Grid Anywhere: Um Middleware Extensível para Grades Computacionais Desktop

Teixeira, F.C., Santana, M.J.S., Santana, R.H.C.

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
São Carlos – SP – Brasil
LASDPC - Laboratório de Sistemas Distribuídos e Programação Concorrente

teixeira@icmc.usp.br, mjs@icmc.usp.br, rcs@icmc.usp.br

Resumo: Este artigo apresenta a proposta e implementação de um *middleware* para grades computacionais *desktop* que permite que novas arquiteturas para esse paradigma de computação distribuida sejam compostas de uma maneira mais simples e rápida por meio de módulos que são acoplados em um núcleo comum. Além do *middleware* principal, são descritas soluções para gerenciamento de segurança, carregamento de classes e hospedagem de objetos remotos. Como aplicações desse *middleware* foram desenvolvidas duas arquiteturas de grade computacional *desktop*. A primeira viabiliza a utilização de computadores pessoais ociosos como infraestrutura para um ambiente de oferta de plataforma como serviços (PaaS). A segunda arquitetura permite a utilização de receptores (*set-top boxes*) do sistema de televisão digital interativa como provedores de recursos para um ambiente de processamento paralelo.

Palavras chave: Computação em grade, computação em nuvem, *sandbox*, carregamento de classes, SOAP, TVDI.

1. Introdução

Em meados da década de 90 foi criado o termo grade computacional para denotar um paradigma de computação distribuída que tem como foco o compartilhamento de recursos entre participantes geograficamente distribuídos e sob administrações independentes [1][2]. Tradicionalmente essa abordagem visa o compartilhamento de recursos como *clusters*, supercomputadores, unidades de armazenamento, entre outros. Nesse tipo de sistema distribuído os participantes podem atuar tanto como provedores como consumidores de recursos. No entanto, o comportamento egoísta dos participantes motiva a formação de um ambiente com poucos provedores e muitos consumidores de recursos [3].

Algum tempo depois foi observado que computadores pessoais espalhados por todo mundo também poderiam ser utilizados para compor um ambiente de processamento de alto desempenho, constituindo as grades computacionais *desktop* [4][5]. Esse tipo de grade computacional consiste em um modelo mestre/trabalhador onde uma instituição de pesquisa, por exemplo, coordena centenas ou milhares de computadores pessoais que realizam o processamento de suas tarefas que são normalmente *Bag-of-Tasks*. Nessa abordagem são encontrados muitos provedores de recursos e poucos consumidores.

Esse trabalho de doutorado investigou as grades computacionais *desktop* e verificou que esse modelo de processamento pode ser estendido para abrigar novos tipos de consumidores e provedores de recursos. Para isso, o *middleware Grid Anywhere* foi proposto e implementado para permitir que novos cenários de computação distribuída possam ser compostos de uma maneira mais simples e rápida.

A segunda seção desse artigo apresenta o *Sam Dog*, um gerenciador de segurança que permite que políticas de segurança possam ser herdades e especificadas em um ambiente hierárquico. A seção 3 descreve o WSBCL (*Web Services Based Classloader*) que é um carregador de classes Java totalmente baseado em serviços *Web* que permite que o processo de carga de classes possa ser realizado mesmo sob restrições de conectividade e baixa potência computacional.

Na seção 5 é descrito o Sesiom, o *middleware* responsável pela hospedagem dos objetos e chamadas de métodos remotos em um ambiente onde é possível definir a forma de transporte das mensagens SOAP por meio de módulos acopláveis.

A arquitetura básica do *Grid Anywhere* é apresentada na seção 5, onde são detalhados seu núcleo estático e os módulos dinâmicos que podem ser acoplados para compor novas arquiteturas de grades computacionais.

Na seção 6 são demonstradas duas aplicações do *Grid Anywhere* para formar novas arquiteturas de grades computacionais. A primeira permite que computadores pessoais ociosos possam ser utilizados como infraestrutura para oferecer plataforma como serviço (PaaS) [6] para consumidores convencionais (domésticos ou corporativos) que necessitam de uma maior potência computacional para executar suas aplicações.

A segunda arquitetura permite que receptores (*set-top boxes*) do sistema de televisão digital interativa possam ser utilizados como provedores de recursos para processamento paralelo. Para isso, objetos Java e mensagens SOAP são enviados a uma emissora de TV que os propaga via *broadcasting* a todos os equipamentos sintonizados a ela. Ao término da execução dos métodos remotos as mensagens SOAP de resposta são enviadas por meio de um canal de retorno [7].

Por fim, na seção 7 são apresentadas as conclusões e trabalhos futuros.

2. Sam Dog

Em uma aplicação computacional é de fundamental importância que se possa definir "o que" pode ser feito e "por quem". Esse tipo de controle garante que pessoas não autorizadas tenham acesso a informações confidenciais e também não permitem que usuários maliciosos realizem operações que possam comprometer o bom funcionamento de um sistema computacional.

O gerenciamento das listas de permissões de acesso não é algo trivial quando um grande número de recursos e usuários é encontrado.

O *Sam Dog* (nome inspirado no cão pastor dos desenhos animados) é um gerenciador de segurança que se baseia em um modelo onde as permissões são organizadas em forma de cascata. Dessa maneira, as políticas definidas em um gerenciador são automaticamente herdadas pelos gerenciadores subordinados.

Gerenciadores de Domínio

O *Sam Dog* é composto primeiramente por um componente chamado Gerenciador de Domínio que é responsável por determinar e fornecer as informações de acesso referentes a um domínio de aplicação (esse domínio não possui vínculo com domínios da internet ou qualquer outro encontrado em outros sistemas, sendo totalmente independente). Um domínio de aplicação pode ser criado, por exemplo, para gerenciar os computadores pessoais encontrados em uma determinada universidade.

Tais domínios podem ser organizados de uma maneira hierárquica, de forma que cada domínio esteja ligado a um único superior e a vários subdomínios. A figura 1 ilustra o exemplo da existência de três domínios de segurança: USP, ICMC e LASDPC.

De Maneira a facilitar a identificação da organização hierárquica, esses domínios são organizados em um espaço de nomes (semelhante àquele utilizado nas classes e pacotes Java) que é formado em função de sua posição na hierarquia. Sendo assim, na ordem que aparecem na figura, os domínios teriam os seguintes identificadores: *usp.icmc.lasdpc*, *usp.icmc* e *usp*.

