inSOA – Uma Linguagem de Composição de Serviços para Dispositivos Móveis

Giovane Barcelos[1], Alexsandro Filippetto[1], Luciano Zanuz[1],
Sérgio Crespo C S Pinto[1]

[1] Programa de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 – São Leopoldo – RS – Brasil
{giovanebarcelos, alexsandrofilippetto, lucianozanuz}@gmail.com, crespo@unisinos.br

Abstract. *Services composition on mobile devices is becoming essential for current and future needs of IT users. These needs are supported by the convergence from centralized to distributed architectures that encourage the web services composition. These web services, motivated by the Web Services architecture, quickly evolved into SOA. This architecture in turn, in the services orchestration, has demanded languages of composition for their effective implementation. Which have progressed to interpreted and declarative languages. In this evolution context a declarative language for service composition called inSOA is presented.*

Keywords: *SOA*; Services Composition Language, Mobile Computing, *DSL*.

1 Introdução

A *Web* foi criada em 1989 nos laboratórios do *CERN*[1] (Conselho Europeu para Pesquisa Nuclear) com o objetivo de facilitar a colaboração dos pesquisadores de seus laboratórios no desenvolvimento de pesquisas e projetos [1]. A implementação inicial deu-se pelo compartilhamento de documentos de pesquisa e desenvolvimento de software em hipertexto utilizando um navegador numa arquitetura em rede. Desde então o uso da *web* tem crescido muito e estimulado novas tecnologias. Entre estas novas tecnologias podemos destacar *Web Services* (*WS*), que permitiu o uso de aplicações por meio de serviços disponíveis na *web*, colaborando para descentralização e conseqüente distribuição de aplicações em diferentes plataformas.

A arquitetura *WS* rapidamente evoluiu para *SOA* (*Service Oriented Architecture*) que permitiu construir aplicações mais dinâmicas com composição de aplicações em tempo de execução [2]. A composição e orquestração de serviços tornaram os benefícios da arquitetura *SOA* [3; 4] mais visíveis, pois possibilitaram a reutilização, colaboração, interoperação e produtividade no desenvolvimento de *software*. Assim, *SOA* se mostrou uma opção para a utilização de serviços e colaboração entre diversos

1 *CERN* - http://public.web.cern.ch/public/

usuários.

Com o advento do *SOA* e a necessidade de produtividade no desenvolvimento de composições de aplicações e serviços, nasceram as linguagens de composição de serviços. Dentre estas linguagens de composição a mais influente é a *BPEL* (*Business Process Execution Language*) [2; 5], que implementa composição e orquestração de serviços a partir de um modelo baseado em *XML*.

Recentemente uma série de trabalhos [7; 8; 5; 9] estão migrando as tecnologias de serviços (*XML*, *WS*, *SOA*, *BPEL*, etc.) do *desktop* para dispositivos móveis, iniciando assim, um processo de convergência tecnológica. Isto se deve a demanda crescente de aplicações com mobilidade, que possibilitem a comunicação e integração com aplicações legadas, e também ao apelo da computação ubíqua. O grande desafio deste ambiente computacional é o desenvolvimento de soluções que sejam do tamanho das limitações de recursos impostas por estes dispositivos. Desenvolver linguagens de composição para dispositivos móveis, apesar de desafiante, é necessário, considerando que estes dispositivos dependem muito de integração e uso de serviços disponíveis externamente.

Este artigo apresenta uma linguagem de composição de serviços para dispositivos móveis chamada de *inSOA* (*invoke SOA*). Esta linguagem faz parte de um projeto de arquitetura ubíqua orientada a serviços chamada de *U-SOA* (*Ubiquitous SOA framework*) [6], que possui também entre seus componentes um motor de execução de composição de serviços, chamado de *SmallSOA*, e um analisador de impacto de composições de serviços, chamado de *gImpact*.

Inicialmente, na seção 2, será realizada uma breve revisão bibliográfica dos conceitos envolvidos neste trabalho. Na seção 3, será apresentada a linguagem *inSOA*, sua arquitetura, características, gramática e o seu protótipo. O estudo de caso e os resultados obtidos nos experimentos são apresentados na seção 4. Finalmente os comentários finais e a conclusão encontram-se na seção 5.

