Infra-Estrutura com Segurança de Funcionamento para Cooperação de Serviços *Web**

Eduardo Adílio Pelinson Alchieri[1]†, Alysson Neves Bessani[2], Joni da Silva Fraga[1‡]

[1]DAS - Departamento de Automação e Sistemas
UFSC - Universidade Federal de Santa Catarina
Florianópolis - Santa Catarina

[2]LaSIGE – Laboratório de Sistemas Informáticos de Grande Escala
FCUL – Faculdade de Ciências da Universidade de Lisboa
Lisboa - Portugal

{alchieri,fraga}@das.ufsc.br,neves@lasige.di.fc.ul.pt

Resumo. *Atualmente tem surgido um grande esforço no sentido de especificar mecanismos para cooperação de serviços web para resolução de tarefas envolvendo diversas organizações. Neste trabalho é apresentada uma infra-estrutura para coordenação de serviços web segura e confiável (tolerante a faltas e intrusões), que oferece um elevado grau de desacoplamento. Esta infra-estrutura se baseia no modelo de coordenação por espaço de tuplas e provê uma série de mecanismos de segurança, os quais permitem a cooperação de serviços web mesmo na presença de partes maliciosas. O trabalho também investiga os custos envolvidos no uso deste suporte e possíveis aplicações de interesse.*

1. Introdução

A tecnologia de *Web Services*, ou *serviços web*, tem se consolidado cada vez mais como um padrão *de facto* quando se consideram sistemas distribuídos na Internet. Esta tecnologia concretiza o modelo de computação orientada a serviços [Bichier and Lin 2006] sobre padrões largamente utilizados na *web*. A adoção desses padrões proporciona aos serviços *web* grande facilidade de uso e baixo custo de implantação. Isto se reflete no suporte maciço da indústria a esta tecnologia.

A computação orientada a serviços – e mais especificamente os serviços *web* – é uma evolução natural de tecnologias como RPC e CORBA [Object Management Group 2002], que exploram a chamada remota de procedimentos e métodos (em objetos distribuídos), respectivamente. De formam semelhante a estas tecnologias, os serviços *web* providos por uma organização devem ser descritos numa interface que pode ser entendida e invocada por outras partes em um sistema distribuído. Os grandes atrativos dos serviços *web* são sua interoperabilidade e sua simplicidade devido ao uso de um modelo conhecido (invocação remota de operações) sobre tecnologias largamente utilizadas (ex., HTTP e XML). Os padrões fundamentais usados pelos serviços *web*, todos baseados em XML, são: o SOAP (*Simple Object Access Protocol*) – protocolo para troca de mensagens entre clientes e serviços, operando sobre diversos protocolos de comunicação; o WSDL (*Web Service Description Language*) – linguagem para descrição de serviços *web*; e o UDDI (*Universal Description, Discovery and Integration*) – repositório onde os serviços *web* são registrados para que possam ser descobertos por clientes.

Sendo a interoperabilidade o ponto fundamental dos serviços *web*, não tardou para que surgissem esforços buscando definir formas adequadas de se combinar serviços, visando

*Realizado com recursos do CNPq (projeto número 550114/2005-0).
†Bolsista CAPES.
‡Bolsista Produtividade CNPq.

a cooperação na execução de tarefas envolvendo várias organizações [Bichier and Lin 2006, Peltz 2003]. Iniciativas como o WS-ORCHESTRATION [Alves A. et al. 2006] e o WS-CHOREOGRAPHY [Burdett and Kavantzas 2004] atacam justamente esse problema, propondo mecanismos para a definição e execução de tarefas complexas que envolvem várias subtarefas encapsuladas em serviços *web*.

Iniciativas como as listadas acima se concentram na integração dos serviços *web* através da especificação de um fluxo de troca de mensagens entre os mesmos (coordenação orientada a controle). Uma abordagem complementar é a coordenação dos serviços usando um repositório de dados compartilhado (coordenação orientada a dados). Nessa abordagem, os serviços cooperantes se comunicam através do uso de um repositório de dados que pode ser usado tanto para o armazenamento de dados compartilhados quanto como um mediador, oferecendo comunicação desacoplada. Um modelo de coordenação orientada a dados bastante popular é o baseado em *espaço de tuplas* [Gelernter 1985].

Neste modelo, os processos interagem através de uma abstração de memória compartilhada – espaço de tuplas – onde estruturas genéricas de dados – tuplas – são armazenadas e recuperadas. As operações básicas suportadas pelo espaço de tuplas são a inserção, a leitura e a remoção de tuplas. O grande benefício do uso deste modelo é o desacoplamento no tempo (os participantes não precisam estar ativos no mesmo instante), no espaço (os participantes não precisam se conhecer) e o seu poder de sincronização (ex. controle de concorrência).

Diversos trabalhos têm explorado o espaço de tuplas como infra-estrutura de coordenação para serviços *web* (ex. [Bright and Quirchmayr 2004, Lucchi and Zavattaro 2004, Maamar et al. 2005, Bellur and Bondre 2006]). A principal vantagem desta abordagem está no desacoplamento entre os serviços cooperantes (nem as interfaces precisam ser conhecidas). Este trabalho segue na mesma linha e propõem um suporte de cooperação para serviços *web* que, além de oferecer os benefícios inerentes aos espaços de tuplas, também é seguro e confiável.

A infra-estrutura aqui proposta, chamada WS-DEPENDABLESPACE, ou WSDS, utiliza trabalhos prévios dos autores sobre segurança de funcionamento (*dependability* [Avizienis et al. 2004]) em espaço de tuplas [Bessani et al. 2006b, Bessani et al. 2007] e incorpora novos componentes que proporcionam a integração do modelo ao mundo dos serviços *web*. A arquitetura do WSDS faz uso de *gateways* "sem estado" que agem como clientes de um espaço de tuplas confiável, repassando as requisições advindas de clientes do serviço de coordenação. Diversos mecanismos foram introduzidos nesta arquitetura para permitir que o sistema tolere faltas acidentais (paradas e *bugs*) e maliciosas (ataques e intrusões) em uma parte dos componentes do sistema. Além disso, o WSDS mantém todas as propriedades de segurança do espaço de tuplas confiável [Bessani et al. 2006b, Bessani et al. 2007] sem ferir nenhuma especificação para serviços *web*.

As principais contribuições deste trabalho são: o projeto e a implementação da infra-estrutura de coordenação WSDS, o primeiro suporte de coordenação centrado em dados, para serviços *web*, que oferece segurança de funcionamento; a avaliação dos custos envolvidos no acesso a esse tipo de infra-estrutura, através de uma análise da latência das operações e suas causas; e uma análise de alguns cenários reais em que este tipo de infra-estrutura pode ser usada, bem como sua relação com os principais padrões para cooperação entre serviços *web*.