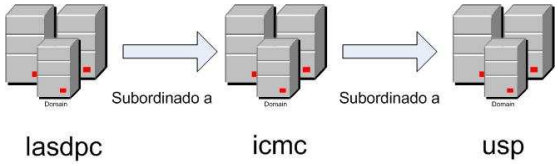

Fig. 1: Relacionamento entre os Gerenciadores de Segurança

Cada um dos domínios possui um administrador independente e, por conseqüência, uma política de segurança individual que são herdadas pelos subordinados. No momento de processar as regras, primeiramente observa-se aquelas definidas no domínio onde a ação está sendo requisitada e, caso nenhuma regra seja encontrada, são analisadas as políticas dos domínios superiores, sempre na ordem hierárquica inversa. Essa é a abordagem das listas de acesso em cascata que permite que regras sejam herdadas e especializadas, quando necessário.

Grupos e Entidades

Um domínio pode possuir subdomínios, entidades e grupos de entidades. Uma entidade é qualquer indivíduo que possa requisitar uma ação junto ao sistema computacional e um grupo é um conjunto de entidades.

Quando uma entidade ou grupo é criado no gerenciador de domínio seu identificador segue o mesmo padrão adotado para os subdomínios. Por exemplo, se dois grupo chamados "*grid*" e "*cloud*" fossem criados dentro do domínio *usp.icmc.lasdpc*, seus identificadores seriam, respectivamente, *usp.icmc.lasdpc.grid* e *usp.icmc.lasdpc.cloud*. Por último, se duas entidade chamadas "*m001*" e "*m002*" tiverem suas criações realizadas nesse

domínio e inseridas no grupo "*grid*" suas identificações seriam *usp.icmc.lasdpc.grid.m001* e *usp.icmc.lasdpc.grid.m002*. A Figura 2 ilustra essa configuração de domínio, grupo e usuário apresentada no exemplo.

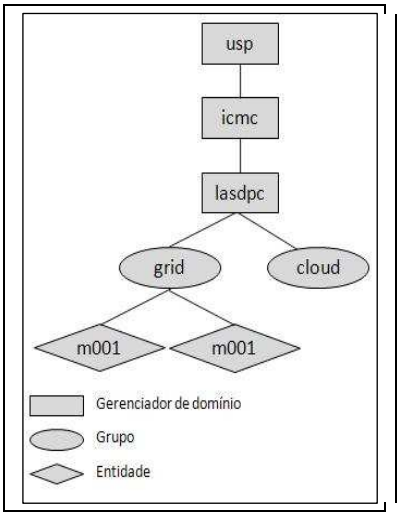

Fig. 2: Gerenciadores de Segurança, Grupos e Entidades

Uma permissão ou negação de acesso pode ser inserida em qualquer ponto do espaço de nomes das entidades, prevalecendo sempre aquela mais próxima do nível inferior. Como exemplo, deseja-se que todos os computadores (mais de 50) do grupo de pesquisa "*grid*" tenham acesso ao recurso "*sfile*", exceto a entidade "*m001*". Para isso, ao invés de criar uma regra para cada entidade, define-se uma permissão para o grupo todo e uma negação somente para a máquina desejada.

De uma maneira mais formal, considere a diretiva *rule(tipo, entidade, tarefa)*, onde *tipo* define se o acesso é concedido (*accept*) ou negado (*reject*), *entidade* define a quem a regra se aplica e *tarefa* identifica para qual tarefa a regra é aplicada.

Retomando ao exemplo dos computadores do domínio Lasdpc, seria preciso definir as seguintes regras na política de segurança:

```
rule(accept, usp.icmc.lasdpc, sfile);
rule(reject, usp.icmc.lasdpc.m001, sfile);
```

Tarefas

Quando se fala em controle de acesso três tópicos são de fundamental importância: (1) Quem está autorizando (2) o que para (3) quem.

Os itens 1 e dois já foram discutidos nesse artigo quando foram mencionados os gerenciadores de domínio e entidades.

De maneira semelhante às entidades, as tarefas também podem ser organizadas de uma maneira hierárquica. Como exemplo é possível imaginar a situação em um sistema computacional distribuído onde é possível ter as seguintes operações: recebimento de conexão via *socket*, requisição de conexão via *socket*. Essas duas operações podem ser agrupadas em um conjunto chamado conexões via *socket*.

Nesse sentido, é possível negar um grupo inteiro de tarefas e liberar apenas alguma tarefa específica. Se em um sistema complexo houver um número grande de tarefas esse tipo de política torna o processo de gerenciamento mais simples.

Utilização de Contexto para Definição de Regras

Já foram discutidas as entidades e as tarefas que determinam, respectivamente, "quem" pode fazer "o que". No entanto, é possível que as requisições de tarefas possam ser liberadas ou bloqueadas para uma determinada entidade em função do contexto em que a requisição está sendo realizada. Sendo assim, passa-se a determinar "quem" pode fazer "o que" e "quando".

Para isso, no momento da requisição de uma tarefa é fornecido um conjunto de variáveis que determina o contexto no qual a execução está ocorrendo. Em função desse contexto são estabelecidas as regras para determinar se a requisição deve ser aceita ou rejeitada.

A definição da regra utiliza o formato padrão para imposição de uma condição: IF/ELSE. Portanto, o administrador cria uma regra por meio de uma proposição que faz uso de operadores relacionais e lógicos para determinar a ação a ser tomada em função das variáveis de contexto.

Descrição de Regras

As regras existentes em uma política de segurança são armazenadas em um documento XML. Nesse documento são descritas as proposições que são analisadas pelo motor de processamento do *Sam Dog* para determinar a ação a ser tomada.

A implementação do *Sam Dog* dá suporte à utilização de variáveis e constantes na construção de uma proposição. Para a construção das sentenças são adotados operadores relacionais (=, !=, <, <=, > e >=) e operadores lógicos (*and/or*).