2 Revisão Bibliográfica

2.1 *Web Services* e *SOA*

Web Services é uma arquitetura que fornece uma forma padronizada de interoperação entre diferentes aplicações, rodando sobre uma variedade de plataformas e/ou *frameworks*. [10]

A arquitetura *WS* foi concebida como uma solução padronizada para utilização na integração de sistemas e na comunicação entre aplicações distintas. A comunicação destas aplicações dá-se por troca de mensagens *XML*, tendo como principais componentes os protocolos *WSDL*, *UDDI*, *HTTP* e *SOAP*.

SOA significa *Service Oriented Architecture*. É um estilo de arquitetura que tenta alinhar os processos de negócios com TI [12], ou seja, é mais um modelo de implementação do que uma tecnologia propriamente dita. É uma abordagem de desenvolvimento de software na qual seus serviços são construídos como componentes reutilizáveis que visam fornecer um baixo acoplamento e interoperabilidade [13]. Este modelo define que todas as funcionalidades das

aplicações devem ser disponibilizadas em forma de serviços. Sendo assim, o modelo *SOA* utiliza-se fortemente dos conceitos de *WS* e estende estes ao propor a padronização e melhores práticas na construção e uso de serviços.

2.3 Linguagens de Composição de Serviços *Web*

A composição de serviços web é caracterizada pela execução de dois ou mais serviços com objetivo de atender um processo de negócio [14]. Segundo Aalst at al. [15], a ideia de composição de serviços por orquestração é tão antiga quanto as primeiras iniciativas de *workflow* ocorridas nas décadas de 80 e 90, as quais devem ser consideradas no estudo de composição de serviços. Portanto, para se avaliar e desenvolver novas linguagens de composição de serviços é razoável que os *design patterns* adquiridos até então com orquestração sejam considerados.

Design patterns ou padrões de projetos descrevem um problema que ocorre repetidas vezes em nosso ambiente [16]. Como referencial para uma efetiva comparação é importante sempre termos um conjunto de *patterns* esperados em um domínio de problema que possam ser comparados e consequentemente medidos. Segundo Will at al. [17], as linguagens de modelagem de processos de negócios e composição de serviços existem há bastante tempo e tem seus *patterns* já bem definidos, sendo estes compostos de 21 *patterns*. Este *patterns* têm evoluído para atender as composições de serviços na *web* com *WS* numa arquitetura *SOA*.

Em resposta a este novo paradigma de composição de serviços *web*, uma grande quantidade de linguagens e técnicas de composição de serviços *web* emergiu e está continuamente evoluindo com novas propostas e diferentes soluções. Entre as linguagens de composição para *web* destaca-se a *BPEL4WS* [2; 5] que se tornou um padrão de fato no mercado e a *OWL-S* [10] devido ao seu foco em ontologias.

3. *inSOA* (*invoke SOA*)

Nesta seção é apresentado o *framework U-SOA* e a linguagem *inSOA*. Durante estas apresentações serão discutidas as características, a estrutura, a gramática e o protótipo da linguagem, bem como, será demonstrado um exemplo e o diagrama da sintaxe da *inSOA*.

3.1 *U-SOA*

U-SOA (*Ubiquitous SOA framework*) é uma pilha de tecnologias separadas em camadas, conforme ilustrado na figura 1. Estas camadas contemplam desde níveis mais baixos, no qual se encontra o *Android*[2] da *Google*, até o topo da pilha, onde se encontram as aplicações colaborativas ubíquas. O objetivo do *U-SOA* é disponibilizar uma meta-arquitetura que suporte ambientes colaborativos em dispositivos móveis.