2. Espaço de Tuplas

Um espaço de tuplas pode ser visto (conceitualmente) como um objeto de memória compartilhada que fornece operações para armazenar e recuperar conjuntos de dados ordenados chamados de tuplas. Uma *tupla* t é uma seqüência ordenada de campos, onde um campo que contém um valor é dito *definido*. Um tupla onde todos os campos são definidos é chamada de *entrada*.

Uma tupla $\bar{t}$ é chamada *molde* se algum de seus campos não tem valor definido. Diz-se que uma tupla t e um molde $\bar{t}$ *combinam* se e somente se eles têm o mesmo número de campos e todos os valores dos campos definidos em $\bar{t}$ são iguais aos valores dos campos correspondentes em t. Por exemplo, uma tupla $\langle$CLIENTE$, 12, abc\rangle$ combina com o molde $\langle$CLIENTE$, *, abc\rangle$ ('$*$' denota um campo não definido do molde).

As manipulações realizadas no espaço de tuplas consistem em invocações de três operações básicas [Gelernter 1985]: $out(t)$ que adiciona a entrada t no espaço de tuplas (inserção); $in(\bar{t})$, que remove do espaço de tuplas uma tupla que combina com o molde $\bar{t}$ (leitura destrutiva); $rd(\bar{t})$, usada na leitura de uma tupla que combina o molde $\bar{t}$, sem removê-la do espaço (leitura não-destrutiva). As operações in e rd são bloqueantes, i.e., se não houver uma tupla que combine com o molde no espaço, o processo fica bloqueado até que uma esteja disponível. Uma extensão comum a este modelo, é a inclusão de variantes não bloqueantes das operações de leitura, denominadas inp e rdp. Estas operações funcionam exatamente como as anteriores, a não ser pelo fato de retornarem mesmo não havendo uma tupla que combine com o molde usado (indicando esta inexistência). Note que, de acordo com as definições anteriores, o espaço de tuplas funciona como uma *memória associativa*: os dados são acessados a partir de seu conteúdo, e não através de seu endereço.

Visando aumentar o poder de sincronização do espaço de tuplas, consideramos também a operação $cas(\bar{t}, t)$ (*conditional atomic swap*) [Segall 1995]. Esta operação funciona como uma execução indivisível do código: **if** $\neg rdp(\bar{t})$ **then** $out(t)$. Sendo assim, a tupla t será inserida no espaço somente se $rdp(\bar{t})$ não retornar alguma tupla, i.e., se não existir uma tupla no espaço que combine com $\bar{t}$. A operação cas é importante porque permite que o espaço de tuplas que a suporta seja capaz de resolver o problema do consenso em sistemas assíncronos [Segall 1995], o qual é a base para a solução de muitos problemas de sincronização distribuída.

3. DEPSPACE: Um Espaço de Tuplas com Segurança de Funcionamento

Segurança de funcionamento é uma característica fundamental dos sistemas ditos confiáveis e seguros [Avizienis et al. 2004]. Esta característica é composta por diversos atributos. Quando consideramos um espaço de tuplas com segurança de funcionamento, os seguintes atributos se fazem necessários: *confiabilidade* (as operações realizadas no espaço de tuplas fazem com que seu estado se modifique de acordo com sua especificação), *disponibilidade* (o espaço de tuplas sempre está pronto para executar as operações requisitadas por partes autorizadas), *integridade* (nenhuma alteração imprópria no estado de um espaço de tuplas pode ocorrer), *confidencialidade* (o conteúdo de campos de uma tupla não podem ser revelados a partes não autorizadas).

O DEPSPACE é a implementação de um espaço de tuplas confiável e seguro, que satisfaz todas as propriedades de interesse da segurança de funcionamento [Bessani et al. 2007]. Um aspecto chave do serviço oferecido pelo DEPSPACE é o suporte a múltiplos espaços de tuplas lógicos, i.e. o sistema fornece interfaces administrativas que permitem a criação de espaços de tuplas, os quais não tem relação nenhuma uns com os outros. Para satisfazer todos os atributos da segurança de funcionamento, o DEPSPACE utiliza uma combinação de diversos mecanismos, os quais são descritos a seguir[1].

Replicação tolerante a faltas bizantinas. No DEPSPACE, o espaço de tuplas é mantido replicado em um conjunto de n_r servidores de modo que falhas (por parada ou maliciosas) em alguns deles (no máximo f_r, sendo $n_r \geq 3f_r + 1$) não ferem nenhum atributo de segurança de funcionamento do sistema. Este conjunto de servidores utiliza a replicação

[1] O DEPSPACE usa também criptografia para garantir confidencialidade [Bessani et al. 2006a]. Esta funcionalidade não é descrita, e nem tampouco sua integração ao WSDS, por limitações de espaço.

Máquina de Estados, uma solução clássica para implementar sistemas tolerantes a faltas bizantinas [Schneider 1990, Castro and Liskov 2002].

Controle de acesso. Este mecanismo é usado para impedir que clientes não autorizados possam executar operações no espaço de tuplas. O controle de acesso é um mecanismo fundamental para a manutenção da integridade e da confidencialidade das informações manipuladas pelo sistema. Atualmente, o DEPSPACE implementa controle de acesso de duas formas:

- **Baseado em credenciais:** para cada tupla inserida no DEPSPACE é possível definir quais são as credenciais necessárias para acessá-la, tanto para leitura quanto para remoção. Estas credenciais são definidas pelo processo que insere a tupla. Existem dois níveis de acesso, sendo possível também definir quais são as credenciais necessárias para inserir uma tupla no espaço.
- **Políticas de granularidade fina:** o DEPSPACE suporta a definição de políticas de acesso de granularidade fina [Bessani et al. 2006b]. Estas políticas controlam o acesso ao espaço de tuplas considerando três parâmetros: o identificador do cliente, a operação que será executada (juntamente com seus argumentos) e o estado do espaço. Um exemplo de política é: *"uma operação $out(\langle CLIENTE, id, x\rangle)$ só pode ser executada se não houver nenhuma tupla que combina com $\langle CLIENTE, id, *\rangle$ no espaço"*. Esta regra não permite a inserção de duas tuplas que representam clientes com um segundo campo igual (mesmo identificador do cliente).

As credenciais requeridas para inserção de tuplas em um espaço de tuplas e sua política de segurança são sempre definidas no momento em que o espaço é criado.

4. WS-DEPENDABLESPACE

Esta seção descreve o WSDS. Primeiramente, algumas premissas e garantias são apresentadas. Após isso, a arquitetura e os princípios básicos de funcionamento do sistema são abordados, para então discutirmos os mecanismos específicos integrados à arquitetura, os quais visam a solução de alguns problemas de segurança que podem ocorrer.