O documento a seguir ilustra o estabelecimento de uma regra que permite que aplicações executadas em um ambiente de execução gerenciado pelo *Sam Dog* aceite conexões somente no período da madrugada na porta padrão para HTTP requisitadas pela entidade *m001*.

```
<sam>
  <rule>
    <entity>
      usp.icmc.lasdpc.grid.m001
    </entity>
    <task>
      tarefas.io.network.aceitarconexoes
    </task>
    <if>
      <expression>
      @porta=8080 and @hora>=0 and @hora<7
      </expression>
      <then>
        <command>accept()</command>
      </then>
      <else>
        <command>reject()</command>
      </else>
    </if>
  </rule>
</sam>
```

3. WSBCL

O WSBCL (*Web Services Based Class Loader*) é um carregador de classes para a plataforma Java totalmente baseado em serviços *Web*. Seu objetivo principal é permitir que classes Java possam ser carregadas independentemente do local onde se encontrem.

A arquitetura do WSBCL foi concebida de uma forma que o carregador de classes possa ser integrado com qualquer aplicativo Java que necessite de um mecanismo capaz de carregar classes localizadas em ambientes remotos. Dessa forma, observando a arquitetura do ponto de vista do *middleware* que necessita carregar as classes, ela não passa de uma aplicação cliente/servidora, onde o *classloader* atua como um cliente de um servidor que atende suas requisições.

Conforme apresentado pela figura 3, tanto o *middleware* quanto o carregador de classes são executados sobre a máquina virtual Java. Uma vez que o WSBCL está definido como carregador de classes do *middleware*, quando este necessita de uma classe, automaticamente o WSBCL é invocado, se conecta ao servidor solicitando os *bytecodes* e aguarda o retorno.

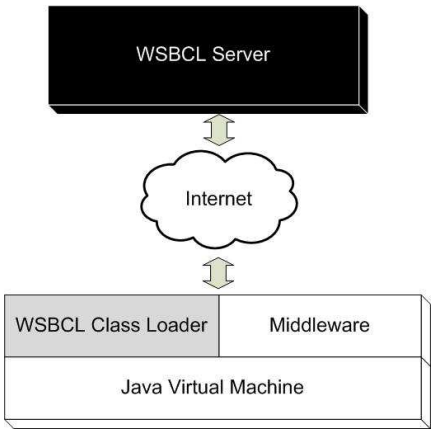

Fig. 3: Arquitetura do WSBCL do ponto de vista do Middleware que faz uso desse Classloader

Servidor Ativo

Para possibilitar que equipamentos de diferentes naturezas possam fazer uso de sistemas distribuídos para aumentar sua potência computacional (como é o caso de telefones celulares, receptores de TV, etc), é preciso que esses dispositivos sejam capazes de executar partes de suas aplicações remotamente.

Um *middleware* localizado em um servidor remoto tem a função de receber essa aplicação, executá-la e gerenciá-la. No momento da execução, novas classes são requisitadas, sob demanda, pela máquina virtual Java, o que faz necessário que o *classloader* utilizado pelo *middleware* se conecte ao equipamento que possui tais classes para requisitá-las. Dessa forma, os papéis de cliente e servidor são invertidos no momento da carga da classe, pois o *middleware* passa a ser cliente do equipamento com baixa potência computacional.

Uma vez que o WSBCL tem a comunicação remota realizada totalmente por meio de serviços *Web*, instalar os componentes de um servidor de aplicações (servidor HTTP, motor de processamento SOAP, etc) em um dispositivo de baixa potência computacional é algo inviável. Em função dessas características, o WSBCL faz uso de um servidor ativo que é executado no equipamento que tem o papel de hospedar e fornecer as classes Java.

Com a utilização de um servidor ativo, o carregador de classes não se conecta diretamente ao equipamento que hospeda as classes para realizar a requisição. Há um terceiro componente denominado *Class Relay* que atua como um

intermediário entre o hospedeiro e o carregador de classes, atuando como servidor de ambos. A figura 4 apresenta a arquitetura real do WSBL, onde a "caixa preta" demonstrada anteriormente é detalhada utilizando os componentes reais do modelo.

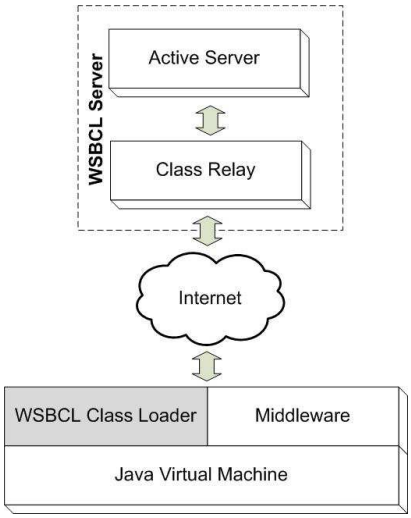

Fig. 4: Arquitetura do WSBCL demonstrando os componentes reais

O servidor ativo e o carregador de classes estabelecem por meio do *Class Relay* uma sessão que é utilizada para transmitir as requisições de classes e os *bytecodes*. Uma vez que o *Class Relay* é o único elemento servidor da arquitetura, é implementado em ambiente produtor/consumidor que utiliza esse componente como repositório de mensagens. Quando uma classe é requisitada o *classloader* produz junto ao *relay* uma solicitação. O servidor ativo mantém um processo de *pulling* que é responsável por consumir as solicitações.

Quando uma nova solicitação é encontrada pelo servidor ativo ele busca os *bytecodes* no dispositivo cliente e se conecta novamente ao *relay* produzindo a classe que será consumida pelo *classloader* também por meio de um processo de *pulling*.

Conforme demonstrado, as pedidos de conexão sempre ocorrem considerando o *Class Relay* como um servidor. Cada um dos componentes (*carregador de classes*, *Class Relay* e servidor ativo) pode ser executado em um computador diferente e a comunicação entre eles é realizada por meio de serviços *Web*.

Essa característica viabiliza que diferentes arquiteturas possam ser estabelecidas de maneira a atender diferentes cenários, conforme apresentado na figura 5.

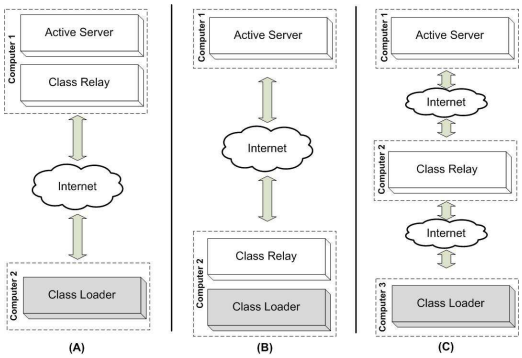

Fig. 5: Possíveis cenários de uso do WSBCL

A figura 5/A representa um cenário onde as classes estão hospedadas em um servidor que pode receber conexões externas. Dessa forma, o *Class Relay* e o Servidor Ativo ficam localizados na mesma máquina e se comunicam utilizando a interface de rede *Loopback* (127.0.0.1).