---

2 *Android - http://code.google.com/intl/pt-BR/android/*

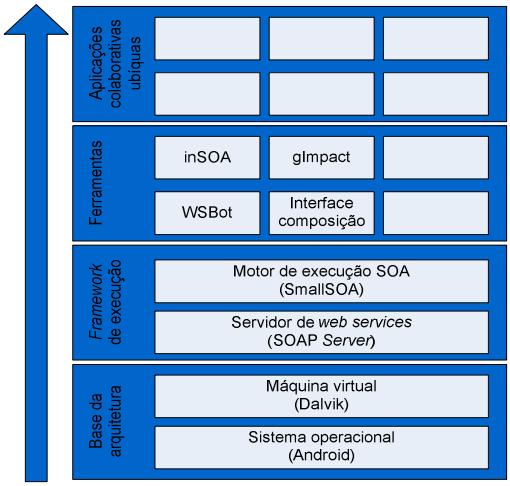

Fig. 1. Componentes de *U-SOA*.

Atualmente, os seguintes trabalhos já foram ou estão sendo desenvolvidos: servidor de *WS SOAP Server*; motor de execução *SmallSOA*; linguagem de composição de serviços *inSOA*; ferramenta para análise de impacto na composição de serviços *gImpact*; robô para descobrimento de serviços *WSBot* e interface para composição de serviços em dispositivos móveis.

3.2 Linguagem de Composição *inSOA*

inSOA (*invoke SOA*) é uma linguagem declarativa de composição de serviços com foco em dispositivos móveis. Ela faz composição de serviços utilizando a arquitetura *SOA* com o protocolo *SOAP*. O editor e compilador da linguagem *inSOA* roda inteiramente no emulador *Android* e a execução da composição é realizada no motor de execução *SmallSOA*. O motor executa a composição a partir de um arquivo *XML* otimizado resultante da compilação do script *inSOA*.

3.2.1 Características

A linguagem *inSOA* possui uma série de características que a tornam uma linguagem de composição de serviços bastante flexível. Estas características são: insensível a capitalização, declaração flexível, validação dos padrões *URI RFC 3986* e *XPath 2.0* [11], tratamento *XML* com *XPath*, opcionalidade dos comandos, tratamento de erros, orquestração, saída customizável com *XML* otimizado via *StringTemplate*, invocação otimizada, embutível em linguagens de propósito geral, leve e sem autoria, abordagem declarativa *DSL* (*Domain Specific Language*) e baseada em texto

Tabela 1: Comparação das características das linguagens de composição.

#	Característica	*BPEL*	*OWL-S*	*inSOA*
a	**Insensível a capitalização**	NÃO	NÃO	SIM
b	**Declaração Flexível**	NÃO	NÃO	SIM
c	**Validação de Padrões**	NÃO	NÃO	SIM
d	**Tratamento *XML***	Externo	Externo	Interno
e	**Opcionalidade**	NÃO	NÃO	SIM
f	**Tratamento de Erros**	SIM	SIM	SIM
g	**Orquestração**	SIM	SIM	SIM
h	**Saída Customizável**	NÃO	NÃO	SIM
i	**Invocação Otimizada**	NÃO	NÃO	SIM
j	**Embutida**	NÃO	NÃO	SIM
k	**Leve**	NÃO	NÃO	SIM
l	**Abordagem**	Funcional	OOP	DSL
m	**Baseada**	XML	XML	Texto

A tabela 1 apresenta um comparativo das características dos dois principais paradigmas de linguagens de composição e a linguagem *inSOA*.

3.2.2 Estrutura e Gramática

A linguagem *inSOA* é representada por um código em formato de *script*. Este *script* está dividido essencialmente em um comando obrigatório e outros oito opcionais. O comando obrigatório é o *invoke*, enquanto os outros oito são: *input*, *into*, *set*, *where*, *return*, *fail*, *id* e *tags*. O compilador *inSOA* foi desenvolvido utilizando a ferramenta *ANTLRWorks*[3] para gerar o *parser* e o *StringTemplate*[4] como *framework* de geração de saída customizável.

Na figura 2 é apresentado o diagrama da sintaxe da linguagem *inSOA*. Nas elipses encontram-se os elementos léxicos ou comandos da linguagem, enquanto nos retângulos são demonstrados os elementos de *parser* da *inSOA*. As setas pontilhadas do diagrama representam a opcionalidade entre os elementos e comandos da linguagem. Como pode-se perceber alguns comandos, tais como, o *invoke*, após o caractere vírgula, podem ser opcionalmente recursivos.

