquitetura características da Computação Autônoma tais como auto-configuração e auto-recuperação.

Políticas de ação ditam ações que devem ser executadas sempre que o sistema se encontra em determinado estado, sendo representadas na forma *ON (Estado) IF (Condição) THEN (Ação)*. Para que o motor de instanciação apresente comportamento racional, as políticas devem cobrir cada estado relevante definindo ações de acordo com condições pré-estabelecidas. A Figura 6 ilustra um conjunto de políticas definidas para reger o comportamento do motor.

```
1  ON UndefinedState {
2
3    IF (event.NonAvailableHost)
4    THEN terminate()
5
6    IF (event.HostUnreachable && event.HostUnreachable:ContFailed < 3)
7    THEN create()
8
9    IF (event.HostUnreachable && event.HostUnreachable:ContFailed >= 3)
10   THEN terminate()
11
12   IF (event.SoapFault && event.SoapFault:ContFailed < 3)
13   THEN create()
14
15   IF (event.SoapFault && event.SoapFault:ContFailed >= 3)
16   THEN terminate()
17
18   ...
19 }
20
21 ON InstantiatedState {
22
23   IF (event.ConfigurationFault && event.ConfigurationFault:ContFailed < 3)
24   THEN initialize()
25
26   IF (event.ConfigurationFault && event.ConfigurationFault:ContFailed >= 3)
27   THEN terminate()
28
29   ...
30 }
31
32 ON InitializedState {
33
34   IF (event.RunFault && event.RunFault:ContFailed < 3)
35   THEN run()
36
37   IF (event.RunFault && event.RunFault:ContFailed >= 3)
38   THEN terminate()
39
40   ...
41 }
42
43 ON RunningState {
44
45   IF (event.TestFault && event.TestFault:ContFailed < 3)
46   THEN ping()
47
48   IF (event.TestFault && event.TestFault:ContFailed >= 3)
49   THEN terminate()
50
51   ...
52 }
```

Figura 6. Representação de políticas

As políticas ilustradas cobrem os quatro principais estados do ciclo de vida de um componente: *undefined* (linhas 1 a 19), *instantiated* (linhas 21 a 30), *initialized* (linhas 32 a 41) e *running* (linhas 43 a 52). Por exemplo, quando o motor procura implantar um componente (estado *undefined*), algumas situações podem ocorrer:

- o escalonador pode não ter estações a oferecer (*NonAvailableHost*), condição que faz com que o motor interrompa a implantação do componente (*terminate()*) e, por conseqüência, a instanciação de toda a aplicação;
- o motor pode não conseguir contactar a estação determinada pelo escalonador (*HostUnreachable*), condição que leva o motor a tentar implantar o componente novamente na mesma ou em uma estação alternativa;
- o motor pode enfrentar dificuldades na interação com o serviço de instanciação que está executando na estação (*SoapFault*), condição que induz o motor a repetir a tentativa.

Observe que o estado que deve ser alcançado pela execução de uma determinada ação não é explicitamente especificado em políticas de ação [Kephart e Walsh 2004]. Assume-se que o autor das políticas o conhece. No caso do motor de instanciação, a execução bem sucedida de uma ação faz com que a instanciação do componente avance ao estado seguinte, considerando a máquina de estados apresentada na Figura 5. Por outro lado, execuções mal sucedidas de uma ação fazem com que ou não haja avanço de estado (no caso de tentativas de procedimentos de contorno) ou o componente avance para o estado *terminated*.

5. Implementação

Um protótipo da arquitetura proposta foi desenvolvido com o objetivo de avaliar sua viabilidade técnica. Esta seção descreve aspectos relacionados com a implementação desse protótipo, incluindo informações sobre cada um dos elementos que compõem a arquitetura.

