**UNIVERSIDADE DO VALE DO RIO DOS SINOS**

**Programa Interdisciplinar de Pós-Graduação em**

# Computação Aplicada

**Mestrado Acadêmico**

Luciano Zanuz

# SMALLSOA – Um Motor para Execução de Composições de Serviços em Ambientes Móveis

**Luciano Zanuz**

# SMALLSOA – Um Motor para Execução de Composições de Serviços em Ambientes Móveis

Dissertação apresentada à Universidade do Vale do Rio dos Sinos como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

**Orientador: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto**

São Leopoldo

2009

Z34s Zanuz, Luciano

SMALLSOA: um motor para execução de composições de serviços em ambientes móveis / por Luciano Zanuz, 2009.

143 f. : tab.; graf. + anexos : 30cm.

Dissertação (mestrado) -- Universidade do Vale do Rio dos Sinos, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, 2009.

"Orientação: Prof. Dr. Sérgio Crespo C. S. Pinto; Ciências Exatas e Tecnológicas".

1. *Web service*. 2. *Service oriented architecture* (SOA). 3. Arquitetura orientada a serviços. 4. Computação móvel. 5. Computação ubíqua. 6. Android. I. Título.

CDU 004.738.52:004.4

Catalogação na Publicação: Ivete Lopes Figueiró - CRB 10/509

Dedico este trabalho às pessoas mais importantes da minha vida: minha noiva Denise e meus pais João e Elda. Vocês foram muito importantes nessa caminhada. Amo vocês!

# AGRADECIMENTOS

A Deus pelo dom da vida e por dar-me forças para lutar sempre.

À minha amada noiva Denise, por seu amor, pelo apoio nos momentos difíceis e por saber compreender as privações e ausências decorrentes do mestrado.

Aos meus queridos pais, João e Elda, exemplos de luta e de vida que levarei comigo para sempre, por não medirem esforços para proporcionar-me a melhor educação possível.

Aos meus irmãos Gustavo e Leonardo, minha cunhada Grasiela, meus sogros Mercino e Mara, pessoas muito especiais, por seu apoio e carinho sinceros.

Ao professor Sérgio Crespo pela sua orientação e conselhos construtivos que serviram para o desenvolvimento deste trabalho.

Aos professores Hugo Fuks e Jorge Barbosa por sua participação na banca e por suas valiosas contribuições no refinamento do trabalho.

Aos meus colegas do projeto U-SOA, Giovane e Alexsandro, pelo companheirismo e ajuda prestados.

Aos alunos do curso de Engenharia da Computação, Cládio, Douglas e João, que compraram a idéia e nos auxiliaram no projeto U-SOA.

Aos professores e funcionários do PIPCA que sempre que foram solicitados deram a sua valorosa contribuição.

Aos meus amigos que souberam compreender a minha ausência.

A CAPES pelo apoio financeiro.

E a todos aqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

Nas grandes batalhas da vida, o primeiro passo para a vitória é o desejo de vencer!

**Mahatma Gandhi**

# RESUMO

Arquiteturas orientadas a serviços (SOA) voltaram a ganhar destaque, após o surgimento dos *web services*, como uma tecnologia capaz de melhorar consideravelmente a interoperabilidade entre aplicações de *software*. Para apoiar a crescente necessidade de colaboração entre pessoas fisicamente separadas, como em sistemas de trabalho ou ensino à distância, ambientes colaborativos estão sendo construídos utilizando a Internet como plataforma. Com a popularização dos dispositivos móveis, *softwares* que permitem interações entre quaisquer dispositivos, móveis ou não, utilizando tecnologias variadas de comunicação rumo a um mundo ubíquo, vêm ganhando destaque. SOA, por sua vez, está sendo considerada uma das plataformas mais indicadas para sistemas colaborativos ubíquos.

Este trabalho apresenta SmallSOA, um motor para execução de composições de serviços em ambientes móveis, um dos componentes centrais da arquitetura orientada a serviços chamada U-SOA que o grupo de pesquisa de Engenharia de *Software* e Linguagens de Programação da Unisinos está desenvolvendo. Um motor para execução de composições de serviços é um componente de *software* que, dada uma linguagem de composição de serviços, interpreta uma descrição de composição escrita nessa linguagem e processa-a, normalmente retornando algo para um usuário cliente. Devido às peculiaridades intrínsecas dos ambientes móveis, SmallSOA trata questões relativas a esse tipo de ambiente.

**Palavras-chave:** SOA; *web services*; computação móvel; computação ubíqua; motor de execução; Android

# *ABSTRACT*

*Services-oriented architectures (SOA) restarted to gain prominence after the emergence of web services as a technology able to improve the interoperability between software applications. To support the growing need for collaboration between people physically separated as in distance work or learning systems, collaborative environments are being constructed using the Internet as platform. With the popularization of mobile devices, software that allows interactions between any devices, mobile or not, using different communication technologies towards a ubiquitous world are gaining prominence. SOA, in its turn, is considered one of the most suitable platforms for ubiquitous collaborative systems.*

*This work presents SmallSOA, an engine for services compositions execution on mobile environments, one of the central components of the service oriented architecture called U-SOA that the research team on Software Engineering and Programming Languages at Unisinos is developing. An engine for services compositions execution is a software component that, given a services composition language, it interprets a composition description written in that language and processes it, usually returning something to a user client. Due to the peculiarities inherent on mobile environments, SmallSOA addresses issues of this kind of environment.*

***Keywords:*** *SOA; web services; mobile computing; ubiquitous computing; execution engine; Android*

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE QUADROS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| 3C | Comunicação, Coordenação e Cooperação |
| 3G | *Third Generation of Mobile Phone Standards* |
| AAC | *Advanced Audio Coding* |
| ADO | *ActiveX Data Objects* |
| AMR | *Adaptive Multi-Rate* |
| API | *Application Programming Interface* |
| BEEP | *Blocks Extensible Exchange Protocol* |
| BPEL | *Business Process Execution Language* |
| BPEL4WS | *Business Process Execution Language For Web Services* |
| CDC | *Connected Device Profile* |
| CDMA | *Code Division Multiple Access* |
| CLDC | *Connected Limited Device Configuration* |
| CLR | *Common Language Runtime* |
| COM | *Component Object Model* |
| CORBA | *Common Object Request Broker Architecture* |
| CPF | Cadastro de Pessoas Físicas |
| CPU | *Central Processing Unit* |
| CSCL | *Computer Supported Collaborative Learning* |
| CSCW | *Computer Supported Cooperative Work* |
| CWE | *Collaborative Working Environments* |
| DARPA | *Defense Advanced Research Projects Agency* |
| DCOM | *Distributed Component Object Model* |
| DHCP | *Dynamic Host Configuration Protocol* |
| DNS | *Domain Name System* |
| EDGE | *Enhanced Data Rates for GSM Evolution* |
| *E-Business* | *Eletronic Business* |
| *E-Government* | *Eletronic Government* |
| *E-Mail* | *Eletronic Mail* |
| EV-DO | *Evolution Data Only/Evolution Data Optimized* |
| FTP | *File Transfer Protocol* |
| GIF | *Graphics Interchange Format* |

| | |
|---|---|
| GPS | *Global Positioning System* |
| GPRS | *General Packet Radio Service* |
| GSM | *Global System for Mobile* |
| H.264 | *MPEG-4 Part 10* ou *MPEG-4 AVC (Advanced Video Coding)* |
| HTML | *Hyper Text Markup Language* |
| HTTP | *Hyper Text Transfer Protocol* |
| HTTPS | *Hypertext Transfer Protocol Secure* |
| IDE | *Integrated Development Environment* |
| inSOA | *Invoke for Service-Oriented Architecture* |
| IP | *Internet Protocol* |
| Java EE | *Java Platform, Enterprise Edition* |
| Java ME | *Java Platform, Micro Edition* |
| Java SE | *Java Platform, Standard Edition* |
| JAX-RPC | *Java API for XML-based RPC* |
| JAXP | *Java API for XML Processing* |
| JME | *Java Platform, Micro Edition* |
| JMS | *Java Message Service* |
| JPG | *Joint Photographic Experts Group* |
| JSR | *Java Specification Request* |
| JVM | *Java Virtual Machine* |
| KB | *Kilobyte* |
| MANET | *Mobile Ad-hoc Network* |
| MIDP | *Mobile Information Device Profile* |
| MMS | *Multimedia Messaging Service* |
| MP3 | *MPEG Audio-Layer 3* |
| MPEG | *Moving Picture Experts Group* |
| MPEG4 | *Moving Picture Experts Group 4* |
| ms | Milissegundo |
| MWS | *Mobile Web Server* |
| NAT | *Network Address Translation* |
| NTLM | *NT LAN Manager* |
| P2P | *Peer-to-Peer* |
| PC | *Personal Computer* |
| PDA | *Personal Digital Assistant* |