3 *AntlrWorks - http://www.antlr.org/works/index.html*
4 *StringTemplate - http://www.stringtemplate.org/*

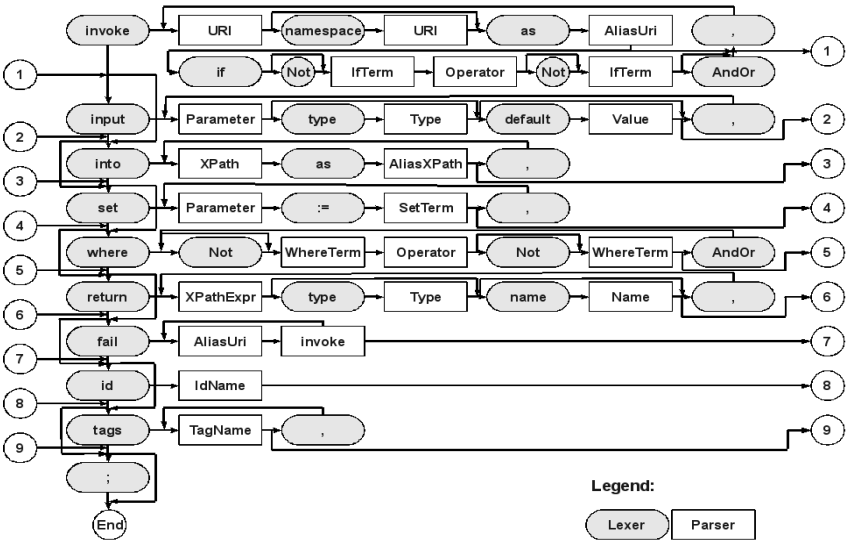

Fig. 2. Diagrama da sintaxe da linguagem *inSOA*.

Um exemplo típico de composição de serviços utilizando a linguagem *inSOA* na implementação do *pattern parallel split* seria como demonstrado na figura 3. Neste exemplo o *WS1* e o *WS3* serão executados em paralelo por não possuírem dependência entre si, enquanto o *WS2* será executado após o *WS1* por possuir dependência com este. Caso ocorra algum erro no *WS2* será enviado um *e-mail* como descrito na cláusula *fail* da figura.

```
invoke http://usoa.insoa/WS1.Operation as a, http://usoa.insoa/WS2.Operation as b,
          http://usoa.insoa/WS3.Operation as c
   into / as ResA, / as ResB, / as ResC
     set a.field := 'Test',
         b.field := ResA/result/text(),
         c.field := 'Test1'
   fail b: invoke http://usoa.insoa/WS4.SendMail as d;
```

Fig. 3. Exemplo da linguagem *inSOA* implementando o *pattern parallel split*.

3.2.3 Protótipo do Compilador *inSOA*

Como parte do processo de avaliação da linguagem foi desenvolvido um ambiente de edição, compilação e execução da *inSOA* ilustrado na figura 4. O protótipo do compilador foi desenvolvimento para a plataforma *Android* com uma interface única.

Na primeira imagem da figura 4 encontra-se a tela principal do compilador que é dividida em três áreas: edição do script *inSOA*, barra de ferramentas e resultado. A área de edição possibilita a edição de um script *inSOA* com tamanho de até 64k. O espaço inferior é reservado para mostrar as mensagens da compilação e o resultado da execução da composição. A barra de ferramentas possui cinco botões com

funcionalidades distintas como descrito abaixo:

- *execute*: este botão compila, gera o *XML* otimizado, publica e executa a composição;
- *cancel*: o *cancel* emite uma sinalização de cancelamento da execução de uma composição em curso.
- *open*: este *botão* tem por objetivo abrir uma composição salva no dispositivo móvel, conforme a segunda imagem da figura 4.
- *save*: possibilita salvar uma composição editada no editor de composições da ferramenta, como ilustrado na terceira imagem da figura 4.
- *create XML*: quando esse botão encontra-se habilitado significa que a operação de execução da *composição* deve gerar o arquivo *XML* de integração com o motor de composição *SmallSOA*.