A aplicação de gerenciamento foi implementada na linguagem Java, permitindo descrever os componentes que compõem uma aplicação de grade (e suas dependências) e verificar, com o apoio da API JDOM, a correção dos arquivos de descrição XML-CDL [Tatemura 2005] resultantes. A aplicação de gerenciamento permite, ainda, gerar pacotes GAR (*Grid Archive*) de componentes – formato compatível com o Globus Toolkit versão 4 [Globus 2006], identificar o motor de instanciação, bem como invocar e acompanhar a instanciação de uma aplicação. Os arquivos de descrição e os componentes são armazenados em um servidor de aplicações *web* (*Apache Tomcat*).

Os serviços de instanciação são serviços *web* que implementam as interfaces e os mecanismos previstos na especificação CDDLM, introduzidos na Seção 3.2. Atualmente encontram-se em desenvolvimento três implementações de referência deste serviço: uma no contexto do projeto BizGrid (por Fujitsu, NEC e Hitachi), uma pela empresa Softricity e outra em um esforço conjunto entre a HP e a UFCG. Como o foco deste trabalho foi investigar técnicas para orquestrar a instanciação de aplicações completas, envolvendo recursos e serviços distribuídos, optou-se por não repetir esforços já em andamento, sem no entanto abrir mão de propor solução alinhada com a referida especificação e os serviços por vir. Nesse contexto, adotou-se temporariamente o serviço de implantação fornecido pelo Globus Toolkit versão 4, acrescido de funcionalidades que permitem a implantação, a configuração e o gerenciamento de componentes emulando a especificação CDDLM.

Por fim, o motor de instanciação tem como característica principal a conformidade com a especificação CDDLM. Também desenvolvido em Java, o motor é um serviço *web* que implementa a máquina de estados ilustrada na Figura 5. A interface oferecida à aplicação de gerenciamento para invocar e acompanhar a instanciação

de aplicações segue a especificação WSDM (*Web Services Distributed Management*) [Bullard e Vambenepe 2005] e foi desenvolvida usando o *framework* Muse [Muse 2006], versão 2.0, da *Apache Software Foundation*. A API JDOM é empregada para ler, interpretar e armazenar em memória as descrições das aplicações. Para interagir com os serviços de instanciação, a atual versão do protótipo utiliza a API de implantação fornecida pelo Globus Toolkit versão 4 – e não a *Deployment* API – pelas razões recém mencionadas. O motor implementa, ainda, um *listener* para receber mensagens de notificação geradas pelos serviços de instanciação.

As políticas de ação são representadas no formato descrito na Seção 4.3 e armazenadas em arquivo de configuração externamente ao motor. Esta característica permite que políticas sejam criadas, modificadas e removidas dinamicamente sem a necessidade de recompilar o motor de instanciação.

6. Avaliação Experimental

Esta seção apresenta a avaliação experimental realizada com o protótipo da arquitetura. AGrADC foi implantada em um ambiente real de grade (executando Globus versão 4) com o objetivo de avaliar sua capacidade para (a) instanciar aplicações sob demanda, seguindo rigorosamente especificações de dependência e configuração, e (b) reagir autonomamente diante de situações adversas geradas sinteticamente.

A aplicação utilizada para avaliar a arquitetura foi baseada naquela ilustrada na Figura 3. O arquivo de descrição da aplicação foi adaptado determinando a instanciação do componente *Hsqldb* em uma estação e a dos componentes *Tomcat* e *GridApp* em três outras estações distintas. *GridApp* consiste em uma aplicação do tipo *bag-of-tasks* para efetuar computações sobre um modelo determinístico de dinâmica intracelular de um vírus [Srivastava et al. 2002]. No cenário modelado, as três instâncias de *GridApp* foram configuradas para utilizar a mesma base de dados *Hsqldb*. Para executar o experimento foram utilizadas quatro estações para hospedar a aplicação, mais uma estação onde foram instalados a aplicação de gerenciamento, o repositório de componentes e o motor de instanciação.