| | |
|---|---|
| PNG | *Portable Network Graphics* |
| QoS | *Quality of Service* |
| RPC | *Remote Procedure Call* |
| SDK | *Software Development Kit* |
| SGML | *Standard Generalized Markup Language* |
| SMS | *Short Message Service* |
| SMTP | *Simple Mail Transport Protocol* |
| SOA | *Service-Oriented Architecture* |
| SOAP | *Simple Object Access Protocol* |
| SQL | *Structured Query Language* |
| SQLSOA | *Structured Query Language for Service-Oriented Architecture* |
| TI | Tecnologia da Informação |
| TIC | Tecnologia da Informação e Comunicação |
| U-SOA | *Ubiquitous Service-Oriented Architecture* |
| UDDI | *Universal Description, Discovery and Integration* |
| URL | *Uniform Resource Locator* |
| WAP | *Wireless Application Protocol* |
| W3C | *World Wide Web Consortium* |
| WCF | *Windows Communication Foundation* |
| Wi-Fi | *Wireless Fidelity* |
| WS-BPEL | *Web Services Business Process Execution Language* |
| WS-I | *Web Services Interoperability Organization* |
| WSCDL | *Web Service Composition Description Language* |
| WSDL | *Web Service Description Language* |
| WSFL | *Web Services Flow Language* |
| XML | *eXtensible Markup Language* |
| *XPath* | *XML Path Language* |

# 1 INTRODUÇÃO

A colaboração de grupos de pessoas para a realização de um objetivo comum é consensualmente apontada como um requisito fundamental para a obtenção de sucesso na realização de tarefas mais complexas, onde uma única pessoa levaria um tempo muito grande para realizá-la ou, até mesmo, poderia nunca conseguir finalizá-la. Essa colaboração, também conhecida por trabalho em grupo, é um conceito antigo e não está relacionado a questões tecnológicas, mas apenas à realização de determinadas tarefas por grupos de pessoas.

A crescente globalização iniciada nos últimos anos propiciou diversas mudanças no comportamento das pessoas e das organizações. Atualmente, com esse desaparecimento virtual das fronteiras é comum que, em busca de mão-de-obra mais barata, empresas terceirizem todo ou parte do desenvolvimento de *software* em outros países, ou seja, *offshore*. Além disso, cada vez mais técnicas de ensino à distância estão sendo empregadas para auxiliar, e algumas vezes substituir, o ensino tradicional.

Para apoiar essa necessidade de colaboração e aproveitando a impressionante popularização da Internet, que forneceu condições para que qualquer pessoa interaja com outra independentemente da sua localização, ambientes colaborativos estão sendo construídos utilizando a Internet como plataforma. Permitem, dessa forma, que equipes geograficamente separadas trabalhem em grupo como se estivessem em um mesmo local. Ambientes colaborativos estão sendo popularizados também na *World Wide Web*, mais especificamente pela *Web* 2.0, também conhecida como *Collaborative Web* (*Web* Colaborativa), que se baseia no contexto da construção de conteúdos de *web sites* através da constante colaboração dos seus usuários.

A evolução da computação durante o século XX e neste início de século XXI é notória. Inicialmente baseada em grandes e caros equipamentos, que chegavam a possuir o tamanho de uma sala, foi gradualmente diminuindo o tamanho do *hardware* e seu respectivo custo, ao mesmo tempo em que aumentava seu poder computacional. Atualmente, existem pequenos dispositivos móveis dos mais diversos tipos, tais como *laptops*, PDA’s, celulares, *smartphones*, entre outros, os quais estão sendo cada vez mais utilizados pela população.

Há mais de 15 anos, Weiser afirmou que as tecnologias mais profundas são aquelas que desaparecem [Weiser 1991], em seu clássico artigo onde previu com muita precisão a

massiva utilização de tecnologias móveis que efetivamente viriam a surgir. Entre os exemplos citados está a questão da escrita, que é a habilidade de capturar uma representação simbólica falada e transcrevê-la para alguma forma física. Segundo Weiser, para que a computação realmente se transformasse em algo profundo ela precisaria ser transparente para os usuários, estando presente em todos os locais, ou seja, ser ubíqua. Computação ubíqua, ou pervasiva, diz respeito à existência de ambientes saturados com capacidades de computação e comunicação e plenamente integrados com usuários humanos [Satyanarayanan 2001].

O surgimento dos *web services* estimulou discussões sobre arquiteturas orientadas a serviço, que haviam surgido há mais de uma década. Por abstrair questões referentes às plataformas de implementações de sistemas e de *hardware*, esse tipo de arquitetura freqüentemente está sendo considerado como possível solução para muitos problemas da indústria de Tecnologia da Informação e Comunicação (TIC). Segundo [Laso-Ballesteros 2006], devido às suas características, é fortemente aconselhado que ambientes colaborativos sejam implementados sobre arquiteturas orientadas a serviço.

A utilização de SOA como arquitetura para computação móvel e ubíqua é plenamente viável e pode ser considerada como uma tendência. Esse novo modelo computacional englobaria todo o tipo de dispositivo, móvel ou não, os quais poderiam integrar uma grande rede de serviços, ou seja, uma rede de funcionalidades distribuídas, localizadas efetivamente nos mais diversos tipos de *hardware*. Todos esses serviços, independente de plataforma, poderiam integrar aplicações SOA, abstraindo-se as arquiteturas de *software* e de *hardware* de cada implementação, devido à utilização dos protocolos padrões da indústria baseados em XML, como SOAP e WSDL. Por exemplo, um condicionador de ar poderia disponibilizar um serviço para controle de temperatura, que seria acionado quando se quisesse aumentar ou diminuir a temperatura de um ambiente. Através de um sistema sobre SOA, seria possível encontrar dinamicamente esse serviço e utilizá-lo para controlar a temperatura do ambiente.

## 1.1 Motivação

A evolução da computação, que por um lado diminuiu o tamanho dos dispositivos e que por outro lado incrementou o seu respectivo poder computacional, aliada à evolução das redes de comunicação, está permitindo que estruturas de *hardware* e *software* sejam criadas para suportarem a construção de sistemas distribuídos ubíqüos. Esses sistemas, por sua vez, apoiarão a implementação de ambientes colaborativos globais que permitirão que pessoas

geograficamente separadas trabalhem de forma colaborativa como se estivessem em um mesmo local físico.

Arquiteturas orientadas a serviço estão sendo consideradas como a principal plataforma de *software* sobre a qual sistemas colaborativos ubíqüos serão construídos. Isso se deve às próprias características de SOA, tais como baixo acoplamento e independência de plataforma. Juntamente com SOA, a utilização de ontologias vem ganhando destaque como plataforma base para esse tipo de sistema.

## 1.2 Definição do problema

Atualmente, arquiteturas orientadas a serviço são implementadas de forma que o fornecimento de serviços, implementados como *web services*, seja feito através de plataformas de *hardware* de alto poder de processamento, implantados em grandes servidores corporativos. Os clientes atuais dessas arquiteturas usualmente são *desktops*, embora já esteja ocorrendo uma popularização do acesso a *web services* através de pequenos dispositivos móveis tais como PDA's ou *smartphones*.

Entretanto, para que sistemas colaborativos ubíquos tornem-se realidade, o fornecimento de serviços também deve ocorrer a partir desses pequenos dispositivos móveis. Todo o dispositivo, ou par, envolvido em uma relação de colaboração baseada em SOA, deve poder tanto ser um cliente como um servidor de serviços. Mas, para aproveitar todo o poder que um ambiente SOA proporciona, esses pares (*peers*) devem também ser capazes de compor esses serviços em novas aplicações.

Portanto, a questão central a ser respondida por este trabalho é como executar composições de serviços em pequenos dispositivos móveis.

## 1.3 Objetivos

Este trabalho tem por objetivo a implementação de um motor para execução de composições de serviços em dispositivos móveis. Esse objetivo está alinhado com a criação de infra-estrutura para permitir a construção de sistemas colaborativos ubíqüos baseados em SOA, que é a proposta do projeto maior chamado U-SOA no qual o motor está inserido e que o grupo de pesquisa de Engenharia de *Software* e Linguagens de Programação da Unisinos está desenvolvendo. O motor deve possibilitar que qualquer dispositivo móvel seja capaz de ser tanto um fornecedor como um cliente de *web service*, sendo que esses dispositivos serão

implementados como pares em uma relação *ad-hoc* tipicamente P2P, permitindo assim a plena colaboração desses pares.