Quando um ambiente de computação em nuvem é utilizado, normalmente o servidor utilizado para executar as aplicações dos clientes pode receber conexões externas. Para isso a configuração apresentada na figura 5/B é ideal, pois ela permite que tanto o carregador de classes quanto o *Class Relay* fiquem juntos utilizando uma comunicação local.

Em situações onde o carregador e o servidor de classes não podem receber conexões externas é preciso empregar a arquitetura demonstrada na figura 5/C.

É importante notar que nos cenários demonstrados pelas figuras 5/B e 5/C, somente o servidor ativo é executado no dispositivo consumidor de recursos. Por se tratar de uma aplicação relativamente pequena ela pode ser executada em equipamentos com baixa potência computacional. Isso permite que dispositivos como telefones celulares e receptores de TV atuem no fornecimento de classes para aplicações executadas remotamente.

4. Sesiom

O Sesiom é um *container* para a plataforma Java que permite que objetos sejam migrados ou criados remotamente. Uma vez que é feito uso do protocolo SOAP na invocação dos métodos a portabilidade é garantida, o que permite, por exemplo, que clientes desenvolvidos em outras linguagens possam fazer uso dos objetos Java remotos.

No provedor, a arquitetura dos módulos é composta conforme a figura 6.

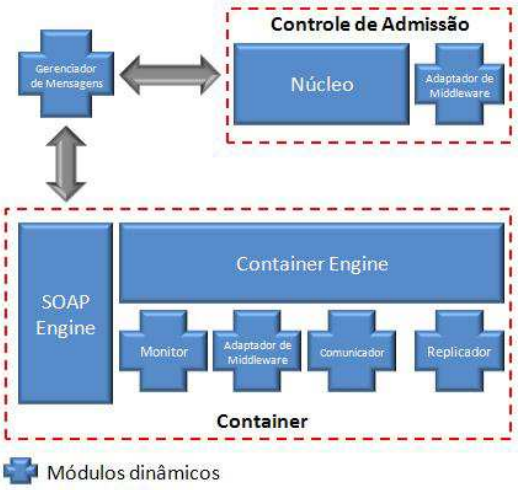

Fig. 6: Diagrama de Blocos do Sesiom

O controle de admissão é responsável por autorizar ou negar a hospedagem ou criação de um objeto no *container*. Uma vez que normalmente essa ação pode ser determinada por outro *middleware*, o Sesiom possui um adaptador que permite que ele se comunique com o módulo responsável por essa tarefa.

O SOAP *Engine* é responsável por receber uma mensagem SOAP, interpretá-la e realizar a chamada do método especificado. A forma de obtenção dessa mensagem é também um módulo dinâmico, o que permite que diversas formas diferentes de transporte dessa mensagem possam ser facilmente utilizadas.

Para cada sessão estabelecida pelo cliente uma nova instância do container é criada para atender suas requisições. O monitor é responsável por realizar o gerenciamento dos recursos consumidos pelo container, bem como contabilizar e cobrar pelo seu uso, se necessário.

O módulo comunicador permite que objetos específicos hospedados (*SesiomLets*) possam trocar mensagens entre si, mesmo estando remotos. O replicador é responsável pela confiabilidade do sistema, mantendo réplicas dos objetos que serão utilizadas em caso de falhas do container.

Referência dos Objetos Remotos

Para que um cliente possa interagir com um *container* instanciado no servidor é criado um *gateway* local que abstrai as operações possíveis. Por meio desse *gateway* é possível solicitar ações como criação e migração de objetos.

Quando um objeto é hospedado no container é preciso que o cliente possua formas para referenciá-lo nas ações desejadas. Conforme é ilustrado na figura 7, o container mantém uma tabela chamada HOT (*Hosted Objects Table*) que é responsável por realizar o relacionamento entre a referência local do objeto alocado na memória *Heap* e uma referência remota gerenciada pelo Sesiom. Além desse relacionamento, a tabela HOT armazena o momento do último acesso ao objeto.

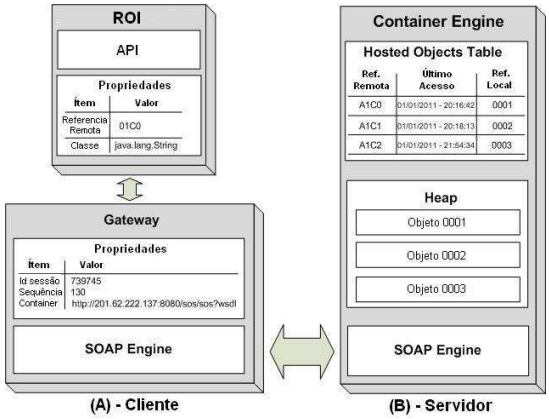

Fig. 7.: Referenciamento e Hospedagem de Objetos Remotos

Para que o cliente possa interagir com o objeto remoto é criado localmente uma interface chamada ROI (*Remote Object Interface*) que disponibiliza meios para realizar a invocação de métodos remotos, migração do objeto para outro *container*, retorno ou remoção.

Utilização do Cabeçalho do Protocolo SOAP

Quando serviços *Web* são utilizados para construir sistemas distribuídos, o próprio protocolo HTTP é suficiente para determinar o serviço que terá seu método invocado. No Sesiom isso não acontece, pois em um mesmo container pode haver muitos objetos hospedados, o que requer uma forma alternativa para realizar o endereçamento.

Para isso, os SOAP *Engines* do Sesiom fazem uso do cabeçalho do protocolo SOAP para realizar a identificação do objeto a ser invocado. No elemento *HEADER* desse protocolo foi inserido um novo elemento chamado *oid* que possui um atributo de nome *id* responsável por armazenar a referência remota do objeto.

Dessa forma, quando o SOAP *Engine* do *container* realiza o processamento de uma chamada de métodos ele recupera esse identificador da mensagem, realiza a busca na tabela HOT, identifica a referência local e faz a invocação do método.

Se o método invocado possui retorno, o Sesiom permite duas formas de envio de resposta. A primeira consiste em serializar o objeto retornado e colocá-lo na mensagem SOAP de

resposta. No entanto, se esse retorno for um objeto mais complexo que exija uma potência computacional maior que a disponibilizada pelo cliente, torna-se interessante que esse objeto de retorno seja também hospedado no *container* e uma nova referência remota retornada ao cliente.