As principais funcionalidades dessa ferramenta são: a edição e a execução dos scripts *inSOA*. A execução dá-se via motor de composição após gerado o *XML* de integração. O mecanismo de geração do *XML* é customizável, visto que, o compilador gera uma árvore de objetos em memória que pode ser traduzido pelo *framework StringTemplate* para qualquer saída, tal como, *bytecodes* Java usando o *Jasmin*[5].

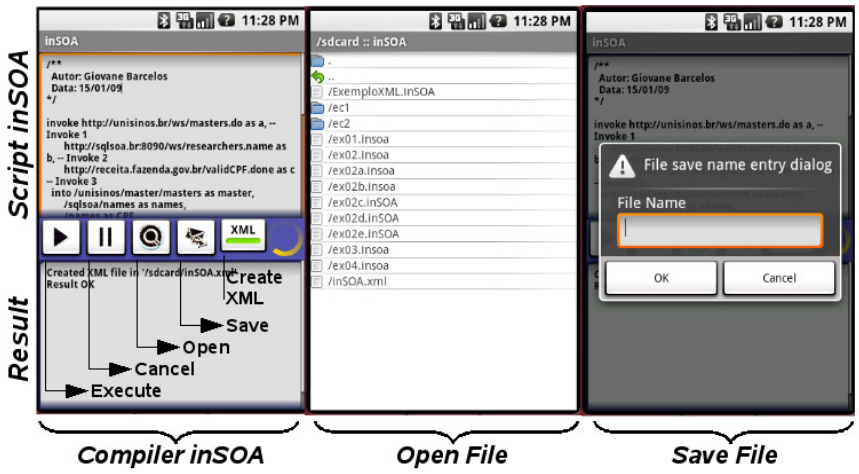

Fig. 4. Interface do protótipo *inSOA*.

4 Estudo de Caso e Resultados

Esta seção apresenta os cenários do estudo de caso e os resultados da utilização da linguagem *inSOA*. Foram desenvolvidos dois cenários de avaliação: consulta de livros e pesquisa para viagem. A consulta de livros realizou a chamada de 9 *WS* divididos

5 *Jasmin - http://jasmin.sourceforge.net/*

em outras 12 composições de serviços intermediárias com o objetivo de consultar livros da *Amazon Books* e buscar informações de câmbio, logística, entre outros. A pesquisa de viagem, por sua vez, utilizou 10 *WS* em 10 composições onde informações de estadia, vôos, aeroportos, condições do tempo, entre outros, foram resgatados.

Todos os cenários foram executados e avaliados no emulador *Android*. Cada composição de serviço com *inSOA* foi publicada e executada em uma máquina e emulador distintos no servidor de composição *SmallSOA*.

Na tabela 2 são apresentados os tempos médios de execução em segundos das composições no motor *SmallSOA* e a execução dos *WS* de forma individual. Como pode-se perceber as composições executadas no motor de composição *SmallSOA* levam em média 36% menos tempo que os *WS* executados individualmente. Isto ocorre, principalmente, porque o motor executa a maioria dos *WS* das composições em paralelo enquanto no processo individual isto é realizado seqüencialmente.

Tabela 2: Tempo de execução das composições no motor *SmallSOA* e individual.