Para alcançar esse objetivo, alguns pré-requisitos devem ser observados, tais como:

- Estudo do estado-da-arte das principais infra-estruturas existentes para permitir a construção de sistemas colaborativos ubíqüos baseados em SOA.
- Definição da arquitetura geral, incluindo aspectos de alto-nível e relacionamentos com outros sistemas e outros trabalhos do grupo de pesquisa.
- Escolha da plataforma sobre a qual a arquitetura será implementada.

Observados os requisitos acima, é possível elencar os objetivos específicos desta dissertação de mestrado, referentes à construção do motor para execução de composições de serviços em dispositivos móveis:

- Definição da arquitetura do motor, incluindo a definição de todos os componentes e como se comunicam.
- Construção do motor, através da implementação de novos componentes e da customização de componentes existentes que forem utilizados. O motor deve:
  - Comunicar-se através dos padrões da indústria.
  - Ser executado em dispositivos móveis.
  - Executar *web services* implantados em quaisquer dispositivos, móveis ou não.
  - Executar *web services* descritos por WSDL.
  - Suportar estruturas de desvio de fluxo.
  - Suportar estruturas de iteração.
  - Suportar a execução de *web services* em paralelo, de forma concorrente.
  - Possuir política para casos em que *web services* não puderem ser executados.
- Validação do motor.
- Apresentação dos principais resultados obtidos, comparando-os com trabalhos similares.

## 1.4 Organização da dissertação

Esta dissertação de mestrado contém 8 capítulos, sendo que a sua seqüência está dividida da seguinte forma:

- Capítulo 2: Revisão Bibliográfica. Descreve as principais tecnologias relacionadas com a dissertação, divididas em ambientes colaborativos, computação ubíqua, arquiteturas orientadas a serviços e plataformas de desenvolvimento.
- Capítulo 3: Trabalhos Relacionados. Apresenta alguns dos principais trabalhos relacionados com a dissertação, referentes a infra-estruturas para computação ubíqua baseadas em SOA.
- Capítulo 4: SmallSOA. Apresenta o projeto U-SOA e descreve as características do motor para execução de composições de serviços em ambientes móveis chamado SmallSOA, inserido como componente daquele projeto.
- Capítulo 5: Construção do motor SmallSOA. Descreve os aspectos referentes à implementação do motor.
- Capítulo 6: Resultados. Apresenta os resultados obtidos na validação simples do motor e na execução de dois cenários de teste, os quais validam as funcionalidades de SmallSOA e de parte da arquitetura U-SOA.
- Capítulo 7: Conclusões. Apresenta as conclusões obtidas ao final do trabalho, bem como sugestões de trabalhos futuros que podem dar prosseguimento aos estudos do corrente tema.
- Capítulo 8: Referências. Lista as referências bibliográficas utilizadas para dar embasamento teórico e conceitual à dissertação.

# 2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma visão geral dos principais conceitos e tecnologias relacionados ao motor para execução de composições de serviços em dispositivos móveis que é explicado no capítulo 4 e que dão o devido suporte teórico e prático à dissertação. A seção 2.1 aborda os ambientes colaborativos, cenário de utilização da arquitetura projetada. A seção 2.2 aborda a computação ubíqua, paradigma computacional contemporâneo e com grande futuro. Na seção 2.3 são apresentadas arquiteturas orientadas a serviço e seus componentes, entre eles *web services* e seus principais padrões. Para finalizar a revisão bibliográfica, a seção 2.4 apresenta as principais plataformas de desenvolvimento de aplicações ubíquas baseadas em SOA.

## 2.1 Ambientes colaborativos

Um espaço de trabalho colaborativo é um ambiente interconectado no qual todos os participantes, em locais dispersos, podem acessar e interagir com os outros como se estivessem dentro de uma única entidade. Ambientes colaborativos dão suporte à colaboração de grupos de pessoas para a realização de um objetivo comum. Embora na prática possam ser ambientes físicos que proporcionem suporte à colaboração, do ponto de vista da engenharia de *software* são sistemas de informação baseado em computador e apoiados por comunicações eletrônicas que suportam colaboração e que habilitam os participantes a superarem os diferenciais de espaço e tempo, incrementar a produtividade e a reduzir custos.

*Softwares* para colaboração, também conhecidos como *groupwares,* são ferramentas de *software* que integram o trabalho de múltiplos usuários concorrentes localizados em espaços de trabalho separados. Usualmente, um pacote de *groupware* consiste de diversas aplicações *web* projetadas para a automação das atividades colaborativas dos usuários. O objetivo do *groupware* é auxiliar grupos a comunicarem-se e a colaborarem e na coordenação das suas atividades. Segundo [Ellis 1991], *groupware* são sistemas baseados em computador que apóiam grupos de pessoas engajados em tarefas ou objetivos comuns e que fornece uma interface para um ambiente compartilhado.

Duas das principais abordagens de ambientes colaborativos são trabalho colaborativo (Trabalho Cooperativo Apoiado por Computador – CSCW) e ensino colaborativo

(Aprendizagem Cooperativa Apoiada por Computador – CSCL), sendo que ambos se apóiam nos conceitos de Ambientes de Trabalho Colaborativo (CWE).

CWE's são definidos como recursos integrados e conectados que fornecem acesso compartilhado a conteúdos e que permitem que atores distribuídos trabalhem de forma conjunta em direção a objetivos comuns. A pesquisa em CWE envolve características organizacionais, técnicas e sociais. As seguintes aplicações ou serviços são considerados elementos de um CWE:

- *E-mail*;
- Mensagens instantâneas;
- Compartilhamento de aplicações;
- Vídeo-conferência;
- Espaço de trabalho colaborativo e gerenciamento de documentos;
- Gerenciamento de tarefa e fluxo de trabalho;
- Grupos Wiki ou esforço comunitário para editar páginas Wiki; por exemplo, páginas Wiki descrevendo conceitos para habilitar um entendimento comum dentro de um grupo ou comunidade;
- Atividade de Blog, onde entradas são categorizadas por grupos ou comunidades ou outros conceitos suportando colaboração.

Para a construção de uma meta-arquitetura de *software* que forneça um *framework* para a criação de aplicações colaborativas é necessário compreender exatamente o que vem a ser a colaboração. Para isso, vários modelos têm sido usados para entender como pessoas trabalham em grupo [Soliman 2005]:

- O modelo de comunicação de Shannon e Weaver (1949) serve como uma descrição para trabalho em grupo. O modelo prevê que uma mensagem seja codificada por um remetente, transmitida através de um canal e decodificada por um destinatário.
- A abordagem de Hymes e Olson (1992) determina atividades que são comuns a todos os trabalhos cooperativos, às quais eles chamaram de atividades componentes. Os autores afirmaram ainda que, após essa definição, é necessária a separação das atividades componentes e a análise das várias características de ferramentas e seus efeitos sobre cada atividade.
- Gutwin e Greenberg (2000) introduziram os mecanismos da colaboração, que podem ser descritos como atividades componentes. Esses mecanismos foram

extendidos [Baker 2001] e são: comunicação explícita, comunicação conseqüente, coordenação da ação, planejamento, monitoração, assistência e proteção.

- Malone e Crowston (1990) introduziram a Teoria da Coordenação, um conjunto de princípios interdisciplinares sobre como atores podem trabalhar de forma conjunta em harmonia.
- O modelo 3C [Ellis 1991], apresentado na figura 2.1, foi muito difundido na literatura [Fuks 2003]. O modelo baseia-se no fato de que, para colaborarem, os indivíduos têm que trocar informações (comunicar), organizar-se (coordenarem-se) e operar em conjunto num espaço compartilhado (cooperar). Uma abordagem de desenvolvimento de *groupware* baseada nesse modelo pode ser encontrada em [Gerosa 2006].

