Construindo Infraestruturas Computacionais Distribuídas e Seguras Baseadas em Receptores de TV Digital

Rostand Costa[1, 2], Giuliano Maia[1], Guido Lemos de Souza Filho[1], Dênio Mariz Sousa[1], Francisco Vilar Brasileiro[2]

[1] Laboratório de Aplicações em Vídeo Digital – LAVID
Universidade Federal da Paraíba (UFPB) - João Pessoa, PB – Brasil

[2] Laboratório de Sistemas Distribuídos – LSD
Universidade Federal de Campina Grande (UFCG) - Campina Grande, PB - Brasil

{rostand, giuliano}@lavid.ufpb.br, {guido, denio}@lavid.ufpb.br, fubica@dsc.ufcg.edu.br

Abstract. *Building a large scale, distributed computational infrastructure based on Digital Television (DTV) receivers enables a potential for expressive processing power for the execution of Many Task Computing (MTC) applications. Since the DTV receivers are non dedicated public resources and also inherently volatile and not reliable there are several challenges involved in the operation of such systems and the establishment of a secure, robust and efficient communication mechanism. This work presents a security model for providing processing services atop of a DTV system infrastructure, identifying the main threats to the communication between its components and discussing the appropriate countermeasures.*

Resumo. *A construção de uma infraestrutura computacional distribuída de larga escala baseada em receptores de TV Digital gera um potencial de processamento expressivo para a execução de aplicações MTC – Many Task Computing. Entretanto como se tratam de recursos públicos não dedicados e inerentemente voláteis e não confiáveis há diversos desafios envolvidos na operação de tais sistemas e no estabelecimento de um mecanismo de comunicação seguro, robusto e eficaz. Este trabalho apresenta um modelo de segurança para a oferta de serviços de computação sobre uma infraestrutura distribuída baseada em receptores de TV Digital, identificando as principais ameaças à segurança da comunicação entre seus componentes e apresentando as correspondentes contramedidas aplicáveis.*

1. Introdução

A crescente popularidade da Internet a fez extrapolar ambientes acadêmicos, científicos e empresariais e ocupar as residências e o cotidiano das pessoas de uma forma quase que onipresente. Este fenômeno tem trazido a reboque uma série de avanços que estão mudando a forma como computadores são usados hoje em dia. A disponibilidade de acesso a redes de alta velocidade combinada com a crescente oferta de computadores com alta capacidade de processamento, agora cada vez mais acessíveis às camadas da população de mais baixa renda, é um fenômeno em escala mundial.

Visando o aproveitamento do poder computacional que este enorme contingente de recursos distribuídos mundialmente representa, abrem-se novas oportunidades técnicas, principalmente com a possibilidade de uso desses recursos para resolver problemas de grande escala nas áreas da ciência, engenharia, medicina, física etc. Os principais atrativos desta idéia são a possibilidade de alocar uma enorme quantidade de recursos para o processamento distribuído de uma aplicação paralela (centenas de milhares de

computadores conectados via Internet, por exemplo) e fazê-lo a um custo muito menor do que alternativas tradicionais, baseadas em supercomputadores paralelos.

O cenário tecnológico atual é fortemente orientado para a convergência e marcado pelo surgimento de serviços e dispositivos que combinam tecnologias que surgiram inicialmente em contextos distintos. Desde celulares com capacidade de captura de imagens e vídeo ao provimento de serviços agregados de telefonia, internet e televisão, dos modems móveis para acesso à Internet aos celulares de terceira geração com grande memória e processadores poderosos, praticamente tudo que é digital é potencialmente convergente.

Um exemplo clássico de dispositivos com poder computacional relevante são os receptores de TV Digital (TVD) [MORRIS 2005], cuja presença nas residências é uma tendência com a digitalização da televisão, a mais popular das mídias de massa. A TV Digital oferece recursos que vão desde a melhoria da qualidade da imagem à capacidade de interação com o conteúdo. Com a TVD o telespectador tem a possibilidade de exercer um papel mais ativo, interagindo com os programas de televisão, que além de áudio e vídeo, passam também a incorporar software de forma sincronizada. Para tanto, o receptor de TV Digital conta com características típicas de um computador: possui memória, processador, sistema operacional e capacidade de se conectar em rede.

Esta miríade de dispositivos digitais recentes ou tradicionais, computacionalmente capazes, virtualmente conectados e eventualmente ociosos, se devidamente coordenados e agrupados, podem representar um potencial de processamento sem precedentes para a construção de infraestruturas computacionais distribuídas de larga escala.

Many Task Computing (MTC) [RAICU 2008] é um paradigma de computação que explora a idéia de usar muitos recursos computacionais, eventualmente distribuídos, para processar tarefas menores em paralelo, em oposição à idéia de processar uma única tarefa grande em um único recurso computacional de maior capacidade. Há uma categoria de aplicações paralelas cujas tarefas são totalmente independentes umas das outras, definidas na literatura como aplicações *Embarassingly Parallel Applications* ou *Bag-of-task Applications* (BoT) [CIRNE 2003]. A vazão obtida quando se executa aplicações MTC, em geral, e BoT, em particular, sobre uma infraestrutura computacional distribuída depende diretamente da escala que a mesma permite. O tamanho do *pool* de processamento, definido como o número de processadores alocados, é o principal promotor de desempenho, enquanto que o esforço de coordenação envolvido é o principal fator de limitação. Para atingir uma vazão extremamente alta é necessário operar eficientemente em escala extremamente alta, assumindo que a distribuição de tarefas para os processadores disponíveis e o fornecimento de qualquer dado de entrada necessário ou coleta dos resultados gerados não sejam um gargalo.

O uso eficiente da infra-estrutura por aplicações MTC com tarefas de curta duração (*short-lived*) requer a capacidade de instanciar um grande *pool* de recursos para uma aplicação a qualquer tempo e somente enquanto durar a execução da aplicação. Estes recursos podem ser depois realocados para aplicações diferentes. Além disso, para permitir a execução de um número ilimitado de aplicações de tipos distintos, é essencial que a configuração da infraestrutura, inclusive a instalação de qualquer componente de *software* específico da aplicação, possa ser realizada de forma leve em termos de complexidade e ágil em termos de tempo. Tal premissa deve continuar válida até mesmo considerando-se que a escala desejada esteja na ordem de centenas de milhões de nós de processamento. Em outras palavras, o usuário deve ser capaz de facilmente e rapidamente personalizar a infraestrutura de processamento inteira de acordo com as suas necessidades. Em resumo, para prover computação de vazão extremamente alta a um número grande de aplicações MTC, uma

infraestrutura de computação distribuída precisa oferecer: a) *escalabilidade extremamente alta*, controlando de centenas a milhões de nós de processamento indistintamente; b) *instanciação sob demanda*, possuindo mecanismos para descoberta, montagem e coordenação dos recursos dinamicamente; e c) *configuração eficiente* das aplicações, sem exigir nenhuma intervenção individual ou especializada.