#	Cenário I – *getInformationTravel*		Cenário II - *getBooks*	
	***WS* Individual**	**Composição**	***WS* Individual**	**Composição**
1	38,815 s	25,230 s	36,048 s	22,350 s
2	42,753 s	34,630 s	29,722 s	23,480 s
3	46,474 s	26,490 s	44,803 s	26,210 s
4	35,680 s	26,760 s	49,252 s	36,200 s
5	42,103 s	24,420 s	65,456 s	37,310 s
6	33,363 s	22,520 s	36,727 s	24,240 s
7	45,805 s	28,857 s	52,560 s	32,640 s
8	57,544 s	33,951 s	51,794 s	27,451 s
9	33,247 s	25,600 s	33,720 s	25,290 s
10	60,122 s	29,460 s	47,260 s	25,993 s
Média	**43,591 s**	**27,792** s	**44,734 s**	**28,116** s

Durante os testes foi avaliada a aderência da linguagem *inSOA* e *BPEL* aos *patterns* de composição, assim como, alguns *patterns* foram testados nos cenários do estudo de caso. Como pode-se perceber na tabela 3, a *BPEL* é aderente a 67% dos principais *patterns* de composição, enquanto a *inSOA* têm 95% de aderência, dos quais 47% foram testados no estudo de caso.

Tabela 3: *Patterns* testados nos cenários do estudo de caso versus *BPEL*.

#	***Pattern***	**BPEL**	**inSOA**	
			Aderência?	**Testado?**
1	*Sequence*	Sim	Sim	Sim
2	*Parallel Split*	Sim	Sim	Sim
3	*Synchronization*	Sim	Sim	Sim
4	*Exclusive Choice*	Sim	Sim	Não
5	*Simple Merge*	Sim	Sim	Sim
6	*Multi-choice*	Sim	Sim	Não
7	*Synchronizing Merge*	Sim	Sim	Sim
8	*Multi-merge*	Não	Sim	Sim
9	*Discriminator*	Não	Sim	Não

#	*Pattern*	BPEL	inSOA	
			Aderência?	Testado?
10	*Arbitrary Cycles*	Não	Sim	Não
11	*Implicit Termination*	Sim	Sim	Sim
12	*Multiple Instances Without Synchronization*	Sim	Sim	Não
13	*Multiple Instances With a Priori Design*	Sim	Sim	Não
14	*Multiple Instances With a Priori Runtime*	Não	Sim	Não
15	*Multiple Instances Without a Priori Runtime*	Não	Sim	Não
16	*Deferred Choice*	Sim	Sim	Não
17	*Interleaved Parallel*	Não	Sim	Não
18	*Milestone*	Não	Não	Não
19	*Cancel Activity*	Sim	Sim	Sim
20	*Cancel Case*	Sim	Sim	Sim
21	*Exception Handling*	Sim	Sim	Não

Foi realizada também uma comparação do tamanho de composições desenvolvidas em *BPEL* e *inSOA*, apresentada na tabela 4. As composições utilizadas nesta comparação foram extraídas dos exemplos fornecidos com o servidor de composição *ActiveVOS*[6].

Tabela 4: Comparação de composições *BPEL* e *inSOA*.

#	Composição	*BPEL*		*inSOA*		% (1 - *inSOA* / *BPEL*)	
		Caracteres	Palavras	Caracteres	Palavras	Caracteres	Palavras
1	*ForEach*	5370	748	262	39	95,12%	94,79%
2	*IsolatedScope*	3000	406	231	40	92,30%	90,15%
3	*MultiStartReceives*	5703	795	374	61	93,44%	92,33%
4	*RepeatUntil*	4444	520	225	29	94,94%	94,42%
5	*While*	4698	656	173	26	96,32%	96,04%
Médias Totais		**4643**	**625**	**253**	**39**	**94,55%**	**93,76%**

Como pode-se perceber na tabela 4, o *inSOA* na médias das composições analisadas, é 94,55% caracteres e 93,76% palavras menor que a linguagem *BPEL*. Este percentual justifica porque a linguagem *inSOA* não possui *tags* e declarações de *namespaces*, tal como, a linguagem *BPEL*, que é baseada em *XML*. Além disso, a linguagem *inSOA* é mais leve e direta na declaração de composições e, possui estruturas *XPath* que facilitam na navegação dos resultados.