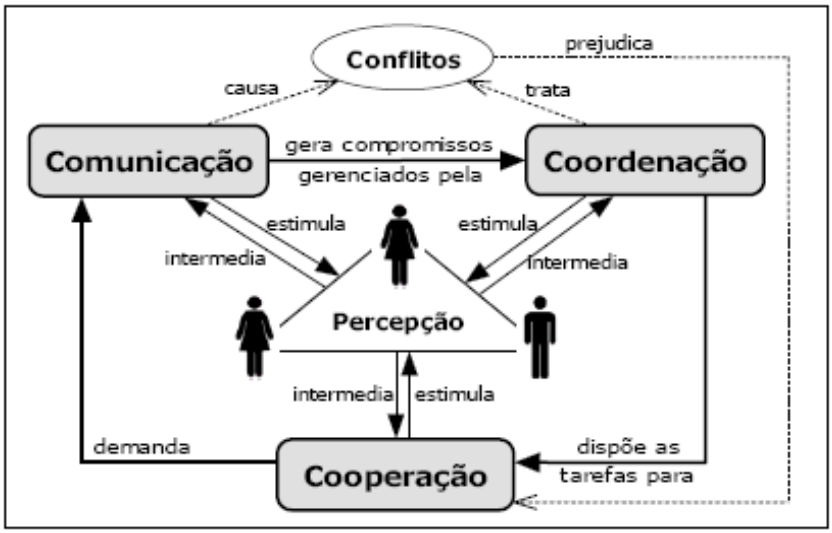


**Figura 2.1 – O modelo 3C**

[Soliman 2005] propôs um *framework* alternativo aos apresentados acima, fornecido através de oito ingredientes essenciais para a colaboração, que são úteis para identificar aspectos para a melhoria, tanto da colaboração tecnológica, quanto de processos de colaboração humana. Os oito ingredientes essenciais, derivados da literatura, estudos de caso, observações e experimentos são:

1. Duas ou mais pessoas;
2. Espaço compartilhado;
3. Tempo;
4. Um objetivo comum;
5. Foco no objetivo;

6. Linguagem comum;
7. Conhecimento sobre a área do objetivo;
8. Interação.

Um dos elementos-chave para dar suporte a ambientes colaborativos são infra-estruturas colaborativas. Segundo [Laso-Ballesteros 2006], essas infra-estruturas têm que fornecer os componentes de sistema necessários para as aplicações colaborativas e, ao mesmo tempo, estarem de conformidade com os princípios de arquiteturas orientadas a serviço (SOA).

Um dos maiores exemplos de ambientes colaborativos, atualmente, é a *Web* 2.0, que atualmente está sendo chamada de *Collaborative Web* (*Web* Colaborativa). Segundo [O'Reilly 2005], *Web* 2.0 é a mudança para uma Internet como plataforma e um entendimento das regras para se obter sucesso nessa nova plataforma. Entre outras, a regra mais importante é desenvolver aplicativos que aproveitem os efeitos de rede para se tornarem melhores quanto mais são usados pelas pessoas, aproveitando a inteligência coletiva.

## 2.2 Computação ubíqua

Computação ubíqua diz respeito à existência de ambientes saturados com capacidades de computação e comunicação e plenamente integrados com usuários humanos [Satyanarayanan 2001]. Essa integração plena com usuários humanos refere-se à invisibilidade citada por [Weiser 1991], que significa a integração das pessoas com os *softwares* e *hardwares* de modo que essa relação passe despercebida.

Segundo [Satyanarayanan 2001], em seu clássico *survey* sobre computação ubíqua, para entender melhor a evolução da computação atingida desde a metade da década de 1970 até a computação ubíqua, deve-se voltar a atenção a dois passos anteriores nessa evolução: sistemas distribuídos e computação móvel.

Sistemas distribuídos surgiram da intersecção de computadores pessoais e redes locais. Seu estudo criou um *framework* conceitual e uma base algorítmica que provaram ter um valor duradouro em qualquer trabalho envolvendo dois ou mais computadores conectados em rede. A relação com a computação ubíqua se dá nas seguintes áreas: comunicação remota; tolerância à falha; alta disponibilidade; acesso à informação remota; segurança.

No que diz respeito à computação móvel, quatro limitações forçaram o desenvolvimento de técnicas especializadas para a computação ubíqua: variação imprevisível

da qualidade da rede; baixa confiança e robusteza de elementos móveis; limitações nos recursos locais impostos por limitações de peso e tamanho; preocupação com o consumo da bateria [Satyanarayanan 1996].

Além de todos os campos de pesquisa relativos à computação móvel, a computação ubíqua vai além e incorpora quatro campos adicionais:

- Uso efetivo de locais espertos (*smart spaces*). Significa embarcar infra-estrutura de computação em infra-estrutura de construção. Um exemplo simples é o ajuste automático da temperatura em uma sala baseada em um perfil eletrônico de seus ocupantes.
- Invisibilidade. O cenário ideal é o completo desaparecimento das tecnologias de computação da consciência do usuário [Weiser 1991]. Na prática, uma aproximação razoável seria a mínima distração do usuário. Se um ambiente de computação ubíqua continuamente alcança as expectativas do usuário e raramente apresenta surpresas, esse ambiente permite que o usuário interaja quase em nível subconsciente.
- Escalabilidade localizada. Escalabilidade, em um sentido amplo, é um problema crítico da computação ubíqua. À medida que os *smart spaces* crescerem em sofisticação e o seu uso aumentar, as interações com os usuários deverão privilegiar àqueles que estão mais perto fisicamente. Diferentemente do que ocorre com um *web server*, por exemplo, onde a escalabilidade tipicamente ignora a distância física.
- Condições desiguais mascaradas. O nível de penetração das tecnologias de computação ubíqua sobre a infra-estrutura irá variar consideravelmente dependendo de muitos fatores não-técnicos, tais como a estrutura organizacional, economia e modelos de negócio. Portanto, uma penetração uniforme levará muitos anos para ser alcançada, se é que isso ocorrerá. Resumindo, grandes diferenças persistirão no nível de esperteza dos diferentes ambientes. Por exemplo, o que está disponível em uma sala de conferência bem-equipada, escritório ou sala de aula pode ser mais sofisticado do que em outros locais.

O desenvolvimento de sistemas ubíqüos não é trivial. Diversas características devem ser motivos de preocupação no desenvolvimento desse tipo de sistema, implementadas sob a forma de requisitos. As principais são:

- Intenção do usuário. Para obter um comportamento efetivamente pró-ativo, um sistema ubíqüo deve rastrear o comportamento do usuário.
- *Cyber foraging*. Significa o uso oportuno de recursos computacionais disponíveis por dispositivos móveis. Refere-se à utilização de *hardware* de poder computacional maior como substituto (*surrogate*) do dispositivo móvel, quando houver disponibilidade.
- Estratégias de adaptação. Refere-se à necessidade de adaptação devido às diferenças significantes entre fornecedor e demanda de um recurso (banda da rede sem fio, energia, memória, etc.).
- Gerenciamento de energia de alto nível. Diz respeito à necessidade de gerenciar a energia devido às capacidades sofisticadas dos dispositivos, as quais estão aumentando a demanda de energia.
- Espessura dos clientes. Significa o poder computacional (CPU, memória, disco, etc.) dos dispositivos clientes.
- Consciência de contexto. Sistemas ubíqüos precisam ter consciência de contexto para poderem oferecer serviços adaptáveis, de forma pró-ativa e sem ser intrusivo. Baseiam-se em sistemas de localização [Hightower 2001].
- Balanceamento de pró-atividade e transparência. Deve haver muito cuidado com o nível de pró-atividade a ser utilizado para não aborrecer o usuário e acabar com o objetivo da invisibilidade
- Privacidade e confiança. Requisito indispensável e ainda maior em sistemas ubíqüos, devido ao fato da necessidade de utilização de mecanismos como rastreamento, locais espertos, substitutos, etc.
- Impacto nas camadas. Sistemas ubíqüos usualmente utilizam informações provenientes de diversos níveis de um sistema para produzir respostas efetivas.

## 2.3 Arquiteturas orientadas a serviços

O conceito clássico de uma arquitetura orientada a serviço (*service-oriented architecture* – SOA) a define como uma abordagem de desenvolvimento de *software* na qual funções chave, implementadas sob a forma de serviços, são construídas como componentes reutilizáveis que implementam padrões da indústria (XML) para comunicações interoperáveis. Fornece baixo acoplamento, interoperabilidade, habilidade de descobrimento, gerenciamento de alterações e operação de serviços de negócio em um ambiente bem

administrado. Segundo [Bih 2006], SOA pode ser considerada, simplesmente, como uma coleção de serviços.

SOA está diretamente relacionada com negócio, tendo como primeiro objetivo alinhar o mundo dos negócios com o mundo da Tecnologia da Informação de uma forma que ambos tornem-se mais efetivos. SOA é uma ponte que cria um relacionamento entre negócio e TI mais poderoso e valioso do que tudo o que já foi experimentado no passado. SOA diz respeito aos resultados que podem ser alcançados a partir de um melhor alinhamento entre o negócio e a TI [High Jr. 2005].

Serviços de negócio operando em um ambiente SOA podem ser, através de técnicas de orquestração ou coreografia, compostos em processos de negócio que alinham TI com o negócio [Pijanowski 2007]. Orientação a serviços é, portanto, uma forma de integrar negócio como um conjunto de serviços interligados [High Jr. 2005].