O Sesiom permite que ambas as formas de retorno sejam utilizadas. Para isso, no cabeçalho da mensagem SOAP foi inserido um novo elemento chamado *returnType*. Esse elemento possui um atributo chamado *remoteReferente* que quando tem seu valor definido como true mantém o retorno hospedado no servidor, caso contrário retorna o objeto propriamente dito.

5. Grid Anywhere

O Grid Anywhere é um *middleware* para grades computacionais que possui, além de um núcleo comum, módulos acopláveis que permitem que diferentes arquiteturas de grades computacionais sejam compostas.

Atualmente, o Grid Anywhere atende soluções para a plataforma Java. Conforme ilustrado na figura 8, uma aplicação baseada no *Grid Anywhere* faz uso de uma máquina virtual local e uma ou mais máquinas virtuais remotas. As camadas ilustradas compreendem:

- *Grid Anywhere: Middleware* e API que permitem que a aplicação localizem recursos remotos e negocie por sua utilização. Além disso, são tratadas questões de monitoramento e replicação de objetos.
- *Sesiom*: *Container* para hospedagem de objetos e execução remota de métodos utilizando transporte flexível de mensagens do protocolo SOAP.
- *WSBCL*: Permite a carga de classes Java sob demanda em ambientes com conectividade e potência computacional limitados.
- *SAM Dog*: Ambiente seguro de execução que permite a herança de regras de utilização.

É importante observar que no momento do desenvolvimento de uma aplicação para a grade o programador faz uso da API do Grid Anywhere apenas para realizar a localização e alocação do provedor a ser utilizado. Nesse momento o recurso é abstraído por um gateway do Sesiom que será o responsável pela interação e gerenciamento de todo o ciclo de vida dos objetos remotos. Isso justifica o contato direto da aplicação com a API Grid Anywhere e com o Sesiom.

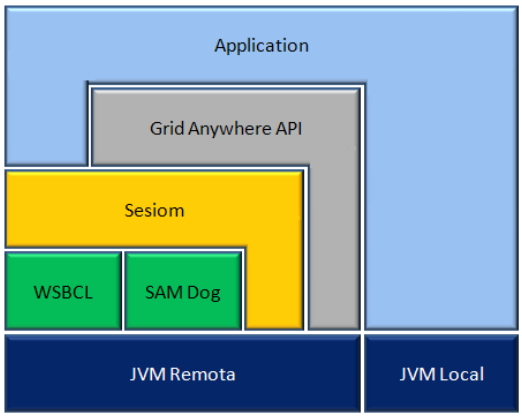

Fig. 8: Organização em camadas do *Grid Anywhere*

O Sesiom possui duas formas de execução. A primeira consiste na execução de métodos dos objetos hospedados. Nesse caso toda execução ocorre sobre o *Sam Dog* e WSBCL, pois é preciso, respectivamente, garantir a segurança da execução e viabilizar o carregamento de classes remotas. Mas, ocorre também a execução dos *bytecodes* que implementam o próprio Sesiom, o que é feito diretamente na máquina virtual Java.

Para que novas abordagens de grades computacionais *desktop* possam ser atendidas pelo *middleware*, os requisitos que são mais dependentes da arquitetura são implementados por módulos que são acoplados ao núcleo, o qual é responsável por realizar a articulação desses módulos dinâmicos que são (figura 9):

- *Gerenciador de arquivos*: Permite a transmissão de arquivos entre o provedor e o consumidor de recursos.
- *Escalonador*: Localiza e negocia a utilização de recursos remotos.
- *Adaptador de Container*: Permite que o núcleo do *Grid Anywhere* se comunique com o *container* (Sesiom) de aplicações para, por exemplo, notificar as execuções aceitas pelo escalonador.
- *Listeners*: São responsáveis por responder a eventos disparados pelos módulos. No provedor são implementados pelo desenvolvedor do *middleware* enquanto no consumidor eles são criados pelo programador da aplicação.
- *Sesiom*: *Container* para hospedagem e execução de métodos remotos.

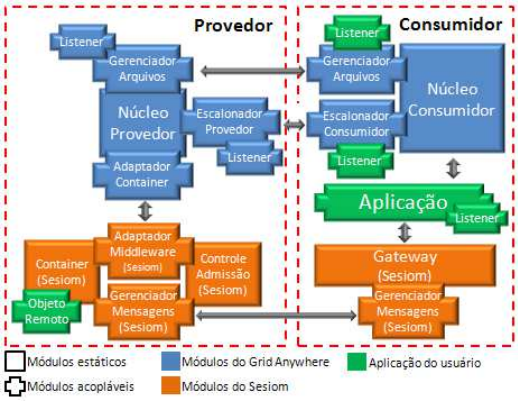

Fig. 9: Diagrama de Blocos do *Grid Anywhere*

Os módulos acopláveis foram definidos para atender aos requisitos que possuem maior variação entre arquiteturas diferentes. Esses módulos são definidos por interfaces Java que são responsáveis por especificar o comportamento mínimo de cada um. Cada módulo tem sua implementação descrita em um arquivo XML e os carregamentos que são realizados pelo núcleo ocorrem por meio da utilização de injeção de dependência.

6. Aplicações

Grade Computacional como Infra Estrutura de um Ambiente de PaaS por meio da utilização de SOAP Over SOAP

Uma variação do conceito de plataforma como serviço (PaaS) é uma alternativa para realizar a execução de aplicações de usuários domésticos em um servidor. É possível utilizar essa abordagem para permitir que equipamentos com recursos limitados sejam capazes de executar aplicações mais complexas por meio da distribuição de seus objetos que exigem maior potência computacional.

Para ser capaz de fornecer um serviço com menor custo faz-se necessária a utilização de uma grade computacional *desktop* como fonte de recursos computacionais para o fornecimento de plataforma como serviço. Para isso, é feito uso da arquitetura do *Grid Anywhere* considerando a existência de módulos para realizar a descoberta e escalonamento de recursos.

Conforme descrito anteriormente, para permitir que diferentes cenários de utilização possam ser viabilizados, o Sesiom admite que novos mecanismos de transporte de mensagens SOAP possam ser implementados por um Gerenciador de Mensagens e acoplados ao SOAP *Engine*. Essa característica é muito importante para o *Grid Anywhere*, pois é ela que torna possível a integração de diferentes dispositivos à grade computacional.