Infelizmente, as tecnologias atuais possuem limitações fundamentais que têm impactos ou na sua escala ou no seu alcance. Os sistemas para *voluntary computing* [ANDERSON 2002][ANDERSON 2004] provaram que é possível construir plataformas computacionais com milhões de nós para suportar a execução de aplicações MTC. Estes sistemas, entretanto, não possuem a flexibilidade das infraestruturas de *desktop grids* [LITZOW 1988][CIRNE 2006][OLIVEIRA 2002][ANDRADE 2007][THAIN 2006], sendo uma solução válida somente para um subconjunto muito pequeno de aplicações que podem se beneficiar da vazão extremamente alta que eles podem entregar. A abordagem de computação voluntária (VC) tem sido bem sucedida apenas nos casos onde a aplicação possui um apelo que motive os usuários a participarem do projeto e doarem os seus projetos. Os casos de sucesso mais relevantes envolvem a busca pela cura de doenças e busca por vida extraterrestre. Por outro lado, as infraestruturas flexíveis atualmente disponíveis, como as baseadas no paradigma de *cloud computing* [WANG 2008], embora sejam, em tese, virtualmente inesgotáveis, estão limitadas tanto pela capacidade física dos provedores atuais quanto pelos modelos de negócios vigentes, que restringem a alocação de uma quantidade muito alta de nós de processamento [AMAZON 2010].

Em um trabalho anterior, os autores apresentaram uma proposta de uma arquitetura nova para computação distribuída que é ao mesmo tempo flexível e altamente escalável. Esta abordagem, chamada de *OddCI – On-Demand Distributed Computing Infrastructure* [COSTA 2009], é suportada pela existência de milhões de dispositivos que podem, simultaneamente e sob demanda, ser acessados através de uma rede com suporte à transmissão em *broadcast*. Aferindo a viabilidade da arquitetura proposta, etapas anteriores da pesquisa demonstraram como ela pode ser modelada e implementada sobre um sistema de televisão digital.

Como o estado real de uma rede de *broadcast*, em termos de quantidade e perfil dos recursos conectados, é desconhecido e os recursos potencialmente acessíveis são públicos, não dedicados e, inerentemente voláteis e não confiáveis, há diversos desafios envolvidos na operação segura de sistemas OddCI.

Neste trabalho, é feita uma analise dos aspectos de segurança relacionados ao funcionamento de sistemas OddCI que, por suas singularidades, são submetidos de forma concomitante as falhas, ameaças e vulnerabilidades presentes em outros contextos apenas de forma isolada. Desta forma, os sistemas OddCI precisam lidar com aspectos de autenticação de fontes de mensagens, como os controles de segurança *multicast* [CANETTI 1999], ao mesmo tempo que precisam tratar da volatilidade de entrada e saída de nós, como fazem os sistemas distribuídos dinâmicos [LIMA 2009] enquanto precisam, simultaneamente, oferecer mecanismos de tolerância à sabotagem, como os presentes em computação voluntária [SARMENTA 2001].

O restante do texto está organizado como segue. Na Seção 2, será apresentada e discutida a arquitetura de sistemas OddCI. Na Seção 3, será detalhado o fluxo de operação de um sistema OddCI para identificação dos requisitos e definição dos objetivos de segurança que devem ser atendidos para conciliar as expectativas de cada um dos atores envolvidos. Na Seção 4, um Modelo de Segurança para Sistemas OddCI é proposto através da apresentação de primitivas básicas de segurança e do detalhamento do protocolo de uso

das contramedidas selecionadas considerando o fluxo de operação descrito na Seção 2. A Seção 5 contém um estudo de caso que aponta a viabilidade de se implementar o modelo de segurança proposto em uma redes de TV Digital. A Seção 6 traz as considerações finais.

2. Arquitetura OddCI

Na arquitetura OddCI, uma rede de *broadcast* padrão é aumentada com três componentes, um *Provider*, um *Controller* e um *Backend*; além disso, é assumido que os dispositivos acessíveis pela rede de *broadcast* também podem se comunicar com o *Controller* e o *Backend* através de um canal de comunicação individual (ponto-a-ponto e *full-duplex*), chamado canal de interação (Fig. 1). O *Provider* interage com os clientes e recebe os pedidos por instâncias OddCI, repassando-as, quando exequíveis, para que o *Controller* possa providenciar a sua instanciação. O *Controller* é o encarregado de montar e manter a infraestrutura computacional sob demanda propriamente dita, enquanto que o *Backend* é responsável pelas atividades de administração específicas de cada aplicação executada. Estas atividades podem incluir escalonamento, provisão de dados de entrada, como também o recolhimento e, eventualmente, o pós-processamento dos resultados gerados pela aplicação distribuída.

Uma característica notável desta arquitetura é que, diferentemente de qualquer alternativa para computação distribuída, não são, necessariamente, requeridos identificação prévia ou procedimentos de registro para os recursos existentes. Simplesmente, a infra-estrutura não existe até que seja solicitada e ativada através do canal de *broadcast*. Por causa desta singularidade, a arquitetura é chamada de OddCI (*On-Demand Distributed Computing Infrastructure*) e o processo de ativação de uma instância OddCI através do envio de mensagens de controle (*wakeup messages*) através do canal de *broadcast* é chamado de *wakeup process*. Os dispositivos que executarão o *processing node agent* (PNA) são descobertos e inicializados através de uma *wakeup message* (WM) enviada via *broadcast*, que contém o executável do PNA. No caso de TV Digital, por exemplo, este processo é nativo e executado sempre que uma transmissão contém uma aplicação com o flag "*autostart*" setado. Alguns mecanismos do *Controller* regulam a quantidade de PNA efetivamente ativados, posto que todos os dispositivos conectados na rede de *broadcast* serão atingidos pela WM.