No entanto, essa abordagem que associa SOA com negócio não é mais a única forma de utilização de SOA. Cada recurso de TI, seja uma aplicação, sistema, parceiro de negócio ou dispositivo, pode ser acessado como um serviço, disponível através de interfaces. SOA está, aos poucos, se desassociando do negócio e se firmando como um estilo arquitetural para qualquer tipo de sistema. Por exemplo, devido a suas características de baixo acoplamento e independência de plataforma, SOA tornou-se o candidato ideal de arquitetura para uma grande variedade de sistemas, móveis e ubíqüos. Ambientes colaborativos também são fortes candidatos a utilizarem SOA. Segundo [Laso-Ballesteros 2006], CWE's serão compostos de aplicações colaborativas que, em muitos casos, seriam criadas por um usuário final sem conhecimento de TI. Nesse contexto, a palavra "criada" significa composição e orquestração de componentes de sistema, sendo que esses componentes de sistema têm que ser oferecidos pela infra-estrutura, a qual deve estar de conformidade com os princípios da orientação a serviços.

SOA é um estilo de arquitetura cujo objetivo é atingir baixo nível de acoplamento entre os agentes de *software* – serviços (*service loose coupling*) – que interagem entre si. Os demais princípios do projeto da orientação a serviços são [Erl 2007]:

- Contratos de serviço padronizados (*standardized service contracts*);
- Abstração de serviço (*service abstraction*);
- Reusabilidade de serviço (*service reusability*);
- Autonomia de serviço (*service autonomy*);

- Serviço sem estado (*service statelessness*);
- Descobrimento de serviço (*service discoverability*);
- Composição de serviço (*service composability*);
- Orientação a serviço e interoperabilidade (*service-orientation and interoperability*).

Serviços são o coração do SOA. Basicamente, serviços são unidades de trabalho criadas por um fornecedor de serviço para alcançar resultados finais desejados por um consumidor de serviço. Ambos, fornecedor e consumidor, são regrados por agentes de software de acordo com o interesse de seus proprietários [He 2003]. Simplificando, um serviço é uma tarefa repetível dentro de um processo de negócio, ou seja, tarefas são serviços e processos são composições de serviços [High Jr. 2005].

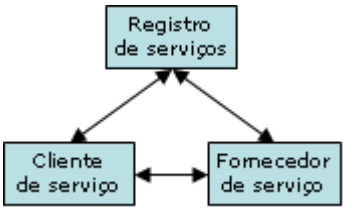


**Figura 2.2 – Arquitetura orientada a serviços**

Ao contrário do que aparenta, arquiteturas orientadas a serviço não são uma tecnologia nova. Em uma abordagem anterior, os serviços eram implementados através de DCOM e CORBA. Pela abordagem atual, todavia, os serviços são implementados através de *web services*. No entanto, essa não é uma obrigatoriedade e todas as formas de implementação de serviços podem ser utilizadas de forma conjunta.

### 2.3.1 *Web services*

Quando os *web services* surgiram, não havia uma definição universalmente aceita do seu significado. Porém, de modo geral, um *web service* é a revelação de uma tarefa de um processo de negócio através de uma rede de comunicação, sendo que a conotação mais comum diz respeito a tráfego baseado em XML movendo-se por uma rede pública (Internet) através do protocolo HTTP. De acordo com [Graham 2001], um *web service* é um componente de *software* independente de implementação e de plataforma que pode ser:

- Descrito usando uma linguagem de descrição de serviços;
- Publicado em um registrador de serviços

- Descoberto através de um mecanismo padrão, em tempo de execução ou de projeto;
- Invocado através de uma API declarada, usualmente sobre uma rede de comunicação.
- Composto com outros serviços.

Para [Cerami 2002], *web service* é qualquer serviço que está disponível através da Internet, utiliza um sistema de transferência de mensagens padrão XML e não está amarrado a nenhum sistema operacional ou linguagem de programação. Ou, simplesmente, pode-se considerar *web services* como componentes de programas reutilizáveis, que utilizam XML como um padrão.

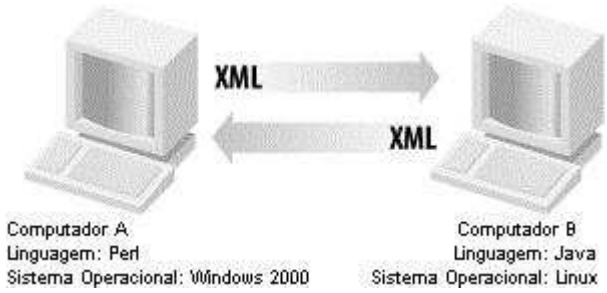


**Figura 2.3 – *Web service* básico**

Além do mais, *web services* são úteis internamente para uma organização, como um mecanismo para encapsulamento e exposição da lógica do negócio inerente a sistemas legados. Novas aplicações podem então reutilizar essa interface *web service* para alavancar a complexa lógica de negócio que foi refinada através dos anos nesses sistemas legados. Isso permite a reutilização dos sistemas no nível lógico, sem considerar a configuração física [Jorgensen 2002].


**Figura 2.4 – Papéis dentro da arquitetura de *web services***

De acordo com o que foi apresentado acima é clara a similaridade entre as definições de *web services* e SOA, inclusive com ambas confundindo-se em certos aspectos. Isso ocorre, de fato, e é absolutamente normal, visto que aplicações SOA são, na prática, composições de *web services*.

Os três principais papéis dentro da arquitetura de *web services*, apresentados pela figura 2.4 são:

- Provedor de serviços. É o provedor do *web service*. Implementa o serviço e o deixa disponível na Internet.
- Solicitante de serviços: É qualquer consumidor do *web service*. Utiliza um *web service* existente abrindo uma conexão de rede e enviando uma requisição XML.
- Registrador de serviços. É um diretório de serviços logicamente centralizado. O registrador fornece um local central onde desenvolvedores podem publicar novos serviços ou encontrar serviços existentes. Serve como órgão centralizador para as companhias e seus serviços.

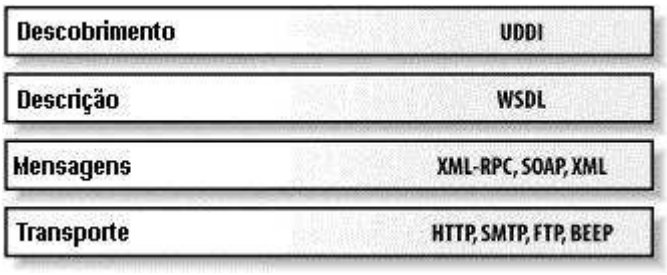


**Figura 2.5 – Pilha de protocolos da arquitetura de *web services***

A figura 2.5 ilustra a pilha de protocolos envolvida com a arquitetura de *web services*, sendo que suas camadas são explicadas abaixo [Cerami 2002]:

- Serviço de transporte. Camada responsável por transportar mensagens entre aplicações. Atualmente essa camada inclui os seguintes protocolos: HTTP, SMTP, FTP e BEEP.
- Transferência de mensagens. Camada responsável por codificar mensagens em um formato XML comum para que essas mensagens possam ser entendidas nas duas pontas. Atualmente essa camada inclui os protocolos XML-RPC e SOAP.
- Serviço de descrição. Camada responsável por descrever a interface pública para um *web service* específico. Atualmente essa camada inclui o protocolo WSDL.

- Serviço de descobrimento. Camada responsável por centralizar serviços em um registrador comum e fornecer funcionalidades para facilitar a publicação e descoberta de *web services*. Atualmente essa camada inclui o protocolo UDDI.

A empolgação que ocorre atualmente com SOA também ocorreu com o surgimento dos *web services*. *Web services* fornecem potenciais quase ilimitados. Qualquer programa pode ser mapeado para *web service* e *web services* podem ser mapeados por qualquer programa. A transformação de dados de e para XML é essencial, mas XML é flexível o suficiente para acomodar qualquer tipo de dado e estrutura e até mesmo para criar novos tipos, se necessário. Quando todos os programas e sistemas estiverem finalmente habilitados para *web service*, o mundo da computação distribuída será muito diferente do que é hoje [Newcomer 2002].

### 2.3.2 XML

O XML [Bray 2006] é a fundamentação básica sobre a qual SOA e *web services* são construídos. Fornece uma linguagem de definição de dados e de como processá-los [Newcomer 2002] e representa uma família de especificações relacionadas, publicadas e mantidas pelo W3C.

Tecnicamente, o XML é um sistema para definição de linguagens de marcação completas, incluindo a habilidade de estender as já existentes. Na prática, no entanto, XML não é utilizado como uma linguagem de marcação, da mesma forma que o HTML. Ao invés disso, é utilizado para representar dados, através do mecanismo de criação das próprias *tags*, além de possuir um conjunto de padrões para assegurar a sua interoperabilidade, estabilidade e longevidade. Dessa forma, XML não é uma linguagem de programação e nem uma linguagem de marcação, mas uma linguagem para representar dados de forma estruturada.