4.1. Premissas e Garantias

Os processos do sistema são divididos em três conjuntos: n_r servidores DEPSPACE $U = \{s_1, ..., s_{n_r}\}$, n_g *gateways* de acesso $G = \{g_1, ..., g_{n_g}\}$ e um conjunto ilimitado de clientes $\Pi = \{c_1, c_2, ...\}$. Os *gateways* são os únicos processos do sistema que necessariamente exportam interfaces WSDL, sendo portanto, serviços *web*. Os clientes se comunicam com os *gateways* através de mensagens SOAP, e estes se comunicam com os servidores através de canais confiáveis e autenticados, que também são utilizados na comunicação entre os servidores[2].

Assumimos o modelo de sistema com sincronia terminal (*eventually synchronous system model*) [Dwork et al. 1988]. Este modelo estipula que em todas as execuções do sistema, existe um limite Δ e um instante de tempo GST (*Global Stabilization Time*) de tal forma que toda mensagem enviada por um processo correto após um instante $u >$ GST é recebida antes de $u + \Delta$. É importante ressaltar que apesar do modelo estipular a existência destes limites, nenhum processo do sistema precisa conhecê-los, e nem tampouco precisam ser os mesmos em diferentes execuções do sistema. A idéia por trás deste modelo é de que o sistema trabalha de forma assíncrona (não respeitando nenhum limite de tempo) a maior parte do tempo. Porém, durante períodos de estabilidade, o tempo para transmissão de mensagens é limitado. Note que este modelo reflete o comportamento da Internet: a latência de comunicação na maior parte do tempo é estável, porém não existe um limite para este valor. Assumimos também que todas as computações locais requerem intervalos de tempo negligenciáveis. Esta premissa se

[2]Estes canais podem ser facilmente implementados através de SSL/TLS [Dierks and Allen 1999].

fundamenta no fato de que, mesmo que o tempo requerido para algumas operações locais seja considerável, as computações locais estão menos sujeitas à interferências externas, e portanto sua característica assíncrona acaba sendo pouco válida na prática.

Todos os processos do sistema estão sujeitos a falhas bizantinas [Lamport et al. 1982]: um processo que apresenta este tipo de falha pode exibir qualquer comportamento, podendo parar, omitir envio ou entrega de mensagens, ou desviar de sua especificação arbitrariamente. Um processo que apresenta comportamento de falha é dito falho, de outra forma é dito correto. Neste trabalho assumimos independência de falhas nos processos: a probabilidade de um processo sofrer uma falha é independente da probabilidade de outro processo sofrer uma falha. Esta propriedade pode ser alcançada através do uso sistemático de diversidade [Obelheiro et al. 2005].

Assumimos também que cada processo do sistema tem um par de chaves pública-privada, sendo a chave privada conhecida apenas pelo próprio processo. As chaves públicas de todos os processos do sistema são conhecidas por todos os outros processos (através de certificados). Essas chaves são usadas na produção e verificação de assinaturas digitais, nos mesmos moldes do RSA [Rivest et al. 1978].

Em termos de garantias, o WSDS permanece correto (provendo um espaço de tuplas que satisfaz os atributos de segurança de funcionamento) enquanto no máximo $f_r \leq \lfloor \frac{n_r+1}{3} \rfloor$ servidores DEPSPACE (menos de um terço, o ótimo para esse tipo de replicação [Castro and Liskov 2002]), $f_g \leq n_g - 1$ *gateways* (no mínimo um correto) e um número ilimitado de clientes falhem.

4.2. Arquitetura e Princípio de Funcionamento

A figura 1 apresenta a arquitetura do WSDS, onde é possível observar que os *gateways* ligam os clientes aos servidores do DEPSPACE. Para isso, disponibilizam a interface de seu serviço em um repositório UDDI para que os clientes possam acessá-los. Conforme já discutido, os clientes e os servidores têm pares de chaves públicas e privadas. As chaves públicas de todos os processos são disponibilizadas através de certificados facilmente verificáveis.

O principal componente introduzido nesta arquitetura é o *gateway web service* (WSG), doravante chamado apenas *gateway*, o qual é um serviço *web* que funciona como "ponte" entre os clientes (deste serviço) e o espaço de tuplas replicado (DEPSPACE), recebendo as mensagens SOAP vindas destes clientes e transformando-as em invocações ao DEPSPACE.

Deste modo, quando um cliente deste serviço for acessar o espaço de tuplas, primeiramente deve consultar um serviço UDDI (disponível na Internet) visando obter um ou mais endereços de *gateway*[3]. A partir daí, o cliente envia sua requisição para um dos *gateways*, o qual, por sua vez, a encaminha ao DEPSPACE usando um protocolo de difusão com ordem total [Castro and Liskov 2002]. Após o processamento da requisição, os servidores respondem ao *gateway* que espera por $2f_r+1$ (n_r-f_r) respostas para então encaminhá-las ao cliente. O cliente obtém a resposta da requisição verificando qual resposta foi enviada por $f_r + 1$ servidores.

O *gateway* não realiza nenhum processamento com o conteúdo das requisições ou das respostas, a não ser as transformações entre os dois "mundos" (SOAP e Java) descrita anteriormente e de reunir as $n_r - f_r$ respostas à requisição antes de enviá-las ao cliente. Além disso, este serviço não possui estado (*stateless*), sendo portanto desnecessário realizar qualquer sincronização entre os n_g *gateways* do sistema.

4.3. Lidando com *Gateways* Falhos

Apesar da aparente simplicidade da arquitetura, com a utilização deste elemento "ponte" surgem alguns problemas quando consideramos um sistema tolerante a faltas e a intrusões. Estes

[3]Para melhorar a disponibilidade do sistema, os *gateways* podem ser registrados em mais de um serviço UDDI e/ou em um serviço que seja tolerante a faltas bizantinas.

problemas são descritos nesta seção, juntamente com a solução empregada em suas resoluções.

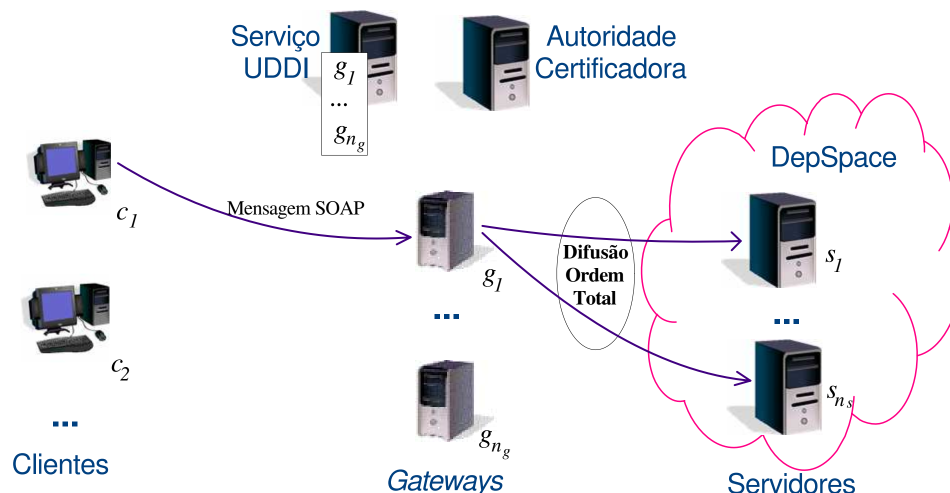


Figura 1. Arquitetura e príncipio de funcionamento do WSDS.