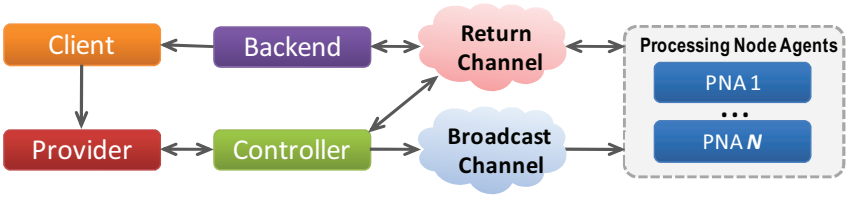

Figura 1. A arquitetura geral de um sistema OddCI

A camada de *software* que suporta a criação de sistemas OddCI está estruturada nos quatro componentes descritos abaixo:

- O *Client* (cliente) é responsável por definir a imagem (código executável a ser processado pelos PNA) e os dados de entrada que eventualmente serão necessários. O cliente então submete um pedido de criação de uma instância OddCI ao *Provider* e aguarda o recebimento dos resultados produzidos, os quais são repassados através do *Backend*. O *Client* é uma aplicação manuseada por um usuário que interage com um *Provider* para obtenção dos serviços de computação de tarefas.
- O *Provider* (provedor) é responsável por criar, manter e destruir as instâncias OddCI de acordo com as solicitações dos clientes. O *Provider* envia as instruções adequadas de tal

forma que cada instância OddCI solicitada possa ser dinamicamente provisionada ou liberada pelo *Controller*.

- O *Controller* (controlador) é encarregado de configurar a infra-estrutura distribuída, conforme instruído pelo *Provider*, através da formatação e envio, via o canal de *broadcast*, de mensagens de controle, incluindo imagens de software, necessárias para construir e manter as instâncias OddCI. Ele recebe informações dos nós de processamento ativos sobre o seu estado e configuração, consolida-as e as repassa para o *Provider*.
- *Backend (ret*aguarda) é responsável pela comunicação direta com o *software* específico da aplicação sendo *ex*ecutado em cada nó de processamento; mensagens são enviadas dos nós de processamento para o *Backend* para solicitar o envio de novas tarefas (dados e/ou programas) e para transferir de volta os resultados de cada tarefa. O *Backend* deve ser configurado e adequadamente provisionado para cada aplicação específica sendo executada.
- *Processing Node Agents* (PNA) (agentes processadores) são os componentes responsáveis pelo gerenciamento e carga de novas imagens de aplicações na memória do dispositivo e pela execução da imagem carregada. *O PNA* é carregado pelo dispositivo em resposta a uma *wakeup message* e se comunica regularmente com o *Controller* através do canal de interação para sinalizar o seu estado corrente.

O modelo básico de funcionamento de Sistemas OddCI (criação e operação*)* pode ser ilustrado através dos fluxos de troca de mensagens possíveis entre os diversos componentes, conforme ilustrado na Figura 2.

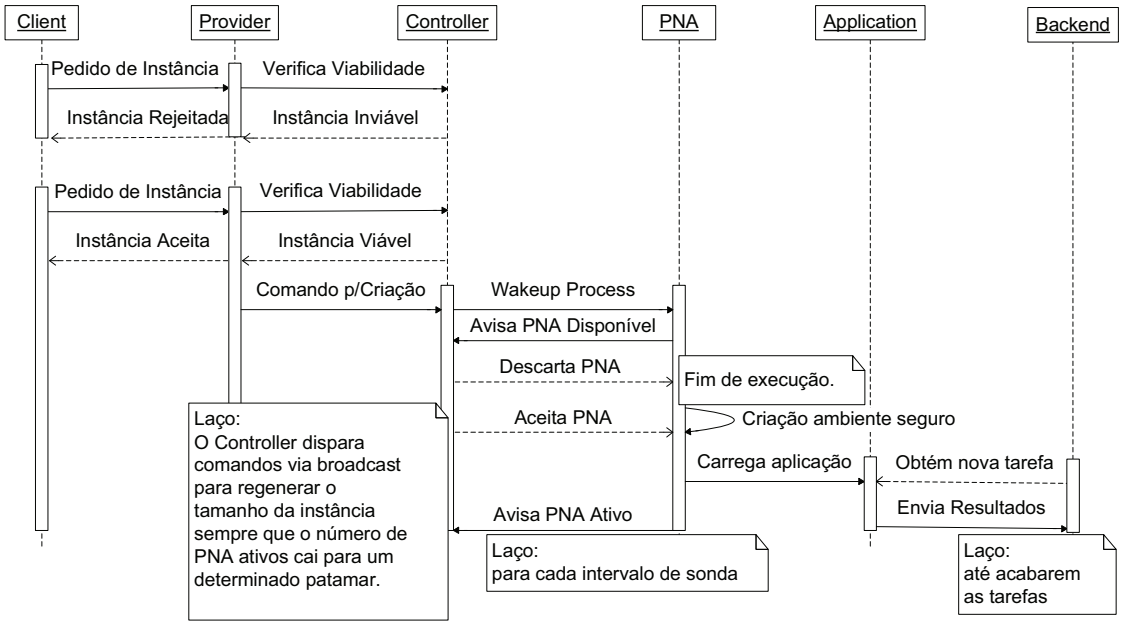

Figura 2. Sequência de operação básica de um sistema OddCI

Inicialmente, o *Client* solicit*a* ao *Provider* a criação de uma instância OddCI, fornecendo a imagem da aplicação a ser executada em cada processador. O *Provider* valida o cliente e a imagem da aplicação e, baseado no histórico e em estimativas dos recursos disponíveis no momento, acata (ou não) o pedido.

Em seguida, o *Provider* formata e encaminha uma mensagem de controle para que esta seja transmitida pelo *Controller*.

O *Controller*, após validar o *Provider* e a mensagem de controle, providencia sua transmissão, via *broadcast*, para todos os dispositivos conectados.

O dispositivo, ao receber a mensagem de controle contendo o código do PNA e a imagem da aplicação cliente, inicia o seu tratamento, carregando o PNA. A primeira providência do PNA ao ser carregado é usar o canal de interação para sinalizar ao *Controller* a sua disponibilidade e, caso seja aceito, cria um ambiente seguro, carrega a aplicação cliente e inicia a sua execução.

A aplicação cliente também usa o canal de interação para obter tarefas e enviar resultados para o *Backend* diretamente.

Enquanto a aplicação cliente está sendo executada, o PNA, em um determinado intervalo, envia mensagens para o *Controller* através do canal de interação para sinalizar que continua ativo. Consolidando as mensagens recebidas de todos os PNAs integrantes da instância, o *Controller* pode monitorar o seu tamanho e disparar novas mensagens de controle via *broadcast* para cooptar novos dispositivos sempre que necessário.