Na seqüência, invocou-se a partir da aplicação de gerenciamento a instanciação da aplicação definida. Ao longo do processo observou-se que a arquitetura comportou-se exatamente como o esperado, respeitando o que foi especificado no arquivo de configuração e concluindo com sucesso a instanciação. Este acompanhamento ocorreu mediante (a) captura e análise do tráfego gerado, (b) observação de entradas em arquivos de *log*, (c) averiguação de componentes instalados e (d) invocação bem sucedida da aplicação.

Em um segundo momento, relaxou-se a descrição da aplicação permitindo que a instanciação da aplicação fosse realizada em estações recomendadas pelo escalonador da grade (usando-se o construtor `lazy`). O escalonador, por sua vez, foi configurado para sugerir uma seqüência conhecida de estações. A primeira e a terceira da lista eram estações inexistentes. Ao invocar, pela segunda vez, a instanciação da aplicação foi possível observar a arquitetura fazendo cumprir as políticas definidas para reger o comportamento do motor. Ao receber mensagens de notificação *HostUnreachable*, o motor autonomamente solicitou ao escalonador novas recomendações de estações e, mesmo diante dessas adversidades, concluiu a instanciação da aplicação de forma bem sucedida.

7. Conclusões e Trabalhos Futuros

Este artigo apresentou uma arquitetura para implantação, configuração e gerenciamento do ciclo de vida de aplicações de grades computacionais. Enquanto os trabalhos relacionados têm se concentrado na investigação de técnicas para permitir a carga e a configuração remota de componentes em estações de grade [Sun et al. 2005, Weissman et al. 2005,

Qi et al. 2007], este trabalho assume tal problemática como (próxima de) resolvida e avança na direção de uma solução para orquestrar o processo de instanciação de aplicações de grade complexas. Para tal, propõe um serviço *inédito* que assume o papel até agora desempenhado por operadores humanos. Ao permitir a especificação de políticas para regular seu comportamento, o serviço é capaz de executar procedimentos de contorno para lidar com problemas enfrentados ao longo do processo de instanciação, imprimindo suporte à auto-configuração e auto-recuperação.

Salienta-se que o foco deste trabalho reside na instalação e na configuração da infra-estrutura de software necessária para implantar uma aplicação, incluindo os binários desta aplicação. A instanciação (ou invocação) da aplicação propriamente dita não faz parte do escopo e, portanto, não foi tratada. Para realizar esta tarefa, é possível utilizar ferramentas de *workflow* para grades, que têm por função invocar, de maneira coordenada, os componentes da aplicação.

Os resultados obtidos apontam que a arquitetura proposta pode ser aplicada em situações reais, fato comprovado pela implementação de um protótipo capaz de conduzir a instanciação de uma aplicação real. Tão logo sejam finalizadas as implementações de referência da especificação CDDLM para executar nas estações-alvo – o que deve se concretizar nos próximos meses – a arquitetura, com ajustes mínimos, estará habilitada para operar em total conformidade com a especificação. Esta característica, apesar de não ser mandatória, é importante para promover interoperabilidade entre soluções desenvolvidas na área de grades computacionais.

Como trabalhos futuros pretende-se realizar um conjunto mais extensivo de experimentos para caracterizar as limitações do emprego da arquitetura para instanciar aplicações de mais larga escala, bem como determinar os tempos envolvidos nesse processo. Pretende-se, ainda, explorar outras alternativas para representação de comportamento autônomo, tais como *políticas de objetivo* e *funções de utilidade*, visando prover um formalismo de mais alto nível para o desenvolvedor da aplicação ou gerente da infra-estrutura de grade.

Referências

Bell, D., Kojo, T., Goldsack, P., Loughran, S., Milojicic, D., Schaefer, S., Tatemura, J. e Toft, P. (2005). Configuration Description, Deployment, and Lifecycle Management (CDDLM) Foundation Document. GGF. http://www.gridforum.org/documents/GFD.50.pdf.