O XML surgiu para suprir a carência deixada pelo HTML no desenvolvimento de aplicações avançadas para a Internet, principalmente no que tange àquelas aplicações que precisam manipular um grande volume de dados. Um dos objetivos por trás dessa linguagem é possibilitar a transferência e manipulação de dados através da Internet de modo consistente, de tal forma que qualquer tipo de aplicação, independentemente da plataforma, sistema operacional, ou linguagem em que foi construída consiga manuseá-los.

Os objetivos projetados para o XML, na sua origem, foram [Bray 2006]:

1. XML deve ser diretamente usável através da Internet;

2. XML deve suportar uma ampla variedade de aplicações;
3. XML deve ser compatível com SGML;
4. Escrever programas que processem documentos XML não deve ser complicado;
5. O número de características opcionais em XML deve ser mantido no mínimo, idealmente zero;
6. Documentos XML deveriam ser legíveis para humanos e razoavelmente limpos;
7. O projeto XML deveria ser preparado rapidamente;
8. O projeto de XML deve ser formal e conciso;
9. Documentos XML devem ser simples de criar;
10. O tamanho dos documentos XML é de mínima importância.

```
<?xml version="1.0" ?>
<note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
</note>
```

**Figura 2.6 – Exemplo de código XML**

O surgimento do XML revolucionou o mercado de *e-business*, habilitando os sistemas computacionais comerciais a comunicarem-se mais facilmente entre si. Outras vantagens do XML são sua adaptabilidade, sua extensibilidade e a simplicidade de seu padrão, facilitando o uso tanto por computadores como por pessoas. Esses fatores contribuíram para a rápida adoção de XML em diversas áreas, ferramentas e aplicações, tornando-o, atualmente, um dos principais padrões de comunicação entre aplicações de *software* através da Internet.

### 2.3.3 WSDL

WSDL [Christensen 2001] é uma estrutura XML que define um *framework* extensível para descrever interfaces *web services*. Inicialmente foi desenvolvida pela Microsoft, IBM e Ariba, sendo posteriormente submetida para a W3C por vinte e cinco companhias. A sua especificação foi criada para descrever e publicar os formatos e protocolos de um *web service* em uma forma padrão. Esses padrões de interface são necessários para assegurar que não tenham que ser criadas interações especiais com cada servidor na *web*, como é feito hoje quando se usa o acesso por URL de um *browser* [Newcomer 2002].

WSDL especifica uma interface pública para um *web service*. Essa interface pública pode incluir informação de todas as funções disponíveis em geral, informação dos tipos de dados, informação de conexão sobre protocolo específico de transporte que será usado e informação de endereço para localizar o serviço especificado. WSDL não é necessariamente amarrada a um sistema específico de mensagens XML, mas inclui extensões embutidas para descrever serviços SOAP.

```
  <binding name="CalculadoraWSPortBinding" type="tns:CalculadoraWS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="divide">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
```

**Figura 2.7 – Trecho de código WSDL**

WSDL representa a camada de descrição de serviços na pilha de protocolos *web services*, ou seja, é utilizado para descrever as capacidades e a interoperabilidade requerida por um *web service*. Por outro ponto de vista, WSDL descreve um contrato entre o requerente e o provedor do serviço [Cerami 2002].

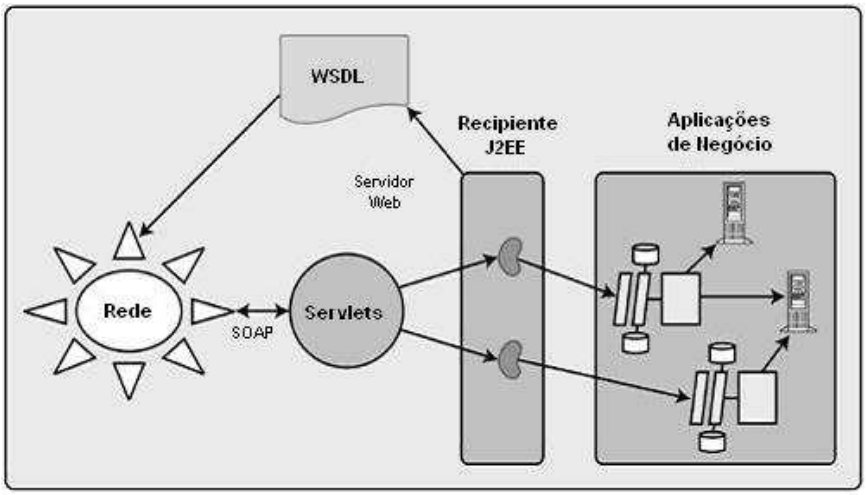


**Figura 2.8 – Aplicações podem usar WSDL para publicar *web services***

Uma descrição de serviço WSDL descreve o ponto do contato de um serviço, também conhecido como *service endpoint*. Ele estabelece a localização física do serviço e fornece uma definição formal da sua interface, de forma que os programas que desejam comunicar-se com

os *web services* possam saber exatamente como estruturar as mensagens de requisição necessárias. Basicamente, uma descrição de serviço WSDL pode ser separada em duas partes [Erl 2005]:

- Descrição abstrata. Fornece uma definição abstrata dos detalhes e regras de comunicação entre duas aplicações – como um *web service* trabalhará com outra aplicação. Estabelece as características de interface do *web service* sem qualquer referência às tecnologias usadas para hospedar ou transmitir as mensagens. Separando essa informação, a integridade da descrição do serviço pode ser preservada independentemente das alterações que possam ocorrer na plataforma tecnológica.
- Descrição concreta. Define o protocolo de transporte físico para possibilitar que a interface abstrata do *web service* possa comunicar-se.

Segundo [Cerami 2002], usando WSDL, um cliente pode localizar um *web service* e invocar qualquer uma de suas funções disponíveis. Com ferramentas WSDL, esse processo também pode ser automatizado, habilitando aplicações a facilmente integrarem novos serviços com pouco ou nenhum código manual. Dessa forma, WSDL representa uma pedra fundamental da arquitetura *web service*, porque fornece uma linguagem simples para descrever serviços e uma plataforma para integrar automaticamente esses serviços.

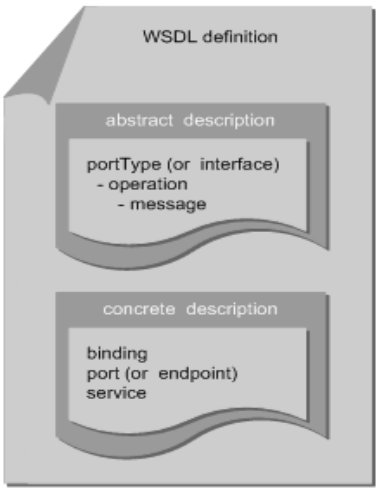


**Figura 2.9 – Definição do WSDL**

### 2.3.4 SOAP

O padrão SOAP não é especificamente uma linguagem, tão pouco um componente, mas um protocolo de mensagens bastante simples que define regras para sistemas diferentes