Opcionalmente, o *Backend* pode repassar ao *Client* informações sobre a vazão sendo fornecida pela instância OddCI. O *Client*, neste caso, pode solicitar ao *Provider* que aumente ou diminua o tamanho da instância, reiniciando o ciclo.

Em tal cenário, relevantes questões de segurança estão envolvidas: Como garantir o sigilo e a integridade dos dados e resultados dos *clientes* durante o seu tráfego no sistema OddCI? Como garantir a legitimidade de cada nova aplicação dinamicamente carregada em um PNA? Como estabelecer canais de comunicação seguros envolvendo partes voláteis e não identificadas previamente? Como conciliar as necessidades de segurança de cada um dos atores envolvidos?

3. Requisitos de Segurança

Do ponto de vista de segurança, cada ator de um Sistema OddCI possui as suas próprias expectativas e interesses. O *Client* espera que os dados das suas aplicações estejam protegidos durante todo o ciclo de vida da instância OddCI, tanto de adversários quanto dos outros atores. O *Provider* precisa autenticar os clientes e garantir que os recursos disponíveis sejam usados apenas por instâncias legítimas. O *Controller* precisa validar as mensagens de controle a serem transmitidas para evitar interrupção ou falhas no seu serviço. O *Backend* precisa autenticar os PNA que enviam resultados e os clientes que o acessam para recuperar tais resultados. Finalmente, os proprietários dos equipamentos que executarão o PNA e as aplicações estão interessados, principalmente, em que a execução das aplicações não interfira no funcionamento dos seus dispositivos. Além das preocupações específicas de cada ator, todos estão sujeitos às repercussões e impactos causados por tentativas de sabotagem ou falhas localizadas em componentes, intencionais ou não.

Os requisitos de segurança que precisam ser atendidos em nosso contexto podem ser consolidados a partir da observação da sua dinâmica de interações. A Figura 3 traz as interações básicas entre os participantes de um Sistema OddCI.

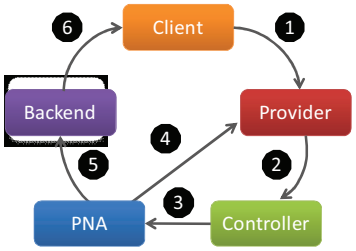

Figura 3. Interações Básicas entre Participantes de um Sistema OddCI

O fluxo (**1**) requer a autenticação do cliente pelo *Provider* e a confidencialidade na comunicação entre eles como forma de proteger a imagem (código a ser executado) e os dados enviados para o *Provider*. No fluxo (**2**), o *Controller* também precisa autenticar o *Provider* e garantir confidencialidade na troca de mensagens de controle. No fluxo (**3**), que ocorre entre o PNA e o *Controller*, o *PNA* precisa receber e enviar mensagens de forma confidencial, além de autenticar a origem da mensagem de controle recebida via *broadcast* visando garantir que elas são realmente oriundas do *Controller*. Entretanto, o canal de *broadcast* estabelece uma comunicação de "um-para-muitos" entre o *Controller* e os equipamentos conectados, o que requer mecanismos de autenticação e confidencialidade distintos dos usados nos fluxos (**1**) e (**2**).

Nos fluxos (**4**) e (**5**), o *PNA* e a aplicação cliente precisam de confidencialidade para estabelecer comunicações seguras com o *Controller* e o *Backend*, respectivamente. Isto pode ser obtido com facilidade se as partes envolvidas puderem ser devidamente identificadas. Como o receptor é um componente volátil e não conhecido previamente, a sua *autenticação* precisa ser tratada de forma especial, não sendo aplicável o uso das formas de autenticação mais tradicionais já descritas. O uso de chaves embutidas (usando os conceitos de *embedded keys* [BOESGAARD 2007] e *ofuscamento de programas* [D'ANNA 2003], por exemplo) dentro do código do PNA e da aplicação cliente é uma alternativa de associar uma identidade para estes processos que executam nas partes não controladas do sistema, tornando-as passíveis de serem autenticadas pelos processos de retaguarda equivalentes no *Controller* e no *Backend*. O uso das técnicas de chaves embutidas e de ofuscamento, além de eficiente, ganha uma vantagem adicional no contexto OddCI no qual as instâncias são formadas dinamicamente. Como o código do PNA e da aplicação fornecida pelo cliente são enviados em cada *wakeup message*, as chaves embutidas e a técnica de ofuscamento podem ser alteradas frequentemente para ficarem obsoletas com rapidez. Isto reduz o tempo de exposição de tais mecanismos e diminui a eficácia de ataques destinados a obter tais chaves e interferir na comunicação entre o *Controller* e o PNA e entre a aplicação e a sua retaguarda.

A confidencialidade da imagem da aplicação precisa ser garantida até a sua efetiva execução, sendo transversal para os fluxos (**1**), (**2**) e (**3**). Confidencialidade transversal, neste caso, significa que a mensagem seja enviada, sequencialmente, da parte 1 para a parte N, mas que só possa ser aberta pelo destino final (Princípio da Não Interferência Intransitiva [SCHELLHORN 2000]). Por exemplo, os dados da aplicação enviados pelo cliente e retransmitidos pelo *Provider* e pelo *Controller*, só devem ser abertos pela aplicação cliente quando a mesma for iniciada pelo PNA. Adicionalmente, o *Backend* precisa validar a *integridade* dos resultados recebidos para se proteger de falhas bizantinas [LIMA 2009] ou tentativas de sabotagem [SARMENTA 2001], o que pode exigir controles específicos que consideram a semântica e a sintaxe adotada em cada aplicação.

O fluxo (**6**) é opcional e envolve uma comunicação particular e controlada entre o cliente e a sua estrutura de retaguarda que não será abordada diretamente aqui. Entretanto, pelas suas características, o mesmo tratamento aplicado nos fluxos (**1**) e (**2**) também pode ser utilizado.

De forma consolidada, a Tabela 1 traz os objetivos de segurança extraídos dos requisitos levantados.