Conforme citado na seção sobre o WSBCL, é possível que os participantes da grade computacional *desktop* tenham restrições de conectividade que impeçam o recebimento de conexões de redes externas. Para isso, um servidor Sesiom não pode atuar no papel passivo do protocolo HTTP para receber as mensagens SOAP.

Esse problema é resolvido utilizando-se os mesmos conceitos de servidor ativo e *relay* empregados no WSBCL. Conforme ilustrado na figura 10, um *relay* de requisições é o único elemento passivo na arquitetura e opera entre o consumidor e o provedor de recursos.

Dessa forma, quando um cliente deseja invocar um método ele se conecta ao *relay* e envia a mensagem que representa a requisição. O *container*, por sua vez, também se conecta ao *relay* para solicitar mensagens destinadas a ele, caracterizando um processo de *pulling*.

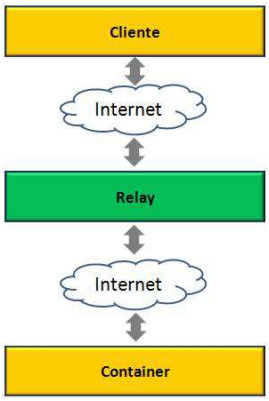

Fig. 10 : Relay de requisições

A requisição de invocação do método do objeto remoto é descrita por uma mensagem SOAP (#2) que é gerada pelo *gateway* existente no cliente, conforme figura 11. Uma vez que para transportar essa mensagem até o *relay* é feito uso de um serviço *Web*, ela é encapsulada dentro de outra mensagem SOAP (#1) gerada pela *stub* do cliente e encaminhada via requisição HTTP ao *relay*.

Como o servidor é ativo, ele realiza uma invocação ao serviço *Web* do *relay* solicitando requisições destinadas a ele. Quando uma nova requisição é encontrada, a mensagem SOAP que a representa (#2) é encapsulada dentro da mensagem SOAP de retorno da chamada do serviço *Web* do *relay* (#1) cujo transporte é feito pela resposta HTTP.

Sendo assim, o *stub* do servidor ativo processa o retorno da chamada do serviço *Web* do *relay*, obtém a mensagem SOAP que representa a chamada do método e a entrega ao SOAP *Engine* do Sesiom para processamento.

A resposta da execução do método é retornada ao cliente utilizando o mesmo princípio da chamada: inserindo uma mensagem SOAP dentro de outra.

Uma análise da figura 11 é importante. É possível observar que o *skeleton* não existe nem no servidor ativo nem no cliente, pois ambos se comportam como elementos ativos na arquitetura, sendo o *relay* o único passivo, possuindo os *skeletons*.

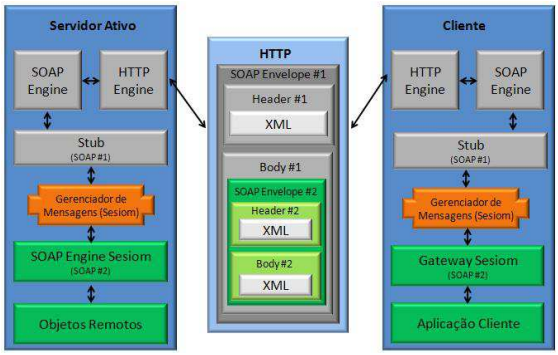

Fig. 11 : Ilustração da utilização de SOAP Over SOAP

Utilização de SOAP Over DTV para Viabilizar Set-top Boxes como Provedores de Recursos

As grades computacionais *desktop* vem há anos fazendo uso de computadores pessoais para realizar processamento paralelo de aplicações distribuídas por uma determinada instituição. Aplicações são enviadas aos colaboradores que as executam no formato de *Bag-of-Tasks* e retornam os resultados.

O *Grid Anywhere* permite a construção de grades computacionais *desktop* baseadas na migração de objetos para os voluntários e execução paralela de métodos por meio do protocolo SOAP. Dessa forma, um único objeto migrado para os voluntários pode ter seus métodos invocados quantas vezes forem necessárias.

No entanto, na grade aqui apresentada os voluntários são receptores digitais de televisão interativa que recebem dados de uma emissora via *broadcasting* e retornam informações por meio de uma conexão de Internet.

Nessa arquitetura, um consumidor de recursos que deseja realizar o processamento paralelo de uma aplicação não tem contato direto com os voluntários, mas sim com a emissora de TV. Sendo assim, o cliente envia um objeto (ou pede sua criação) à emissora que faz o encaminhamento, via difusão, para todos os receptores sintonizados a ela.

O *Grid Anywhere* permite a execução das aplicações de duas formas: a primeira utiliza o formato de *batch* (*Bag-of-Tasks*), quando o objeto é um *SesiomLet*, sendo a implementação iniciada pelo próprio *container* no momento de sua hospedagem. A segunda opção faz uso do protocolo SOAP para realizar a chamada remota dos métodos.

Conforme pode ser analisado na figura 12, quando o consumidor deseja fazer uso dos receptores de TV para executar uma aplicação paralela, primeiramente ele realiza, por meio das APIs do *Grid Anywhere*, o escalonamento de recursos. Esse escalonamento de recursos ocorre observando qual emissora possui uma melhor possível potência computacional naquele momento.

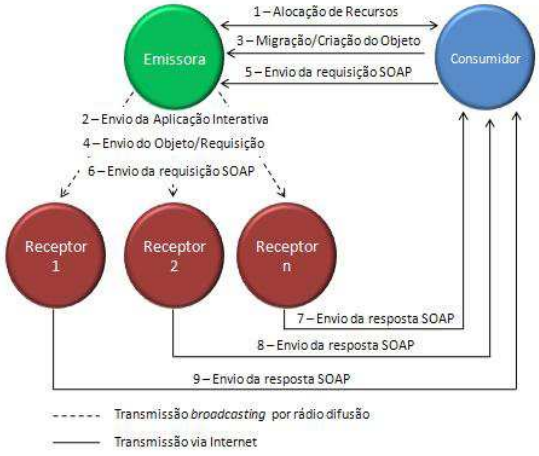

Fig. 12: Utilização de SOAP em um ambiente de difusão

Quando uma emissora é selecionada para ser utilizada, ela envia uma aplicação interativa (XLet) via difusão que é apresentada aos usuários que possuem o *Grid Anywhere* instalado no receptor. Os telespectadores, por meio dessa aplicação, autorizam ou rejeitam (utilizando o controle remoto) a execução da aplicação da grade.