4.3.1. Autenticidade e Integridade das Requisições e das Respostas

Um primeiro problema que surge na arquitetura da figura 1 reside no fato de que um *gateway* falho pode solicitar a execução de falsas requisições (que não foram enviadas por algum cliente). Além disso, poderá alterar requisições, modificando alguns de seus dados (parâmetros, tipo de operação, etc...). Da mesma maneira, poderá forjar ou modificar as respostas dos servidores. A questão principal aqui é garantir que somente requisições (inalteradas) decorrentes de invocações de clientes e respostas (inalteradas) produzidas por servidores sejam processadas.

Outro ponto a destacar, é que o cliente precisa enviar suas credenciais junto com as requisições, pois as mesmas são necessárias para que os servidores verifiquem se o mesmo possui permissão para acesso ao espaço e/ou à tupla acessada. Estas credenciais são enviadas aos servidores através de um certificado digital relacionado com o cliente, o qual também é utilizado para provar a autenticidade das requisições. Deste modo, cada cliente deverá obter um certificado digital (junto a uma autoridade certificadora reconhecida) e assinar suas requisições. Os servidores apenas executarão determinada operação se esta assinatura for válida de acordo com o certificado correspondente (enviado pelo cliente em sua primeira mensagem para o servidor).

Com o intuito de provar que uma resposta é autêntica, a mesma deve ser assinada pelos servidores com sua chave privada. Assim, os clientes poderão verificar a autenticidade das respostas utilizando as chaves públicas dos servidores, também contidas em certificados válidos que acompanham as respostas. As assinaturas das requisições e das respostas garantem a autenticidade e a integridade na comunicação fim-a-fim, i.e., nada pode ser alterado pelo *gateway* sem ser detectado pelos servidores do DEPSPACE ou clientes. As verificações dos certificados que acompanham tanto as requisições como as respostas podem ser feitas com procedimentos padrões (ex. PKI X.509).

4.3.2. Atendimento Incompleto de Requisições

Nem sempre um cliente consegue que sua requisição seja corretamente executada (atendida) caso esteja usando um *gateway* falho. Para que isso ocorra, basta que este *gateway* não envie as respostas ao cliente, fazendo com que o mesmo fique bloqueado indefinidamente a espera destas. Além disso, na execução de operações de remoção de tuplas, o *gateway* acessado poderá remover as tuplas do DEPSPACE e não enviar as respostas ao cliente, fazendo com que a tupla

"desapareça" do espaço sem ser consumida por cliente algum. A solução completa deste problema de desaparecimento de tuplas envolve a combinação do mecanismo descrito nesta seção com o de eliminação de requisições duplicadas (seção 4.3.3).

Para resolver estes problemas, um *timeout* é associado a cada envio de requisição no cliente. Caso acontecer o *timeout* de uma requisição e a resposta ainda não foi obtida, o cliente solicita a execução desta requisição a outros f_g *gateways*, garantindo que acessou pelo menos um *gateway* correto. Deste modo, a execução desta requisição estará completa quando o cliente receber o primeiro conjunto de respostas válidas de um *gateway* e conseguir determinar a resposta. Este mecanismo deverá ser acionado sempre que uma resposta não possa ser determinada, seja pelo *timeout* ou por falhas nas verificações de assinaturas dos servidores.

No caso das operações bloqueantes (rd e in) não é possível determinar se o cliente ainda não recebeu as respostas pelo fato do *gateway* acessado ter falhado ou não existir uma tupla no espaço que combine com o molde usado na operação, o que impossibilita o uso de *timeouts* para estas operações. A solução para este problema é fazer com que o cliente envie estas solicitações para $f_g + 1$ *gateways* diferentes e determine a resposta como descrito anteriormente (casos onde ocorre o *timeout* para a execução de uma requisição). O funcionamento correto destas operações depende, é claro, de que se assuma comunicações confiáveis entre clientes e *gateways* (ex., uso do SOAP sobre o protocolo HTTP).

4.3.3. Requisições Duplicadas

Com o mecanismo descrito na seção anterior é possível perceber que, como a requisição poderá ser enviada a mais de um *gateway*, possivelmente será enviada mais de uma vez aos servidores DEPSPACE. Além disso, um *gateway* faltoso poderá solicitar a execução de uma operação já executada pelo DEPSPACE (ataque de *replay*). Nestes casos, é necessário que estas requisições duplicadas sejam eliminadas para manter a consistência do estado do espaço de tuplas. Para que isto seja possível, o cliente deve identificar cada requisição de forma única, através de sua identidade e de um número de seqüência, assim os servidores poderão verificar se tal requisição pode ser executada.

Para este fim, cada servidor contém um *buffer* onde são armazenadas as respostas correspondentes a última requisição de cada cliente. Assim, uma requisição é executada por um servidor apenas se ela tem um número de seqüência uma unidade maior que a invocação cuja resposta está no *buffer*. Caso a requisição recebida pelos servidores tenha o mesmo identificador da última executada para este cliente, apenas a resposta armazenada no *buffer* é enviada ao *gateway* solicitante. Em qualquer outro caso a requisição é descartada. Deste modo, um cliente não pode solicitar a execução de uma requisição sem que a anterior esteja completamente atendida[4]. Este mecanismo, além de evitar que uma requisição seja executada mais de uma vez, resolve o problema do desaparecimento de tuplas devido ao atendimento incompleto de requisições, pois quando o cliente solicitar a reexecução de uma operação através de outro *gateway*, a mesma resposta (com a mesma tupla) estará nos *buffers* dos servidores pronta para ser enviada.

4.4. Implementação

Todas as implementações foram realizadas utilizando a linguagem de programação Java e a implementação do DEPSPACE já disponível[5]. Deste modo, as operações criptográficas utilizam

[4]Note que esta limitação pode ser relaxada para k requisições se o servidor armazenar as últimas k respostas a cada cliente.

[5]Disponível em `http://www.das.ufsc.br/~neves/jitt/`

a biblioteca de criptografia do Java 1.5 (*Java Cryptography Extensions*). As assinaturas são implementadas usando os algoritmos *SHA-1* e *RSA* [Rivest et al. 1978] para resumos e criptografia assimétrica, respectivamente. Os canais confiáveis e autenticados do sistema são implementados através do uso de *sockets* TCP e chaves de sessão baseadas no algoritmo *HmacSHA-1*.

Os *gateways* foram concretizados sobre o *Axis* 1.3 (`http://ws.apache.org/axis/`), uma implementação livremente disponível do protocolo SOAP. Estes *gateways* foram instalados no container J2EE *Tomcat* 5.5.20 (`http://tomcat.apache.org/`).