Tabela 1. Objetivos de Segurança

OS	OBJETIVO DE SEGURANÇA
O1	Autenticação mútua de partes previamente identificadas nos fluxos (**1**) e (**2**)
O2	Autenticação unilateral de partes previamente identificadas no fluxo (**3**)
O3	Autenticação unilateral de partes voláteis e não identificadas nos fluxos (**4**) e (**5**)
O4	Comunicação síncrona segura para os fluxos (**1**), (**2**), (**4**) e (**5**)
O5	Comunicação assíncrona segura para o fluxo (**3**)
O6	Comunicação transversal segura para os fluxos (**1**), (**2**) e (**3**)
O7	Controle semântico fim-a-fim no fluxo (**5**)
O8	Confidencialidade e integridade em todos os fluxos

4. Modelo de Segurança

No modelo de segurança descrito neste trabalho, propõe-se um conjunto de primitivas e um protocolo de uso que permitam endereçar os requisitos de segurança[1] envolvidos no fluxo operacional de um sistema OddCI.

4.1. Primitivas

Tabela 2. Primitivas Básicas

PRIMITIVA	DESCRIÇÃO
H = Hash(m)	Calcula um *hash* não inversível para a mensagem *m*
Y = Crypt(m, k)	Cifra a mensagem *m* usando a chave *k*
X = DeCrypt(m, k)	Decifra a mensagem *m* usando a chave *k*
K = KeyGen(id_1, id_2)	Gera uma chave para uso em sessão de comunicação entre as identidades id_1 e id_2
C = SecureChannel(d)	Estabelece um canal de comunicação seguro *C* com o destino *d*. O canal poderá ser usado para envio de mensagens subseqüentes. O estabelecimento do canal seguro pré-supõe a autenticação mútua dos parceiros envolvidos
SecureSend(c, m)	Envia uma mensagem *m* usando o canal seguro *c*
M = SecureReceive(c, m)	Recebe uma mensagem *M* usando o canal seguro *c*
P_{id}=PublicKey(id)	Retorna P_{id}, a chave pública associada à identidade *id*
S = Sign(m, k)	Assina a mensagem *m* usando a chave privada *k*
Verify(m, id)	Verifica a autenticidade e integridade da mensagem assinada *m* pelo autor *id* e retorna TRUE caso a assinatura seja autêntica e integra ou FALSE caso contrário
Auth(id)	Verifica a autenticidade da identidade *id* mediante algum protocolo síncrono de autenticação baseado em troca de certificados ou um protocolo de desafios
Challenge(m)	Prepara a mensagem *m* para atuar como um desafio em um controle fim-a-fim entre aplicações. Seguindo algum protocolo ou frequência, mensagens normais serão substituídas por mensagens especiais com a finalidade única de validar o lado cliente
Validate(m)	Valida a mensagem *m* recebida em um regime de controle fim-a-fim entre aplicações. A comparação da resposta recebida com a resposta correta esperada servirá para validar o lado cliente, inibindo a ação de sabotadores e falhas não intencionais
I = FormatImage(e, d, a)	Cria uma imagem *I* a partir do executável *e*, dos dados *d* e da aplicação *a*
O = CreateInstance(c)	Solicita a criação de uma instância OddCI *O* através do canal seguro *c*. Assume-se que o canal seguro é estabelecido com um elemento do tipo *Provider*
Broadcast(c, m)	Envia pelo canal de broadcast c a mensagem m
ProcessID(p, id)	Vincula um processo p à identidade id através de algum mecanismo que permita a inserção de tokens embutidos no código fonte de um processo

[1] Não está contemplado no modelo proposto o tratamento de ameaças físicas de nenhuma natureza nem ameaças em nível de corrupção de *hardware* ou *software* básico, reuso de memória ou acesso direto a registradores internos, não sendo possível garantir total isolamento e privacidade em cenários onde há a possibilidade de comprometimento dos dispositivos envolvidos.

As primitivas de segurança necessárias para o atendimento dos objetivos de segurança identificados na seção anterior estão relacionadas na Tabela 2. Neste documento foi assumido que tais primitivas representam apenas conceitos e não funções atômicas, mas que são plenamente suportadas computacionalmente pelos recursos existentes em cada componente de um Sistema OddCI.

4.2. Protocolo de Uso

Fundamentalmente, o modelo de segurança que estamos propondo é baseado em camadas de "envelopes" criptográficos e técnicas de controle fim-a-fim que permitem ativar autenticação, confidencialidade e também proteção contra falhas e sabotagens. Mesmo considerando que uma cadeia de confiança [SANTIN 2002] possa ser estabelecida entre os atores, a privacidade de comunicação em cada um dos fluxos identificados na Fig. 4 também deve ser garantida. A seguir, iremos detalhar o protocolo de utilização do modelo de segurança proposto em um Sistema OddCI.

Seja *I={c, p, o, b, P}* uma instância OddCI onde *c* é o cliente, *p* é o provedor, *o* é o controlador, *b* é a retaguarda e *P* é o conjunto de agentes processadores (receptores).

No nosso modelo, o cliente solicita ao provedor a criação de uma instância OddCI. Assumindo que obteve sucesso, o provedor retorna um envelope contendo um identificador único da instância criada (*OddCI_ID*). O cliente arbitra uma chave a ser usada na comunicação com a retaguarda para acesso as tarefas e resultados (*BackendKey*) e embute esta chave no executável da sua aplicação que rodará nos agentes em P. O cliente acrescenta nos dados da aplicação informações sobre os endereços dos *hosts* que compõem a infraestrutura de retaguarda. A infraestrutura de retaguarda usará a mesma chave para autenticar os dispositivos que em breve se conectarão para estabelecer um canal seguro de comunicação para recepção de eventuais resultados e envio de novas tarefas. Em seguida, um envelope é criado pelo cliente para conter os dados da sua aplicação, o qual é enviado para o provedor *p*. Salienta-se que o estabelecimento do canal seguro pré-supõe a autenticação mútua dos parceiros, como apresentado na Tabela 1. A sequência de primitivas abaixo representa o que foi discutido.

```
sc_provider = SecureChannel( p )
OddCI_ID  = createInstance( sc_provider )
executável = ProcessID( executável, BackendKey )
AppImage = FormatImage( executável, Crypt( dados, BackendKey ), OddCI_ID,
BackendInfo )
SecureSend( sc_provider, AppImage )
```

Do lado do provedor *p*, a mensagem do cliente *c* é recebida de forma confidencial, da seguinte forma:

```
sc_client = SecureChannel( c )
AppImage = SecureReceive( sc_client )
```

O passo seguinte para o provedor *p* é repassar para o controlador *o* uma mensagem de controle contendo a AppImage e instruções sobre o tipo de instância a ser criada.

```
ControlMessage = Formata( AppImage, params, OddCI_ID )
sc_controller = SecureChannel( o )
SecureSend( sc_controller, ControlMessage )
```