Quando o consumidor termina de realizar a alocação dos recursos ele envia o objeto a ser migrado ou a requisição de criação para a emissora que faz o encaminhamento para os receptores que realizam a hospedagem desse objeto.

Para realizar a chamada de um método, que ocorre de forma paralela, o consumidor envia uma mensagem SOAP para a emissora que faz a transmissão para os receptores que recebem essa mensagem, executam o método indicado e

retornam a mensagem SOAP de resposta por meio da Internet (canal de retorno).

Para que esse modelo de execução paralela possa ser implementado é feito uso de uma configuração de módulos do *Grid Anywhere* que é demonstrada na figura 13. Nessa arquitetura há dois níveis de provedores de recursos. No primeiro nível encontra-se a emissora de TV e no segundo os receptores digitais (*set-top boxes*).

Os módulos acopláveis do *Grid Anywhere* foram desenvolvidos e organizados de forma a atender cada um dos níveis de provedor. Muitos componentes se encontram em ambos, mas com requisitos diferentes:

- *Escalonador*: Na emissora é responsável por negociar com o consumidor pelo uso da grade *desktop*. Esse módulo no receptor é responsável por determinar se a aplicação deverá ser executada localmente de acordo com as preferências do telespectador que podem incluir, por exemplo, o limite máximo de utilização dos recursos computacionais.
- *Gerenciador de arquivos*: Na transmissão entre emissora e consumidor a comunicação de dados é bidirecional e é realizada utilizando a Internet. O envio de arquivos da emissora para os receptores é realizado utilizando o multiplexador que fará a transmissão utilizando o carrossel de dados. Sendo assim, no *set-top box* o gerenciador de arquivos apenas recebe dados da emissora, mas pode enviar e receber diretamente do consumidor por meio da Internet.
- *Gerenciador de Mensagens:* Quando um objeto é invocado pelo consumidor a mensagem SOAP é propagada até a emissora por meio da Internet e sua recepção é feita pelo gerenciador de mensagens. Esse componente ao receber o documento XML realiza o seu encaminhamento via carrossel de dados e a recepção é feita no *set-top box* por um outro gerenciador de mensagens que realiza a leitura dos dados através do XLET JavaDTV. No momento do envio da mensagem SOAP de resposta, o documento é enviado via Internet diretamente para o consumidor

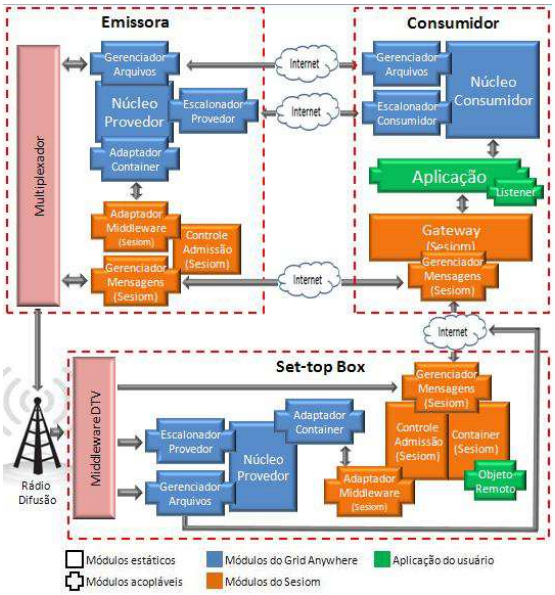

Fig. 13: Arquitetura de grade computacional para utilização de receptores de TV Digital na grade computacional

Quando uma emissora admite a execução de uma aplicação para a grade ela necessita enviar uma aplicação para todos os receptores sintonizados a ela. Essa aplicação tem como objetivo permitir a interatividade do telespectador para que ele opte por aceitar ou não a execução das tarefas da grade computacional e também alimentar o *middleware* do Grid Anywhere instalado no set-top box com mensagens obtidas do carrossel de dados.

O aplicativo transmitido consiste em um XLet do *Grid Anywhere* que é assinado pela emissora de TV. As mensagens são documentos XML que podem conter informações de controle (escalonamento da grade, estabelecimento de sessões do Sesiom, etc) e de chamadas remotas de métodos (SOAP).

A transmissão das mensagens é feita utilizando o *Broadcasting File System* especificado pelo JavaDTV. Na emissora, o gerenciador de mensagens do Sesiom realiza a inserção de um arquivo chamado gaw.xml no carrossel de dados (passos 1 e 2 da figura 14) que será posteriormente consumido pelo XLet.

Para cada nova mensagem a ser enviada o arquivo gaw.xml é atualizado pela emissora, o que é verificado pelo *middleware* DTV (3) que realiza o disparo (4) de um evento (*com.sun.dtv.broadcast.BroadcastFileEvent*), o qual é recebido (5) e processado pelo XLet (*com.sun.dtv.broadcast.BroadcastFileListener*).

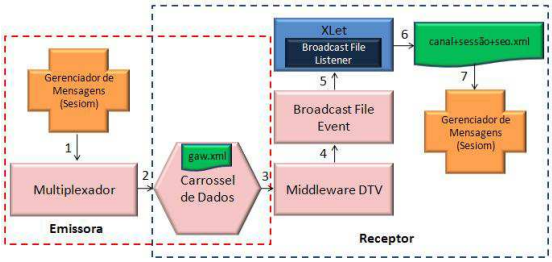

Fig. 14: Transmissão das mensagens SOAP pelo fluxo de dados do sistema de TV Digital

O XLet tem a função de fazer a leitura dos novos dados e gerar (6) um documento no sistema de arquivos local utilizando o diretório especificado para a aplicação. Esse arquivo tem o nome composto pelo número do canal da emissora, número da sessão estabelecida pelo Sesiom e número de sequência das mensagens (dados que são obtidos do documento gaw.xml).

Um módulo gerenciador de mensagens realiza a busca desses arquivos e os entrega ao *middleware* para grade computacional (7).

Uma vez que essa arquitetura é baseada na especificação Java DTV, o *Grid Anywhere* pode ser acoplado a qualquer *middleware* de televisão digital interativa que atenda às normas. Nesse trabalho foi feito uso do middleware Ginga [8].