Benatallah, B., Dumas, M., Sheng, Q. Z. e Ngu, A. H. H. (2002). Declarative Provisioning and Peer-to-Peer Provisioning of Dynamic Web Services. In *IEEE International Conference on Data Engineering (ICDE 2002)*, p. 297–308.

Bullard, V. e Vambenepe, W. (2005). Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1. Standard. OASIS. http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf.

Ganek, A. e Corbi, T. (2003). The Dawning of the Autonomic Computing Era. *IBM Systems Journal*, 42(1):5–18.

Globus (2006). Globus Toolkit Project Home Page. http://www.globus.org.

Graham, S., Hull, D. e Murray, B. (2005). Web Services Base Notification 1.3 (WS-BaseNotification). Standard. OASIS. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.htm.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/.

Keahey, K., Foster, I., Freeman, T., Zhang, X. e Galron, D. (2005). Virtual Workspaces in the Grid. In *International Euro-Par Conference (Euro-Par 2005)*, p. 421–431.

Kephart, J. O. e Walsh, W. E. (2004). An Artificial Intelligence Perspective on Autonomic Computing Policies. In *IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, p. 3–12.

Kesselman, C. e Foster, I. (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

Loughran, S. (2005). Configuration Description, Deployment, and Lifecycle Management. CDDLM Deployment API. Draft 2005-02-25. GGF. http://xml.coverpages.org/CDDML-Deployment-API-SpecificationDraft20050308.pdf.

Muse (2006). Muse Project Home Page. http://ws.apache.org/muse/.

Nemeth, Z. e Sunderam, V. (2002). A Formal Framework for Defining Grid Systems. In *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, p. 188–197.

Qi, L., Jin, H., Foster, I. e Gawor, J. (2007). HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4. In *Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2007)*.

Rauch, F., Kurmann, C. e Stricker, T. M. (2000). Partition Repositories for Partition Clonning – OS Independent Software Maintenance in Large Clusters of PCs. In *IEEE International Conference on Cluster Computing*, p. 233–242.

Reverbel, F., Burke, B. e Fleury, M. (2004). Dynamic Deployment of IIOP-Enabled Components in the JBoss Server. In *Component Deployment: Second International Working Conference (CD 2004)*, p. 65–80.

Salehie, M. e Tahvildari, L. (2005). Autonomic Computing: Emerging Trends and Open Problems. In *Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005)*, volume 30, p. 1–7.

Simmons, B. e Lutfiyya, H. (2005). Policies, Grids and Autonomic Computing. In *Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005)*, p. 1–5.

Srivastava, R., You, L., Summers, J. e Yin, J. (2002). Stochastic vs. Deterministic Modeling of Intracellular Viral Kinetics. *Journal of Theoretical Biology*, 218(3):309–321.

Sun, H., Zhu, Y., Hu, C., Huai, J., Liu, Y. e Li, J. (2005). Early Experience of Remote and Hot Service Deployment with Trustworthiness in CROWN Grid. In *International Workshop on Advanced Parallel Processing Technologies (APPT 2005)*, p. 301–312.

Talwar, V., Milojicic, D., Wu, Q., Pu, C., Yan, W. e Jung, G. (2005). Approaches for Service Deployment. *IEEE Internet Computing*, 9(2):70–80.

Tatemura, J. (2005). Configuration Description, Deployment, and Lifecycle Management. XML Configuration Description Language Specification Version 1.0. Draft 03-05-2005. GGF. https://forge.gridforum.org/.

Weissman, J., Kim, S. e England, D. (2005). A Framework for Dynamic Service Adaptation in the Grid: Next Generation Software Program Progress Report. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)*.

White, S. R., Hanson, J. E., Whalley, I., Chess, D. M. e Kephart, J. O. (2004). An Architectural Approach to Autonomic Computing. In *International Conference on Autonomic Computing (ICAC 2004)*, p. 2–9.