O controlador *c*, no fluxo (**2**), recupera a mensagem de controle, gera uma chave randômica exclusiva (*InstanceKey*) para a instância *OddCI_ID* e a embute no código do PNA. Na prática essas informações servirão de credenciais para autenticar cada PNA, de maneira que o *controlador* apenas aceitará como participante da instância o PNA que apresentar a *InstanceKey* correta como credencial. Em seguida, o controlador formata, cifra

e depois assina a aplicação recebida do provedor e a propaga através do canal de radiodifusão para todos os receptores conectados ao serviço.

```
sc_provider = SecureChannel( p )
ControlMessage = SecureReceive( sc_provider )
InstanceKey = Random( OddCI_ID )
PNA = ProcessID( PNA, InstanceKey )
ControlMessage = Formata( ControlMessage, PNA )
M = Crypt( Sign( ControlMessage, Kpriv_c )
SignControlMessage = Sign( M, Kpriv_c )
Broadcast( BroadcastChannel, SignControlMessage )
```

Todos os dispositivos conectados ao canal de *broadcast* recebem a mensagem que contém a aplicação assinada. Conforme o modelo OddCI [COSTA 2009], o dispositivo fará a validação da mensagem usando a chave pública da *controlador* que está autenticada por uma autoridade certificadora bem conhecida. Validada a mensagem pelo dispositivo, o PNA é carregado e faz a comunicação com o *Controller* usando o identificador *InstanceKey*, que traz embutido, como chave para garantir a autenticação e o sigilo no fluxo (**4**).

O passo seguinte do PNA, caso seja aceito pelo *Controller*, é iniciar a aplicação cliente propriamente dita, que de posse da chave *BackendKey*, pode finalmente abrir o primeiro envelope criado pelo *Client* e recuperar os dados da aplicação. Esta mesma chave é usada como identificador para estabelecer um canal seguro com a retaguarda através do fluxo (**5**). Por sua vez, o fluxo (**6**), entre o *Backend* e o *Client*, é bem específico e pode usar mecanismos próprios.

As chaves embutidas na aplicação cliente *(BackendKey)* e no *PNA (InstanceKey)*, criadas de forma exclusiva e independente pelo *Client* para cada aplicação e pelo *Controller* para cada instância OddCI representam uma adaptação do conceito de *trusted process* proposto por Bell/LaPadula [BELL 1976][LUNT 1998] e permitem que os elementos voláteis do sistema possam ser validados, neste caso duplamente. Embora estas chaves específicas tenham um ciclo de vida curto e estejam embutidos nos respectivos executáveis, ainda representam uma fragilidade. Estas são as únicas chaves potencialmente acessíveis a partir de nós remotos que poderiam ser obtidas via engenharia reversa dos executáveis ou varredura de memória em dispositivos comprometidos. Entretanto, técnicas como a apresentada em [BOESGAARD 2007] podem ser utilizadas para tornar muito mais improvável que ataques deste tipo sejam bem sucedidos.

Além destes mecanismos, nos fluxos (**4**) e (**5**) são aplicados o tratamento de falhas bizantinas [LIMA 2009] e técnicas de controle de sabotagem [SARMENTA 2001] encapsuladas em controles semânticos fim-a-fim. Usando controles deste tipo, o *Backend* pode enviar tarefas especiais e conferir os resultados recebidos para validar cada PNA ou criar certa quantidade *r* de réplicas das tarefas e enviar para serem processadas por mais de um PNA. Somente quando um número *r-n* de resultados convergirem, a tarefa é considerada completa. Os valores de *r* e *n* podem ser manipulados para se adaptarem a contextos com maior ou menor grau de suscetibilidade a ataques de adversários. E o PNA que devolver algum resultado espúrio será descartado. A estratégia de controle fim-a-fim adotada, independentemente da sua forma de implementação, deverá ficar localizada na distribuição de tarefas e recolhimento de resultados pelo *Backend* específico de cada aplicação cliente através do uso das primitivas *Challenge* e *Validate*.

4.3. Proteções Complementares

Além das proteções já implementadas nativamente pelo dispositivo hospedeiro (ex. *middleware* de um receptor de TV digital), mecanismos de isolamento da execução tanto do PNA quanto da aplicação cliente são implementados usando o conceito de *Dynamic*

Virtual Environments (DVE) [KEAHEY 2004]. Estes ambientes disponibilizam uma quantidade controlada de recursos para a aplicação em execução, garantindo dessa forma, a manutenção dos principais serviços dos dispositivos hospedeiros e preservando a plataforma de possíveis ataques de alocação de recursos.

As ameaças de Denial of Service (DoS) [CHAMPAGNE 2006] podem estar presentes em sistemas OddCI através de ataques externos disparados contra o *Controller* e o *Backend*. Isso pode ser minimizado pela divulgação de endereços IP apenas através das mensagens de controle. Esta é uma prerrogativa que os sistemas OddCI possuem ao possibilitar a montagem sob demanda e de caráter volátil de infraestruturas computacionais de grande porte.

5. Estudo de Caso: Um sistema OddCI sobre uma rede de TV Digital

Para instanciar a arquitetura OddCI sobre uma rede de televisão digital, OddCI-DTV, é necessário implementar os quatro componentes de *software* discutidos, isto é: o *Provider*, o *Controller*, o *Backend* e o PNA. O papel do provedor pode ser exercido pela emissora de TV digital que detém a concessão do canal de TV ou um parceiro. O papel do controlador deve ser exercido pela emissora de TV digital que detém a concessão do canal de TV, uma vez que ela deve enviar as aplicações (imagens dos cientes) para os receptores através de um fluxo elementar junto com sua programação. A retaguarda pode ser montada como um conjunto de computadores distribuídos sob controle do cliente ou de um terceiro. Os agentes processadores são os receptores de TV digital, cujos canais de retorno usados para interatividade possibilitam o envio e recebimento de dados com o *Backend* e o *Controller*. Na Fig. 4, são identificadas as tecnologias atualmente disponíveis para o segmento de TV Digital que podem ser usadas e como elas estão associadas com os elementos da arquitetura OddCI genérica.