Outro ponto a destacar é que todos os mecanismos e processamentos adicionais relacionados ao WSDS (descritos na seção 4.3) são integrados aos servidores DEPSPACE através de interceptadores, o que permite a mudança no comportamento dos servidores sem alterações em sua estrutura. A implantação deste suporte foi inspirada nos interceptadores portáveis previstos na arquitetura CORBA [Object Management Group 2002]. Os dados adicionais, necessários na execução de uma requisição (ex. dados relacionados com assinaturas), são enviados através de uma abstração (contexto) que acompanha as mensagens.

5. Análise de Desempenho

Nesta seção é apresentada uma análise preliminar acerca do desempenho do WSDS, em particular da latência envolvida na execução de uma operação deste serviço, a qual é determinada pela equação:

$$L_{wsds} = L^{c}_{sign} + L^{c \to g}_{comm} + L^{g \to s}_{tom} + L^{s}_{ver} + L^{s}_{op} + L^{s}_{sign} + L^{s \to g}_{comm} + L^{g \to c}_{comm} + (f+1)L^{c}_{ver} \quad (1)$$

As latências envolvidas nesta equação são: L^{c}_{sign} - assinatura da requisição; $L^{c \to g}_{comm}$ - envio da requisição ao *gateway*; $L^{g \to s}_{tom}$ - envio da requisição aos servidores e sua ordenação; L^{s}_{ver} - verificação da integridade da requisição; L^{s}_{op} - execução da operação; L^{s}_{sign} - assinatura da resposta; $L^{s \to g}_{comm}$ - envio da resposta ao *gateway*; $L^{g \to c}_{comm}$ - envio das respostas ao cliente, note que $L^{c \leftrightarrow g}_{comm} = L^{c \to g}_{comm} + L^{g \to c}_{comm}$ representa a comunicação entre o cliente e o *gateway*; L^{c}_{ver} - verificação da integridade da resposta.

Assumindo que as operações de assinatura e verificação têm latências semelhantes no cliente e nos servidores e que o custo do processamento local das operações no espaço de tuplas é desprezível ($L^{s}_{op} = 0$), chegamos a latência esperada para os serviços oferecidos pelo WSDS:

$$L_{wsds} = 2L_{sign} + L^{c \leftrightarrow g}_{comm} + L_{tom} + L^{s \to g}_{comm} + (f+2)L_{ver} \quad (2)$$

Dentro destas mesmas premissas, a latência esperada para o DEPSPACE é $L_{ds} = L_{tom} + L^{s \to g}_{comm}$. Conseqüentemente, o custo adicional do WSDS em relação ao DEPSPACE consiste nas assinaturas da requisição e da resposta, na comunicação extra para acesso ao *gateway* e na verificação da autenticidade da requisição e das $f + 1$ respostas.

Visando analisar se essa latência é observada na prática, alguns experimentos foram realizados num ambiente de rede local composto por 4 máquinas com a mesma configuração de *hardware* (AMD Athlon XP 1.9Ghz, 512MB de RAM, placa *ethernet* de 100MB/s) conectadas por um *switch* 100 Mbps. O ambiente de *software* em todas as máquinas é também homogêneo: S.O. GNU/Linux *kernel* 2.6.12, máquina virtual Java da SUN versão 1.5.0_06.

Os experimentos foram realizados com 4 servidores DEPSPACE (um em cada máquina), um *gateway* e um cliente que foram executados em máquinas diferentes, porém ambas contendo um servidor DEPSPACE. Os resultados destes experimentos são apresentados nesta seção. Todos os valores reportados aqui compreendem o tempo médio necessário para a execução de uma operação por um cliente do sistema, recolhido a partir de 1000 execuções da operação e excluindo-se os 5% dos valores com maior desvio.

A figura 2 apresenta a latência média para a execução das quatro operações, variando-se o tamanho das tuplas. Nesta figura, podemos perceber que a latência apresentou um crescimento suave com o aumento do tamanho das tuplas, o que indica que o sistema é escalável em relação a este fator. Além disso, pode-se notar que os desvios apresentados ($2 - 6ms$) são apropriados, levando em consideração o tipo de serviço oferecido.

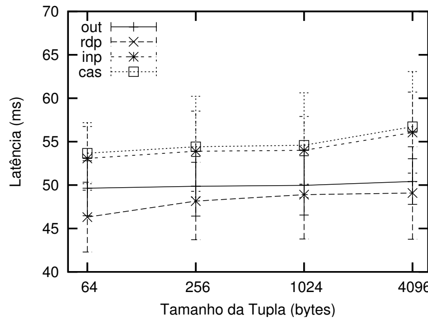


Figura 2. Latência do WSDS.

Latência	Custo (ms)	$\%L_{wsds}$
$L_{comm}^{s\rightarrow g}$	0.63	1.27
$L_{comm}^{c\leftrightarrow g}$	13.53	$27,27$
L_{sign}	13.51	54.45
L_{ver}	0.85	$5,13$
L_{tom}	5.90	11.88
L_{wsds}	49.63	100
L_{ds}	6.53	13.15

Tabela 1. Custos da latência.

A tabela 1 apresenta os custos relacionados com cada fase do protocolo (para uma tupla de 64 *bytes*), juntamente com sua percentagem correspondente na latência total observada pelo cliente, de acordo com a equação 2. Nesta tabela, podemos observar que as comunicações entre o cliente e o *gateway* (SOAP) é a fase que consome mais tempo, seguido pelas assinaturas (em redes de larga escala, a tendência é que esta fase tenha uma participação menor na latência).

6. Aplicações e Relações com Especificações para Serviços *Web*

Esta seção apresenta alguns exemplos de uso do WSDS e discute como este tipo de serviço de coordenação se relaciona com diversos padrões de interesse desenvolvidos pela comunidade de serviços *web*. O objetivo desta seção não é apenas mostrar que o WSDS pode ser usado para resolver problemas tão dispares de forma tão boa quanto sistemas desenvolvidos exclusivamente para solucionar tais problemas, mas também mostrar que o mesmo é genérico suficiente para ser utilizado em um grande número de aplicações.