interagirem, trocando informações tipadas e estruturadas através da Internet. [Clements 2002] afirma que SOAP possibilita a comunicação de objetos (ou código) de qualquer tipo, sobre qualquer plataforma e em qualquer linguagem, inclusive permitindo que o controle dos processos seja passado entre duas aplicações conectadas em rede.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <soap:Header>
      <h:identity
xmlns:h=”http://www.wrox.com/header”>author@wrox.com</h:identity>
      </soap:Header>
      <soap:Body>
      <m:GetStockQuote xmlns:m="http://www.wrox.com/getstockquote/">
      <m:ticker>WROX</m:ticker>
      </m:GetStockQuote>
      </soap:Body>
</soap:Envelope>
```

**Figura 2.10 – Exemplo de mensagem SOAP**

Utilizando SOAP é possível construir aplicações invocando remotamente métodos em objetos. Esse protocolo remove os requisitos que dois sistemas devam rodar sobre a mesma plataforma ou serem escritos na mesma linguagem de programação. Em vez de invocar métodos através de protocolos binários proprietários, SOAP utiliza XML para fazer chamadas de métodos. Do ponto de vista dos *web services,* SOAP pode ser implementado, tanto como cliente, como servidor.


**Figura 2.11 – Anatomia de um envelope SOAP**

Segundo [Snell 2001], uma mensagem SOAP consiste em um envelope contendo um cabeçalho opcional e um corpo necessário, como mostra a figura 2.11. O cabeçalho contém blocos de informações relevantes de como a mensagem deve ser processada. Isso inclui

configurações de roteamento e entrega, declarações de autenticação e ou autorização e contexto de transações. O corpo contém a mensagem que efetivamente será enviada e processada. No corpo da mensagem pode constar qualquer coisa, desde que possa ser expressa através do formato XML.

[Hendricks 2002] cita as vantagens do SOAP em relação a outros sistemas distribuídos:

- Transpõe *firewalls* com facilidade por usar o protocolo HTTP;
- Estrutura dados em XML;
- Pode ser usado, potencialmente, em combinação com vários protocolos de transporte como HTTP, SMTP e JMS;
- Mapeia satisfatoriamente para o padrão de solicitação e resposta HTTP e do HTTP *Extension Framework*;
- É razoavelmente superficial como um protocolo;
- É suportado por vários grandes fornecedores incluindo a Microsoft, a IBM e a Sun.

SOAP também possui algumas desvantagens em relação a outros sistemas distribuídos, também apresentadas pelo mesmo autor:

- Falta de interoperabilidade entre *toolkits* do SOAP;
- Mecanismo de segurança imaturo;
- Inexiste a garantia de entrega da mensagem;
- Não existe publicação, nem assinatura.

### 2.3.5 UDDI

Depois que um *web service* está configurado, deve ser possível encontrar e utilizar o serviço. Essa é a proposta do “cartório” UDDI, estabelecido por um consórcio de indústrias para criar e implementar um diretório de *web services*. Esse “cartório” aceita informações descrevendo negócios, incluindo os *web services*, que oferecem e permitem que partes interessadas executem pesquisas e *downloads online* das informações [Newcomer 2002]. UDDI descreve a camada de descobrimento na pilha de protocolos *web services*. Originalmente foi criado pela Microsoft, IBM e Ariba e representa uma especificação técnica para a publicação e descobrimento de negócios e *web services* [Cerami 2002].

As especificações do UDDI definem uma maneira de publicar e descobrir informações sobre *web services*. Neste caso, o termo *web service* descreve funcionalidades de negócio específicas exibidas por uma companhia, geralmente através de uma conexão Internet, com o propósito de fornecer uma forma para que outra companhia ou programa de *software* utilize o serviço [UDDI 2000].

UDDI é um diretório *web* que permite que aplicações anunciem sua disponibilidade e seus serviços para outras aplicações. *Web services* registrados em um diretório UDDI podem ser descobertos por outras aplicações *web* que estejam procurando por serviços específicos, mercadorias ou parceiros.

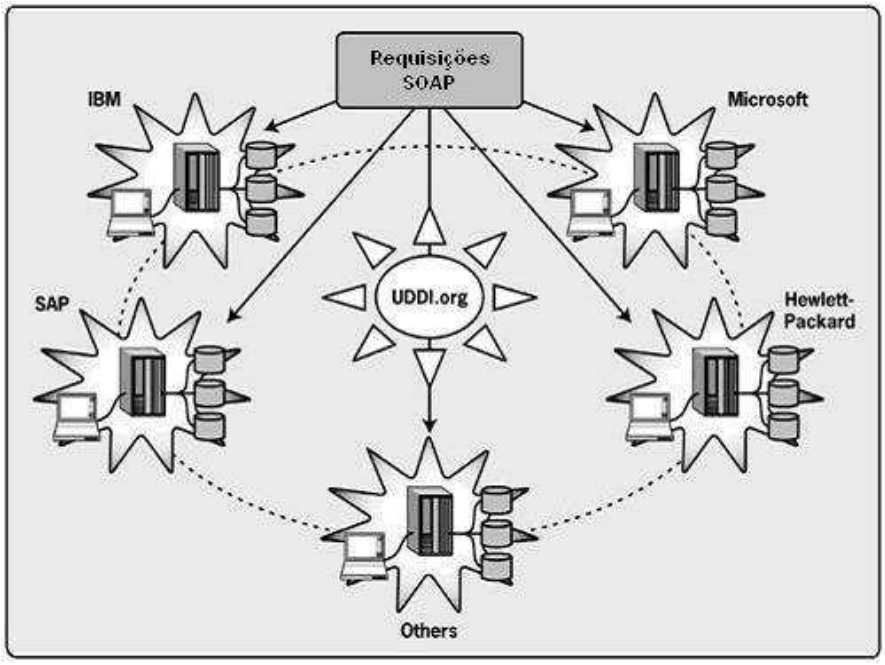


**Figura 2.12 – Múltiplos vendedores hospedam os serviços UDDI públicos**

Usualmente, UDDI é definido como sendo as páginas amarelas dos *web services*. Da mesma forma que as tradicionais páginas amarelas, podem-se encontrar companhias que ofereçam serviços que estejam sendo procurados, obter informação sobre o serviço oferecido e contatar alguém para maiores informações. Ampliando um pouco essa definição, os dados contidos dentro do UDDI podem ser divididos em três categorias principais [Cerami 2002]:

- Páginas brancas *(White pages)*. Essa categoria inclui informações gerais sobre uma companhia específica. Por exemplo, nome, descrição e endereço do negócio.
- Páginas amarelas *(Yellow pages)*. Essa categoria inclui classificação geral de dados para a companhia ou para o serviço oferecido. Por exemplo, esses dados

podem incluir indústria, produto ou códigos geográficos baseados em taxonomias padrões.

- Páginas verdes *(Green pages)*. Essa categoria inclui informação técnica sobre um *web service*. Por exemplo, um apontador para uma especificação externa e um endereço para invocar o *web service*.

### 2.3.6 BPEL

Entre os diversos componentes de uma arquitetura orientada a serviços, pode-se destacar como principal os serviços (*web services*). SOA, no entanto, propõe algo mais do que a utilização simples e unitária desses serviços. SOA propõe aplicações compostas a partir desses serviços individuais. E, para a criação dessas aplicações, linguagens de composição de serviços estão sendo definidas, sendo que a que está tornando-se o padrão é BPEL.

```
   <sequence name="Main">
      <receive name="RecebeParametros" createInstance="yes"
partnerLink="LinkInterface" operation="interfaceCalculadoraOperation"
portType="ns1:interfaceCalculadoraPortType"
variable="InterfaceCalculadoraOperationIn1"/>
      <if name="If">