O processo de distribuição de execução de aplicações interativas, descritos no padrão Brasileiro de TV Digital, ocorre da seguinte forma: inicialmente o conteúdo da aplicação é serializado na forma de um carrossel de objetos no padrão DSM-CC (*Digital Storage Media Command and Control*) [ISO 1995], onde os arquivos e pastas da aplicação são codificados em sessões e encapsulados em um fluxo (*stream*) MPEG2-TS [ITU 2000]. Após a codificação dos dados, propriedades da aplicação como nome, tipo, classe principal e outras características são definidas e estruturadas na forma da tabela AIT (*Application Information Table*) e encapsulados em pacotes TS. Terminada a preparação dos dados, ocorre a configuração da tabela PMT (*Program Map Table*) com o PID utilizado pelo TS de dados (*Object Carousel*) e o PID da AIT, além da adição dos descritores necessários para identificar a existência de um fluxo de dados para um determinado programa ou serviço. Por fim, o *stream* é multiplexado com outros fluxos de áudio, vídeo e dados para então ser transmitido em *broadcast* pela emissora de TV.

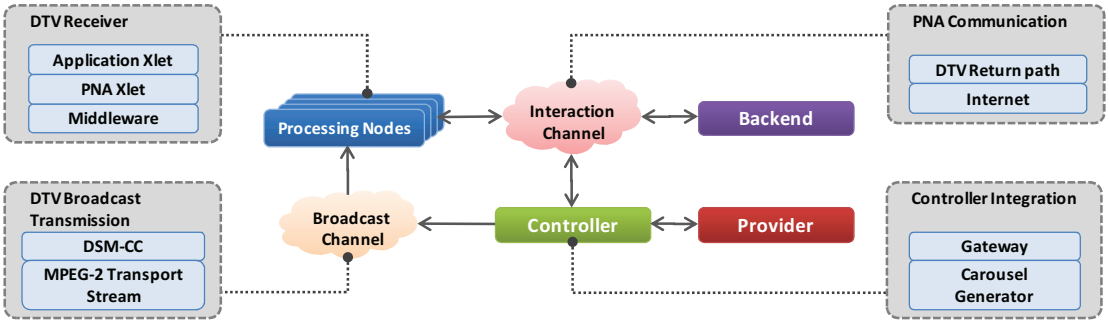

Figura 4. Tecnologias atuais de TV Digital para suporte a sistemas OddCI-DTV

Ao sintonizar a frequência da emissora, o receptor de TV verifica a existência do *stream* de dados, e executa uma rotina de processamento desses dados, que é responsável por verificar a integridade do conteúdo recebido através do CRC de cada informação. Os dados são gravados obedecendo à estrutura de pastas e arquivos configurados na AIT. Ao término do processamento, o *middleware* é notificado da existência de uma nova aplicação passando informações sobre o nome, o tipo e o modo de execução da aplicação para o gerenciador de aplicações que seleciona o módulo de apresentação (*engine*) adequado ao tipo de aplicação: NCL/Lua [ABNT 2009-A] ou Java DTV [ABNT 2009-B].

O esforço em identificar e decompor as vulnerabilidades presentes e mapeá-las em primitivas básicas permite aplicar as mesmas técnicas já validadas em outros contextos. Com tal estratégia, foi possível relacionar as primitivas necessárias com as primitivas esperadas para os sistemas de TV Digital que atendam as normas estabelecidas. Desta forma, foi possível constatar que as primitivas necessárias para o funcionamento seguro de um sistema OddCI podem ser implementadas em um sistema de TV Digital, ou porque já fazem parte das bibliotecas padrão ou porque podem ser construídas sobre estas.

A validação foi baseada no *middleware* Ginga [ABNT 2009-C], que define as linguagens que podem ser utilizadas para codificação das aplicações e as interfaces de programação (APIs – *Application Program Interfaces*) disponíveis. Parte das primitivas de segurança descritas no modelo de segurança aqui definidos foram implementadas nas linguagens NCL/Lua [ABNT 2009-A] e Java DTV [ABNT 2009-B] ou mapeadas para recursos nativos destes ambientes. Tomando por exemplo uma aplicação Java DTV, uma API de segurança complementar é especificada no pacote *com.sun.dtv.security* que estende a API *java.security*. De forma similar, as aplicações implementadas em NCL/Lua, é possível fazer uso da biblioteca aberta Lua MD5 [KEPLER 2010], que já contempla uma implementação dos algoritmos MD5 e des56. Conforme a norma, o *middleware* que está instalado nos receptores também fará automaticamente a validação da cada aplicação recebida usando a chave pública da emissora que está assinada por uma autoridade certificadora bem conhecida. Além disso, os recursos nativos que estão previstos para proteger os recursos e o funcionamento do *middleware* de aplicações de comportamento indevido, intencional ou não, foram mapeados para obter o mecanismo de DVE previsto.

Também foram implementadas algumas das primitivas para uso em aplicações instaladas diretamente nos receptores, denominadas aplicações residentes. Neste caso, foram usadas rotinas de segurança disponibilizadas pelo sistema operacional do próprio receptor de TV e também pelas funcionalidades acessíveis através do *middleware*. No nosso estudo de caso, utilizamos uma versão de kernel Linux desenvolvido para processadores MIPS + *busybox*.

Na Tabela 3, é ilustrado como algumas das primitivas de segurança foram mapeadas para uso em aplicações Java, NCL/Lua e Residente.

Tabela 3. Exemplo de primitivas básicas em Java, Lua e Aplicações Residentes

Primitiva	Java	NCL/Lua	Residente (C++)
Hash(msg)	*public byte[] getHash(byte[] m);*	*security:getHash(m: string)→hash: string*	*char * getHash(char * m);*
Crypt(m, k)	*public byte[] crypt(byte[] m, String k);*	*security:crypt(m:string) →m_crypt: string*	*char * crypt(char * m, char * key);*
DeCrypt(m,k)	*public byte[] decrypt(byte[] m, String k);*	*security:decrypt(m:string) →m_decrypt: string*	*char * decrypt(char * m, char * key);*
Sign(msg,id)	*public byte[] sign(byte[] m, int id);*	*security:sign(m:string) →m_sign: string*	*char * sign(char * m, int id);*
GenKey(id$_1$ id$_2$)	*public String getKey(int id1, int id2);*	*Security:getKey(id1,id2: number)→key: string*	*char * getKey(int id1, int id2);*

6. Conclusão

O presente trabalho propõe um modelo de segurança para que sistemas OddCI, em geral, e sistemas OddCI-DTV, em particular, possam ser usadas de forma segura para processar aplicações MTC. Os diversos desafios envolvidos na operação de tais sistemas baseados em recursos públicos e não dedicados foram discutidos. Foi definido um mecanismo de comunicação seguro para a oferta de serviços de computação sobre uma infraestrutura distribuída volátil e demonstrada a viabilidade de sua implementação em uma rede de receptores de TV Digital.