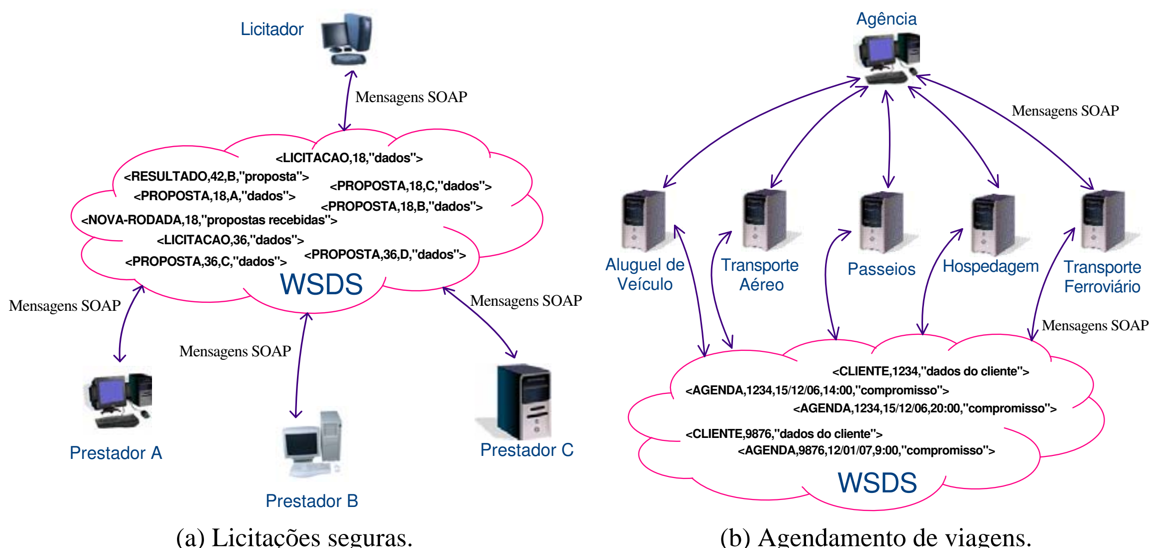
6.1. Licitações Seguras

Um tipo de aplicação que pode ser facilmente implementada usando o WSDS é um sistema de gerenciamento de licitações. Utilizando o WSDS, um licitador interessado em determinado serviço escreve uma tupla no espaço descrevendo as características do serviço a ser contratado (ou produto a ser comprado). A partir daí, os prestadores de serviço interessados (que podem ter sido previamente cadastrados no WSDS) também inserem uma tupla no WSDS descrevendo sua proposta para prestação do serviço. Por fim, depois da data limite para se efetuar as propostas, o cliente solicitante lê todas as tuplas com propostas relacionadas ao seu serviço e faz uma de três coisas: *(i.)* escolhe o fornecedor com a melhor proposta, pondo fim a licitação; *(ii.)* não escolhe nenhuma proposta pois decide que todas são inadequadas no que diz respeito à qualidade e ao preço estipulado na descrição da licitação; ou *(iii.)* faz um resumo das propostas obtidas e as publica no espaço, dando início a uma outra rodada de licitações (em uma espécie de leilão, onde os prestadores poderiam fazer novas propostas melhorando seu preço e ou ofertando melhorias no serviço). A figura 3(a) ilustra essa aplicação e os diversos tipos de tuplas que podem aparecer no WSDS.

Neste sistema existem vários requisitos de segurança: *(i.)* um prestador de serviço não deve ter acesso a proposta de outro prestador; *(ii.)* um prestador não pode realizar mais de uma proposta; *(iii.)* se um cadastramento prévio for exigido, um fornecedor não cadastrado não deve ser capaz de fazer propostas; entre outros específicos de cada licitação. Usando as capacidades do WSDS, em particular seus mecanismos de controle de acesso, é possível garantir que todos

esses requisitos sejam respeitados: o requisito *(i.)* pode ser implementado através da inclusão de tuplas contendo propostas que requerem credenciais de licitador para leitura e os requisitos *(ii.)* e *(iii.)* podem ser implementados através de políticas de granularidade fina que estipulam regras do tipo *"se houver uma proposta do prestador A para licitação X no espaço, não é permitida a inclusão de uma nova proposta de A para X"* e *"o prestador A só pode inserir uma proposta se houver uma tupla do tipo ⟨PRESTADOR, A⟩ no espaço"*[6], respectivamente[7].

Outros exemplos onde o espaço de tuplas é usado como mecanismo de comunicação desacoplado: o sistema I3 [Stoica et al. 2004] oferece *multicast*, *anycast* e suporte a comunicação móvel na Internet através de uma abstração bastante semelhante a um espaço de tuplas; o padrão *master-worker* (onde um processo distribui tarefas a outros e depois agrupa os resultados), muito usado para distribuição de tarefas no processamento em *clusters*, pode facilmente ser implementado usando um espaço de tuplas [Carriero and Gelernter 1989] e, uma vez que este espaço esteja disponível na Internet, este mesmo modelo de programação pode ser usado na distribuição de tarefas em um *grid* computacional [Favarim et al. 2006]. A alta disponibilidade e as políticas suportadas pelo WSDS podem garantir que estes serviços de comunicação e distribuição mantenham suas propriedades mesmo na presença de faltas e intrusões. Além disso, a característica de desacoplado da coordenação por espaço de tuplas permite que as partes comunicantes interajam mesmo sem conhecer o endereço uma das outras e sem estar ativas ao mesmo tempo (ex. devido a desconexões temporárias).



(a) Licitações seguras. (b) Agendamento de viagens.

Figura 3. Exemplos de aplicações usando o WSDS.

6.2. Compartilhamento de Dados entre Serviços Cooperantes

Um requisito recorrente em aplicações que envolvem a invocação de diversos serviços *web* para execução de uma determinada tarefa é o compartilhamento de informações entre estes serviços. Esse tipo de aplicação faz uso de uma *base de dados compartilhada* (cujos diversos serviços têm acesso) ou requer que todas as *informações de sessão* da tarefa (pertinentes aos diversos serviços) sejam anexadas a todas as mensagens trocadas durante a execução da tarefa. Quando consideramos que os serviços *web* não estão instalados no mesmo domínio administrativo, a

[6]Esta regra deve ser complementada com outra que diz que esse tipo de tupla só pode ser inserida por um administrador.

[7]Lembrar que o DEPSPACE suporta confidencialidade de tuplas, sendo que os requisitos de confidencialidade podem ser mantidos mesmo que até f_r servidores sejam maliciosos [Bessani et al. 2006a, Bessani et al. 2007]. Esta funcionalidade foi integrada ao WSDS, porém não é discutida por razões de espaço.

existência de uma base de dados aberta para acesso direto a partir da Internet pode incorrer em sérios problemas de segurança e torna os serviços cooperantes dependentes de um ponto único de falha. Já a passagem de informações de sessão em todas as mensagens pode exigir que um volume não desprezível de informações tenha que ser enviado de um serviço para outro, o que pode prejudicar muito o desempenho do sistema. Se um serviço como o WSDS estiver disponível, os serviços cooperantes podem armazenar todas as informações de sessão em um espaço de tuplas compartilhado. As características do WSDS garantem *(i.)* a confiabilidade deste repositório (replicação tolerante a faltas bizantinas), *(ii.)* controle de acesso ao espaço e as tuplas armazenadas e *(iii.)* acesso universal (através dos *gateways*, que são serviços *web*).