Utilizando uma máquina virtual disponibilizada pelos desenvolvedores do Ginga, foram criados os receptores. Esses receptores receberam as implementações dos componentes descritos na figura 13. Para poder simular nessas máquinas virtuais o comportamento do carrossel de dados e do evento *BroadcastFileEvent* foi feito uso do NFS (*Network File System*), que por meio de um sistema de arquivos distribuído simulou a chegada e atualização dos arquivos que são transmitidos pela emissora.

Fig. 15: Imagem do Ginga apresentando um vídeo e solicitando permissão do usuário para executar a aplicação da grade

Na figura 15 é apresentada a imagem da interface do Ginga realizando a transmissão de um vídeo. No momento do inicio da alocação de recursos pelo *Grid Anywhere* é apresentada a solicitação de permissão de execução de aplicações de grade. Os dados que são apresentados ao usuário são obtidos do arquivo XML transmitido juntamente com a aplicação.

7. Conclusões

Durante os estudos realizados foi possível identificar os problemas que são comuns nas possíveis diferentes arquiteturas para grades computacionais *desktop*, como também aqueles que necessitam de tratamento individual para cada circunstância. Por isso, este trabalho apresentou modelos e implementações concretas para as questões que são mantidas entre as diversas possíveis configurações de uma grade computacional *desktop* e especificações que permitem que os requisitos variáveis possam ser resolvidos e acoplados, de maneira simples, àqueles já prontos.

Assim, as principais contribuições dessa tese podem ser destacadas como:

- *Especificação da arquitetura:* Foram geradas interfaces que permitem que módulos com maior dinamismo sejam implementados e acoplados, por meio de injeção de dependência, aos núcleos do *middleware*.
- *Container de objetos com transporte flexível de mensagens SOAP*: Uma vez que o *middleware* proposto tem seu foco no compartilhamento de recursos de processamento, foi criado o Sesiom, um ambiente de hospedagem de objetos e invocação remota de métodos totalmente baseado em SOAP. Para isso, novos elementos foram inseridos no *HEADER* do protocolo para dar suporte ao modelo de execução proposto. As mensagens desse protocolo podem ter sua forma de transporte implementada em um módulo distinto que é acoplado ao container.
- *Transporte de SOAP por meio de outra mensagem SOAP*: Para permitir que no ambiente da grade computacional todos os participantes atuem como agentes ativos e consigam fazer parte do sistema mesmo estando em situações com conectividade restrita (atrás de *firewalls* e NATs, por exemplo) foram utilizados serviços *Web* para implementar um ambiente de transporte de mensagens SOAP, baseado na existência de um intermediário passivo.
- *Transporte de objetos e mensagens SOAP por meio de um sistema de televisão digital*

interativa: A integração do *Grid Anywhere* com o *middleware* JavaDTV permite que o ambiente de comunicação das emissoras de TV seja utilizado como meio de transporte de objetos Java, que são transmitidos via difusão e posteriormente invocados através de mensagens SOAP, também transmitidas via *broadcasting*.

- *Carregamento de classes por meio de serviços Web*: As soluções para carregamento de classes Java disponibilizadas pela literatura não foram adequadas para situações onde há problemas de conectividade e de baixa potência computacional dos participantes de sistema. Por isso, um mecanismo totalmente baseado em serviços *Web* (WSBCL) foi desenvolvido durante este trabalho para permitir que o processo de carga também possa ser realizado de forma que tanto o cliente quanto o servidor atuem como agentes ativos na arquitetura.
- *Ambiente seguro de execução com configuração escalável*: Devido ao aumento do número de consumidores proporcionado pelas grades computacionais *desktop* providas pelo *Grid Anywhere*, foram apresentadas a proposta e a implementação de um ambiente seguro de execução de aplicações Java (*Sam Dog*), que permite a herança e sobreposição de regras que permitem, de forma flexível, definir o que pode ser feito, por quem e em quais contextos.
- *Arquitetura de grade computacional como infraestrutura de um ambiente de PaaS*: Configuração dos componentes implementados para permitir que recursos compartilhados possam ser ofertados como serviço por usuários convencionais.
- *Arquitetura de grade computacional desktop que faz uso de receptores de TV como provedores de recursos*: O *Grid Anywhere* foi utilizado para permitir que os receptores (*set-top boxes*) de sistemas de televisão digital interativa baseados em JavaDTV possam atuar como provedores de recursos.

Com esses resultados é possível concluir que a tese atingiu plenamente seus objetivos. As pesquisas realizadas culminaram em modelos que foram validados por meio das implementações, as quais deverão ser disponibilizadas pelo Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) para a implementação de aplicações reais.

Como atividades futuras destacam-se duas principais frentes de trabalho:

- Construção de módulos que permitam que a grade computacional *desktop* para oferta de PaaS possa ser acomodada em um ambiente P2P. Para isso é preciso que módulos para descoberta de recursos e escalonamento sejam implementados utilizando esse paradigma de computação distribuída.
- Extensão do Sesiom para permitir que dispositivos móveis dotados do sistema Android possam fazer uso de recursos localizados na nuvem para ampliar a potência computacional do equipamento.

Referências

[1] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," Physics Today, vol. 55, pp. 42-47, 2002.

[2] I. Foster, C. Kesselman, e S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputer Applications, vol. 15, pp. 200-222, 2001.

[3] F.C. Teixeira e M.B.F. Toledo ; “Arquitetura de Grade Computacional Baseada em Modelos Econômicos”. 5th International Information and Telecomunication Technologies Symposium, Cuiabá, 2006.

[4] D.P. Anderson, "BOINC: A system for public-resource computing and storage," in Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, pp. 4-10, 2004.

[5] N. Constantinescu-Fülöp, "A Desktop Grid Computing Approach for Scientific Computing and Visualization," Norwegian University of Science and Technology, Tese de doutorado 2008.

[6] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in Grid Computing Environments Workshop, pp. 1-10,2008.

[7] J. Fernandes, G. Lemos, and G. Silveira, "Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas," in Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, 2004.

[8] G. Lemos, L.E.C. Leite, C.E.C.F. Batista, and T.P. Falcão, "Ginga-J: The Procedural Middleware for the Brazilian Digital TV System," Journal of the Brazilian Computer Society, vol. 13, 2007.