Os próximos passos da pesquisa incluem uma generalização da proposta com objetivo de tornar possível o uso de diferentes tipos de dispositivos para montagem de um Sistema OddCI (e.g. dispositivos móveis), a realização de simulações das operações dos fluxos de comunicação juntamente com a condução de experimentos reais para avaliar o desempenho do protocolo proposto e das primitivas usadas, bem como o esforço de coordenação necessário para controlar e usar potencialmente milhões de dispositivos simultaneamente. Uma análise mais sólida dos aspectos de segurança aqui introduzidos também está prevista, com a definição dos modelos de atacantes aos quais o sistema está exposto e a aferição da eficiência das contramedidas previstas.

O uso dos recursos de processamento do receptor de TV para executar as tarefas dos clientes pode ser negociado nos contratos de *leasing* em sistemas de TV por assinatura. No caso de sistemas de TV abertos entende-se que ao sintonizar um canal, o proprietário do receptor cede o uso de seus recursos – processador, memória, display etc. – ao emissor de TV que, em troca, permite o acesso gratuito ao conteúdo por ele produzido ou adquirido. Uma discussão detalhada sobre possíveis modelos de negócio e modelos de incentivo que estimulem a participação de usuários e emissoras em sistemas OddCI também será tema de trabalho futuro.

7. Referências

ABNT (2009-A) NBR 15606-2. Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 2.

ABNT (2009-B) NBR 15606-4. “Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 4.

ABNT (2009-C) NBR 15606-1. “Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 1.

Amazon (2010) Amazon Elastic Compute Cloud - Amazon EC2. Disponível em http://aws.amazon.com/ec2/

Anderson, D. P. (2004) BOINC: A System for Public-Resource Computing and Storage. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), pp. 4-10.

Anderson, D. P. et al. (2002) SETI@Home An Experiment in Public-Resource Computing. Communications of the ACM Archive, vol. 45(11), pp. 56—61.

Andrade, N., Brasileiro, F., Cirne, W. and Mowbray, M. (2007) Automatic grid assembly by promoting collaboration in peer-to-peer grids. In Journal of Parallel and Distributed Computing. vol. 67(8).

Bell, D. E. and LaPadula, L. J. (1976) Secure Computer Sytems: Unified Exposition and Multics Interpretation. Technical Report ESD–TR–75–306, MITRE Co. Hanscom , MA.

Boesgaard, M. and Zenner, E. (2007) Protecting Online Transactions with Unique Embedded Key Generators. In Second International Conference on Availability, Reliability and Security (ARES'07).

Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M. and Pinkas, B. (1999) Multicast Security: A Taxonomy and Some Efficient Constructions" In Infocom '99.

Champagne, D. and Lee, R. B. (2006) Scope of DDoS Countermeasures: Taxonomy of Proposed Solutions and Design Goals for Real-World Deployment. In 8th ISSIS (SSI'2006).

Cirne, W., Paranhos, D., Costa, L. et al (2003) Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In Proceedings of the ICCP'2003 - International Conference on Parallel Processing.

Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R. and Mowbray, M. (2006) Labs of the world, unite!!!. Journal of Grid Computing. vol. 4(3)), pp. 225—246.

Costa, R., Brasileiro, F., Filho, G. L., and Sousa, D. M. (2009) OddCI: on-demand distributed computing infrastructure. In Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers (Portland, Oregon). MTAGS '09. ACM.

D'Anna, L., Matt, B., Reisse, A., Van Vleck, T., Schwab, S. and LeBlanc, P. (2003) Self-protecting mobile agents obfuscation report. Technical Report #03-015, Network Associates Labs, June 2003.

ISO/IEC (, 1995) 13818-6: Digital Storage Media Command and Control.

ITU-T (2000) Recommendation H.222. Information Technology. Generic coding of moving pictures and associated audio information: Systems.

Keahey, K., Doering, K. and Foster, I. (2004) From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In 5th International Workshop in Grid Computing.

Kepler Project (2006) MD5 Cryptographic Library for Lua. Disponível em http://www.keplerproject.org/md5/.

Lima, M. S. and Greve, F. G. P. (2009) Detectando Falhas Bizantinas em Sistemas Distribuídos Dinâmicos.

Litzkow, M. Livny, M. and Mutka, M. (1988) Condor - a hunter of idle workstations. Proc. 8th Int'l Conf. Dist. Computing Systems.

Lunt, T. F., Neumann, P. G., Denning, D. et al. (1998) Secure distributed data views – vol.1: Security policy and policy interpretation for a class A1 multilevel secure. Technical Report SRI-CSL-88-8, SRI International, Menlo Park, CA.

Morris, S. and Smith-Chaigneau, A. (2005) Interactive TV Standards – A Guide to MHP, OCAP and JavaTV. ISBN-13 978-0-240-80666-2. Focal Press , Elsevier.

Oliveira, L., Lopes, L., and Silva, F. (2002) P3 (Parallel Peer to Peer): An Internet Parallel Programming Environment. Web Engineering and Peer-to-Peer Computing Workshop. Lecture Notes in Computer Science. vol. 2790, pp. 274--288. Pisa, Italy, Springer.

Raicu, I. and Foster, I. (2008) Many-Task Computing for Grids and Supercomputers. IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08).

Santin, A., Fraga, J. et al (2002) Modelo de Autorização e Autenticação Baseado em Redes de Confiança para Sistemas Distribuídos de Larga Escala. In SSI´02. S. J. Campos - SP.

Sarmenta, L. (2001) Sabotage-Tolerance Mechanisms For Volunteer Computing Systems. In Cluster Computing And Grid. Brisbane, Australia, 2001

Schellhorn, G., Reif, W., Schairer, A., Karger, P. et al. (2000) Verification of a Formal Security Model for Multiapplicative Smart Cards. Computer Security - ESORICS. Pg 17-36.

Thain, D., Tannenbaum, T. and Livny, M. (2006) How to Measure a Large Open Source Distributed System, Concurrency and Computation: Practice and Experience. vol. 8(15).

Wang, L., Von Laszewski, G., Kunze, M. and Tao, J. (2008) Cloud computing: A Perspective Study. Proceedings of the Grid Computing Environments (GCE) workshop. Austin, Texas.