Como exemplo deste tipo sistema, considere um conjunto de serviços *web* usados por uma agência de turismo para alocar os recursos requeridos em uma viagem de férias (ex. transporte aéreo, hospedagem, aluguel de carro, agendamento de passeios turísticos, dentre outros). Para que todo esse processamento possa ser feito de forma automatizada, as informações sobre o viajante, suas preferências e sua agenda devem estar disponíveis a todos estes serviços. Além disso, os serviços podem alterar esses dados (ex. agenda) conforme executam sua parte do planejamento e agendamento da viagem. A figura 3(b) ilustra esse sistema.

Este exemplo é extremante interessante no sentido de necessitar também das capacidades de sincronização do espaço: as tuplas da agenda podem ser usadas de tal forma que os problemas decorrentes de diferentes serviços agendando compromissos concorrentemente, para um mesmo horário, possam ser resolvidos através de operações com poder de sincronização providas pelo espaço, como cas e inp.

A política de segurança poderia definir que tipo de informação (tupla) pode ser acessada por cada tipo de serviço e que apenas a agência de viagem pode remover um compromisso já agendado ou cancelar o aluguel de um recurso (hospedagem, carro) já alugado. Além disso, as tuplas com os dados do cliente devem ser mantidas tão confidenciais quanto possível, visando manter sua privacidade.

Neste exemplo, a agência poderia contratar cada serviço para o viajante através do mecanismo de licitações seguras apresentado na seção anterior. Os dois modelos de programação podem ser integrados no mesmo espaço de tuplas ou usando diferentes espaços lógicos para cada licitação e para a agenda do cliente.

Outros exemplos de serviços desse tipo já explorados são: integração de sistemas de detecção de intrusões [Yegneswaran et al. 2004] – onde resumos de comportamentos observados por diferentes sistemas são compartilhados no espaço de tuplas para permitir a correlação de atividades suspeitas; e o suporte a empresas virtuais [Ricci et al. 2001, Bright and Quirchmayr 2004] – onde um espaço de dados compartilhado é usado entre os diversos parceiros que formam a empresa virtual, visando compartilhar informações.

6.3. Especificações WS-*

Existe uma série de especificações desenvolvidas (ou em desenvolvimento) pela comunidade para prover integração de serviços *web*. Nesta seção discutimos a relação do WSDS com algumas destas especificações.

WS-Coordination Este padrão define um arcabouço genérico através do qual diversos serviços *web* podem se coordenar para realização de uma determinada tarefa [Cabrera L. F. et al. 2005]. O ponto central deste arcabouço é o contexto de coordenação, uma abstração que contém todas as informações sobre a coordenação (ex. participantes). Atualmente este arcabouço já foi instanciado para implementação de transações distribuídas envolvendo vários serviços *web*. A abstração de espaço de tuplas, provida pelo WSDS, pode ser implementada como outra instanciação do WS-COORDINATION onde o contexto de coordenação é

o espaço de tuplas lógico e os serviços registrados no contexto podem ser entendidos como os clientes que podem interagir com o espaço.

WS-Orchestration A orquestração de serviços *web* compreende o uso de diversos serviços coordenados por um "maestro", chamado motor de orquestração. A idéia básica é definir quais serviços *web* serão invocados e em que ordem, usando uma linguagem de coordenação, como a WSBPEL (*Web Services Business Process Execution Language*) [Alves A. et al. 2006]. A agência de viagens descrita na seção anterior é um exemplo de tarefa que poderia ser implementada com orquestração. Até o momento não existe um padrão de repositório de dados compartilhados para orquestração de serviços *web*, e toda informação requerida pelos serviços deve ser mantida pelo motor de orquestração e enviada a cada serviço como parâmetro da operação requisitada, podendo ser bastante ineficiente se o volume de dados for grande. Um repositório de dados seguro e confiável pode ser útil nesse sentido, e sua integração com a linguagem WSBPEL pode ser facilmente feita, bastando que os serviços sendo orquestrados usem o WSDS.

WS-Choreography A coreografia de serviços *web* permite a definição formal de um conjunto de interações entre serviços *web*, a verificação de sua corretude (ex. provando a ausência de *deadlocks*) e a geração do código responsável pelas interações [Burdett and Kavantzas 2004]. A coreografia difere da orquestração em pelo menos dois aspectos: é distribuída (enquanto a orquestração tem um coordenador – o motor de orquestração); e as linguagens de coreografia (como o WSCL – *Web Services Choreography Language*) são usadas para *especificação* de interações requeridas durante a realização de tarefas, não sendo *executáveis* (diferindo de linguagens para orquestração, como o WSBPEL) [Peltz 2003]. A introdução da coreografia de serviços *web* visa tornar essas interações menos suscetíveis a erros, através de uma especificação mais rigorosa das relações entre os serviços. Porém, esta tecnologia não oferece nenhum tipo de garantia que as interações vão ocorrer como especificado em tempo de execução. O uso de um mediador como o WSDS preenche esta lacuna. As interações poderiam ser feitas através de tuplas inseridas e lidas do espaço de tuplas. Políticas podem ser geradas, a partir da coreografia especificada, para garantir que o sistema funciona da forma prevista mesmo na presença de falhas e intrusões. Além disso, o uso do WSDS permite a implementação de interações multi-parte e a gravação (e posterior auditoria) de todas as interações executadas pelo serviços.

7. Trabalhos Relacionados

Atualmente existe um grande esforço da comunidade de sistemas distribuídos no desenvolvimento de mecanismos padronizados que permitam a cooperação e a integração de serviços *web*. Iniciativas como WS-COORDINATION [Graham S. at al. 2004], WS-ORCHESTRATION [Alves A. et al. 2006] e WS-CHOREOGRAPHY [Burdett and Kavantzas 2004] vão exatamente nessa direção (ver seção 6.3). Estas iniciativas consideram basicamente a integração dos serviços *web* através de trocas de mensagens, e não definem nenhum tipo de abstração que suporte o armazenamento de dados relacionados a uma tarefa cooperativa sendo executada pelos serviços. Esse tipo de abstração é crucial em aplicações onde o volume de dados compartilhado é grande ou onde desacoplamento é necessário. O WSDS visa fornecer essa abstração sem no entanto esquecer de aspectos de segurança de funcionamento. Mais especificamente, o serviço de coordenação por ele suportado é seguro e tolerante a falhas e intrusões.