5 Conclusão

Desde o início da *web* até os dias atuais o principal objetivo da *internet* tem sido a colaboração entre as pessoas. Esta colaboração, mais recentemente, tem sido estendida para troca de dados e informações entre aplicativos. A mais promissora das tecnologias para suportar estas trocas é a arquitetura *SOA*, que é baseada em *WS* e inspirada na orquestração e composição de serviços. Por sua vez, a composição de serviços realizada por linguagens de composição de serviços, têm estimulado ainda mais a colaboração e aproveitamento de serviços na *web*. Serviços esses, que estão crescendo e se popularizando na internet na mesma velocidade em que os dispositivos

6 *ActiveVOS – http://www.activevos.com/*

móveis vêm substituindo os dispositivos de mesa.

Este trabalho apresentou *inSOA*, uma linguagem declarativa de composição de serviços desenvolvida para rodar em dispositivos móveis. Inicialmente foi realizada uma revisão bibliográfica das principais tecnologias e trabalhos relacionados, seguido pela apresentação da linguagem *inSOA* e finalizada com um estudo de caso abrangendo dois cenários.

Referências

1. Berners-Lee, T. Information Management: A Proposal. CERN, March 1989, The Original Document File. http://www.w3.org/History/1989/proposal.html. Mai/09.
2. Nakamura, Masahide; Igaki, Hiroshi; Tamada, Haruaki e Matsumoto, Ken-Ichi. Implementing Integrated Services of Networked Home Appliances using service oriented Architecture. 2nd International Conference on Service Oriented Computing. 2004.
3. IBM. Site oficial da IBM. Arquitetura Orientada a Serviço (SOA). http://www-306.ibm.com/software/br/info/topic/openenvironment/soa/. Mai/09.
4. Microsoft. Criando web services Seguros. http://www.microsoft.com/brasil/security/guidance/topics/devsec/secmod85.mspx . Mai/09.
5. Hackmann, G.; Haitjema, M.; Gill, C. e Roman, G.-C. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. Washington University, Department of Computer Science and Engineering, St. Louis, Missouri. ICSOC 2006 .
6. Zanuz, L. ; Barcelos, G. ; Filippetto, A. ; Pinto, S. C. C. S.: U-SOA - Towards a Ubiquitous Platform Based on Service - Oriented Architecture. In: XIV Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), Vila Velha, (2008).
7. Srirama, S; Jarke, M. e Prinz, W. Mobile Host: A feasibility analysis of mobile Web Service provisioning. UMICS '06, 2006.
8. Balani, Naveen. Using kXML to access XML files on J2ME devices. https://www6.software.ibm.com/developerworks/education/wi-kxml/wi-kxml-a4.pdf. Mai/09.
9. HP SOA Quality - Deliver outstanding SOA web service performance. Disponível em: <http://h71028.www7.hp.com/enterprise/cache/484284-0-0-225-121.html>. Mai/09.
10. W3C. Web Services Architecture. http://www.w3.org/TR/ws-arch/#introduction. Mai/09.
11. W3C. XML Path Language (Xpath) 2.0. W3C Working Draft. http://www.w3.org/TR/xpath20/. 2003.
12. Ang, J.; Cherbakov, L.; Ibrahim, M.. SOA antipatterns. The obstacles to the adoption and successful realization of Service-Oriented Architecture, Nov 2005. http://www.ibm.com/developerworks/webservices/library/ws-antipatterns/. 2005.
13. Pijanowski, Keith. Visibility and Control in a Service-Oriented Architecture. MSDN Architecture Center, Technical Articles. http://msdn2.microsoft.com/en-us/library/bb507204.aspx. Mai/09.
14. Crespo, Sérgio. Composição em WebFrameworks. Tese de Doutorado. PUC-RIO. 2000.
15. Aalst, W.M.P van der; Dumas, M. e Hofstede, A.H.M. Ter. Web service composition languages: old wine in New bottles?. Euromicro Conference. IEEE. 2003.
16. Gammar, E.; Helm, R.; Johnson, R. e Vlissides, J.M. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 416 Pág. ISBN: 978-0201633610. 1994.
17. Will, van der Aalst; A.H.M. ter Hofstede; B. Kiepuszewski e A.P. Barros. Workflow Patterns. QUT Technical report. FIT-TR-2002-02. Queensland University of Technology. Brisbane. 2002.