A integração do modelo de coordenação por espaço de tuplas no ambiente dos serviços *web* tem se tornado um tema de pesquisa ativo nos últimos anos (ex. [Bright and Quirchmayr 2004, Lucchi and Zavattaro 2004, Maamar et al. 2005, Bellur and Bondre 2006]). Esses trabalhos enaltecem as facilidades que um repositório de dados compartilhado, com interfaces bem definidas e alguma capacidade de sincronização, pode proporcionar na colaboração de serviços *web* [Maamar et al. 2005] e na execução de *workflows*

[Bright and Quirchmayr 2004, Bellur and Bondre 2006]. Um aspecto importante que escapa a quase todos esses trabalhos é a segurança de funcionamento [Avizienis et al. 2004] requerida nesse serviço. A segurança de funcionamento deve ser provida em dois níveis: *(1)* mecanismos de segurança para controle de acesso ao espaço de tuplas e *(2)* disponibilidade do serviço de coordenação em si. A maioria dos trabalhos na área não implementa nenhum desses níveis, focando-se na integração do modelo de coordenação por espaço de tuplas com os padrões para serviços *web* [Bright and Quirchmayr 2004, Maamar et al. 2005, Bellur and Bondre 2006]. Uma notável excessão é o WS-SECSPACES [Lucchi and Zavattaro 2004], que se preocupa com o nível *(1)*, provendo um modelo de controle de acesso em nível de espaço e tuplas, permitindo a especificação de quem pode inserir, ler e remover determinada(s) tupla(s). O WSDS, apresentado neste artigo, suporta um modelo de controle de acesso semelhante ao provido pelo WS-SECSPACES além de suportar a definição de políticas de segurança de granularidade fina, que dão poder de coordenação ao espaço de tuplas mesmo na presença de processos maliciosos [Bessani et al. 2006b]. Além disso, o WSDS implementa mecanismos para suprir o nível *(2)* de segurança de funcionamento, através de uma arquitetura simples e eficiente que faz uso de *gateways* para acesso a um serviço de coordenação confiável – DEPSPACE [Bessani et al. 2007] – implementado através de replicação máquina de estado tolerante a faltas bizantinas [Castro and Liskov 2002]. O sistema resultante tolera faltas acidentais e maliciosas em todos os componentes do sistema.Todas essas características são providas sem ferir a adesão aos padrões para serviços *web*.

8. Conclusões e Trabalhos Futuros

Este trabalho apresentou o WSDS, um serviço de coordenação para serviços *web* que provê elevadas garantias em termos de segurança de funcionamento, como controle de acesso através de políticas de granularidade fina e tolerância a faltas e intrusões. A arquitetura do sistema se baseia no uso de um espaço de tuplas com segurança de funcionamento e em serviços *web* sem estado (*gateways*) para acesso a este espaço. Diversos mecanismos foram empregados para garantir a segurança do sistema mesmo na presença de *gateways* maliciosos.

O sistema foi implementado usando ferramentas de código aberto e uma análise preliminar de seu desempenho foi apresentada, mostrando que grande parte dos custos em termos da latência do serviço vêm do processamento das assinaturas digitais requeridas, o que deve ser diluído quando da instalação do sistema em uma rede de larga escala como a Internet.

Os próximos passos deste trabalho consistem em instalar um serviço WSDS no PLANETLAB (`http://www.planet-lab.org/`) e desenvolver uma aplicação completa que explore as potencialidades deste serviço.

Referências

Alves A. et al. (2006). Web services business process execution language, version 2.0. OASIS Public Draft. Disponível em `http://docs.oasis-open.org/wsbpel/2.0/`.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bellur, U. and Bondre, S. (2006). xSpace: a tuple space for XML and its application in orchestration of web services. In *Proceedings of the 2006 ACM symposium on Applied computing – SAC 2006*, pages 766–772.

Bessani, A. N., Alchieri, E. A. P., Correia, M., Fraga, J. S., and Lung, L. C. (2006a). Provendo confidencialidade em espaços de tuplas tolerantes a intrusões. In *Anais do 6o Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2006*.

Bessani, A. N., Alchieri, E. A. P., da Silva Fraga, J., and Lung, L. C. (2007). Design and implementation of an intrusion-tolerant tuple space. In *Proceedings of the International Workshop on Recent Advances on Intrusion-Tolerant Systems (with EuroSys 2007)*.

Bessani, A. N., Correia, M., Fraga, J. S., and Lung, L. C. (2006b). Sharing memory between Byzantine processes using policy-enforced tuple spaces. In *Proceedings of 26th IEEE International Conference on Distributed Computing Systems - ICDCS 2006*.

Bichier, M. and Lin, K.-J. (2006). Service-oriented computing. *IEEE Computer*, 39(3):99–101.

Bright, D. and Quirchmayr, G. (2004). Supporting web-based collaboration between virtual enterprise partners. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications – DEXA 2004*.

Burdett, D. and Kavantzas, N. (2004). The WS-Choreography model overview. W3C Draft. Disponível em `http://www.w3.org/TR/ws-chor-model/`.

Cabrera L. F. et al. (2005). Web Services Coordination Specification - version 1.0. Disponível em `http://www-128.ibm.com/developerworks/library/specification/ws-tx/`.

Carriero, N. and Gelernter, D. (1989). How to write parallel programs: a guide to the perplexed. *ACM Computing Surveys*, 21(3):323–357.

Castro, M. and Liskov, B. (2002). Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461.

Dierks, T. and Allen, C. (1999). The TLS Protocol Version 1.0 (RFC 2246). IETF Request For Comments.

Dwork, C., Lynch, N. A., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–322.

Favarim, F., Correia, M. P., Lung, L. C., and Fraga, J. (2006). Fault-tolerant multiuser computational grids based on tuple spaces. In *Proceedings of the International Workshop on Dependability in Service-oriented Grids (with SRDS 2006)*.

Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programing Languages and Systems*, 7(1):80–112.

Graham S. at al. (2004). WS-Base Notification. Disponível em http://ifr.sap.com/ws-notification/WS-BaseNotification.pdf.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programing Languages and Systems*, 4(3):382–401.

Lucchi, R. and Zavattaro, G. (2004). WSSecSpaces: a secure data-driven coordination service for web services applications. In *Proceedings of the 19th ACM Symposium on Applied Computing - SAC 2004*, pages 487–491.

Maamar, Z., Benslimane, D., Ghedira, C., Mahmoud, Q. H., and Yahyaoui, H. (2005). Tuple spaces for self-coordination of web services. In *Proceedings of the 2005 ACM symposium on Applied computing – SAC 2005*, pages 1656–1660.

Obelheiro, R. R., Bessani, A. N., and Lung, L. C. (2005). Analisando a viabilidade da implementação prática de sistemas tolerantes a intrusões. In *Anais do V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2005*.

Object Management Group (2002). The common object request broker architecture: Core specification v3.0. OMG Standart formal/02-12-06.

Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52.

Ricci, A., Omicini, A., and Denti, E. (2001). The tucson coordination infrastructure for virtual enterprises. In *Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 348–353.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Schneider, F. B. (1990). Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Segall, E. J. (1995). Resilient distributed objects: Basic results and applications to shared spaces. In *Proceedings of the 7th Symposium on Parallel and Distributed Processing - SPDP'95*, pages 320–327.

Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. (2004). Internet indirection infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218.

Yegneswaran, V., Barford, P., and Jha, S. (2004). Global intrusion detection in the DOMINO overlay system. In *Proceedings of Network and Distributed Security Symposium – NDSS 2004*.