<condition>contains($InterfaceCalculadoraOperationIn1.body/ns0:operacao,
'soma')</condition>
         <sequence name="Soma">
            <assign name="AtribuiParametrosSoma">
         <copy>
            <from>$InterfaceCalculadoraOperationIn1.body/ns0:valor1</from>
            <to>$SomaIn1.parameters/valor1</to>
         </copy>
         <copy>
            <from>$InterfaceCalculadoraOperationIn1.body/ns0:valor2</from>
            <to>$SomaIn1.parameters/valor2</to>
         </copy>
      </assign>
            <invoke name="ChamaSoma" partnerLink="LinkCalculadora"
operation="soma" portType="ns2:CalculadoraWS" inputVariable="SomaIn1"
outputVariable="SomaOut1"/>
            <assign name="AtribuiResultadosSoma">
         <copy>
            <from>$SomaOut1.parameters/return</from>
            <to variable="InterfaceCalculadoraOperationOut1" part="body"/>
         </copy>
      </assign>
         </sequence>
```

**Figura 2.13 – Trecho de código BPEL**

O projeto da linguagem foi implementado por desenvolvedores da BEA *Systems*, IBM e Microsoft e iniciou-se com o nome BPEL4WS [Weerawarana 2002], em substituição às

linguagens IBM WSFL e Microsoft XLANG. Atualmente é conhecido como WS-BPEL ou simplesmente BPEL, sendo que a versão atual, WS-BPEL 2.0, foi especificamente projetada para orquestrar conversações de longa duração entre *web services*.

BPEL é uma linguagem de programação para especificação de processos de negócios que envolvam *web services*. É uma linguagem baseada em XML, projetada para habilitar o compartilhamento de tarefas para um ambiente de computação distribuída ou computação em *grid*, mesmo através de múltiplas organizações, utilizando uma combinação de *web services*.

BPEL não tenta ser uma linguagem de programação de propósito geral, mas sim se assume que será combinada com outras linguagens, que são usadas para programar as funções de negócio (*programming in the small*), enquanto o BPEL implementa a lógica do processo de negócio (*programming in the large*) [Blow 2004]. Por exemplo, em um ambiente Java EE os serviços seriam programados como Java *web services*, seriam referenciados na aplicação composta e manipulados através de BPEL.

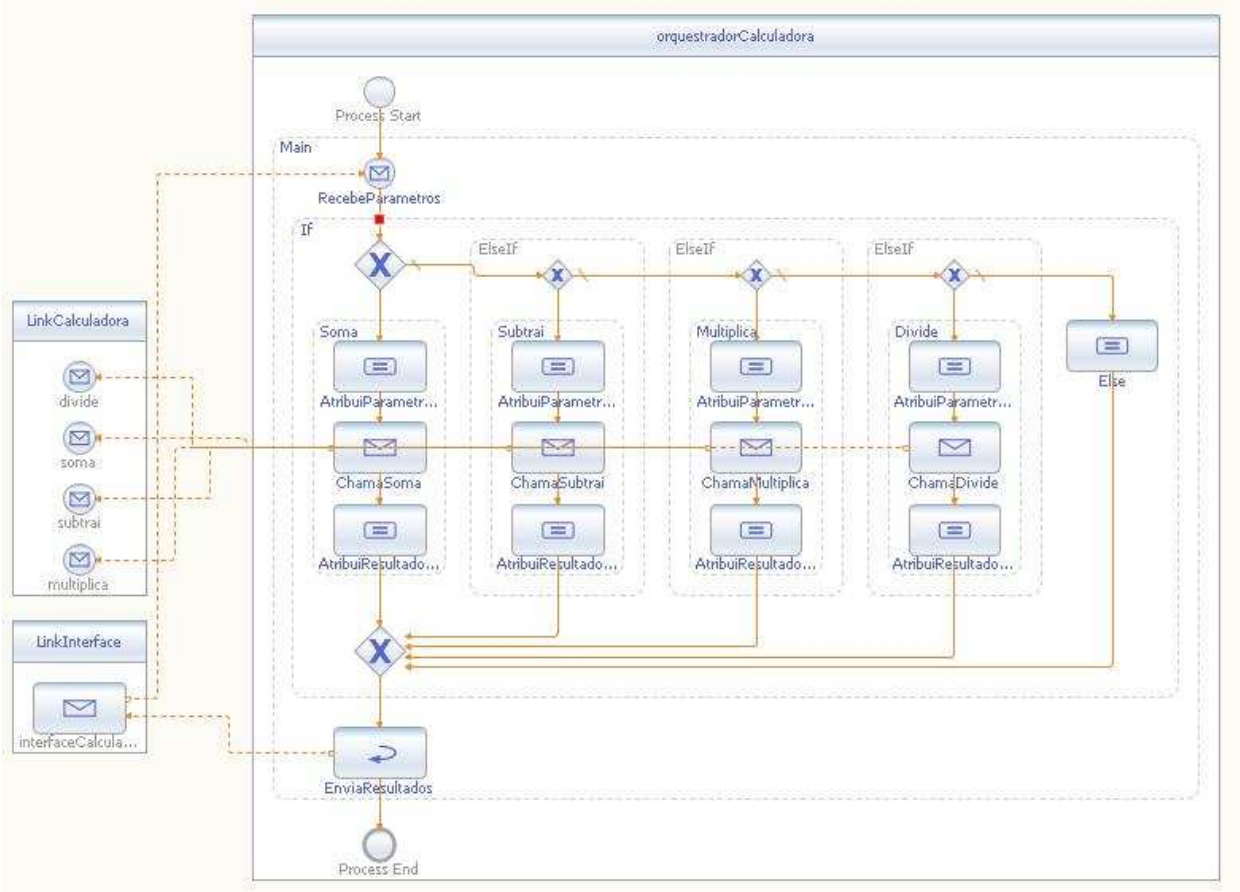


**Figura 2.14 – Orquestração BPEL**

Usando BPEL, um programador formalmente descreve um processo de negócio que terá lugar através da *web* de tal forma que qualquer entidade cooperante possa executar um ou mais passos no processo da mesma forma. Por exemplo, num processo *supply chain*, um programa BPEL pode descrever um protocolo de negócio que formaliza de quais pedaços da informação um pedido de produto consiste e quais exceções devem ser suportadas. No entanto, um programa BPEL não deveria especificar como um dado *web service* deve processar o pedido internamente.

BPEL suporta dois tipos de cenários de uso: implementando processos de negócios executáveis e descrevendo processos abstratos não-executáveis. Como linguagem de implementação de processos executáveis, o papel do BPEL é definir um novo *web service* através da composição de um conjunto de serviços. A interface do serviço composto é descrita como uma coleção de WSDL *portTypes*, como qualquer outro *web service* [Weerawarana 2002]. Portanto, na prática, quando um processo BPEL é definido, efetivamente é definido um novo *web service* que é uma composição de serviços existentes. Já a utilização de BPEL como linguagem para descrever processos abstratos não-executáveis pode fazê-lo funcionar como linguagem de coreografia.

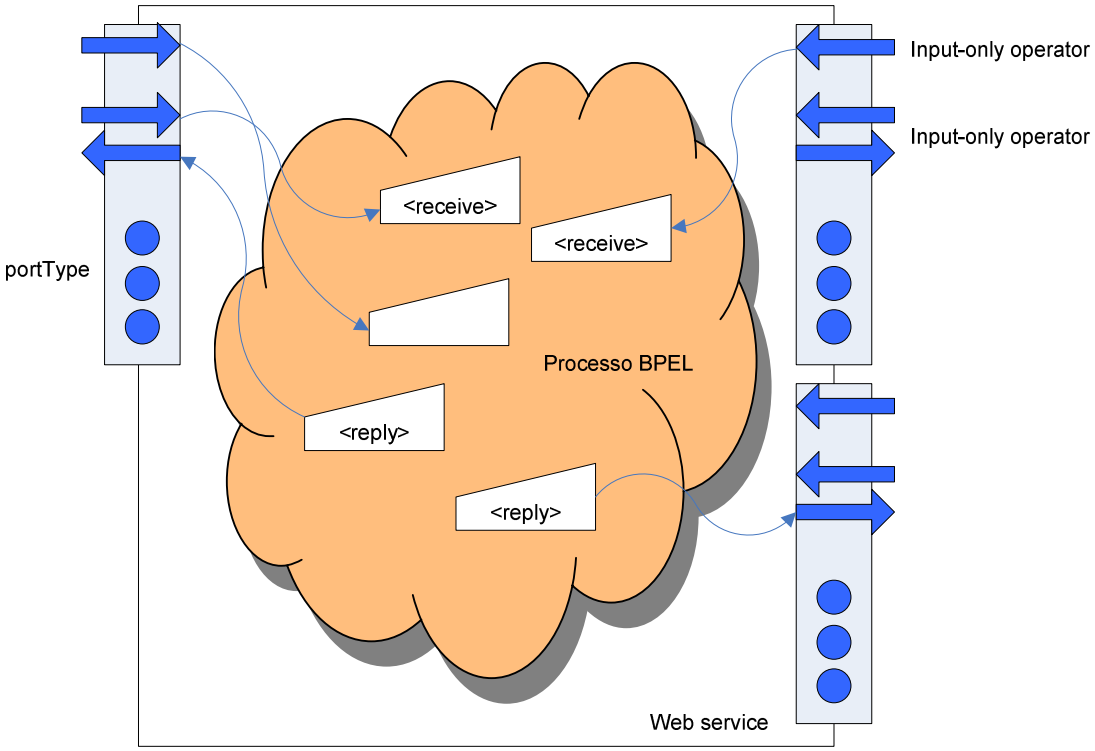


**Figura 2.15 – Comunicação entre o processo BPEL e os *web services***

Dentro da pilha de protocolos utilizados em SOA, BPEL e as outras linguagens de composição de serviços estão no nível mais alto, como ilustra a figura 2.16.

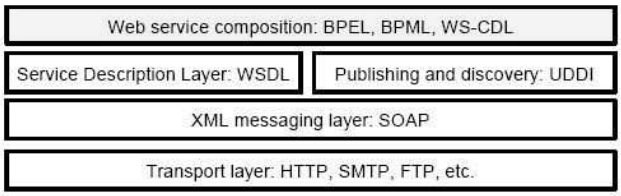


**Figura 2.16 – Nova camada na pilha de protocolos**

## 2.4 Plataformas de desenvolvimento

As principais plataformas de desenvolvimento de *software*, Sun Java e Microsoft .NET, também possuem ambientes para o desenvolvimento de aplicações móveis e ubíquas, respectivamente Java *Platform, Micro Edition* (Java ME) e .NET *Compact Framework*. Além dessas, uma nova plataforma, especificamente para ambientes móveis e ubíqüos, foi lançada no final de 2007: Android, da Open Handset Alliance, liderada pelo Google. Essas três plataformas são mais bem explicadas nas subseções a seguir.

### 2.4.1 .Net *Compact Framework*

A plataforma da Microsoft para o desenvolvimento de aplicações para dispositivos móveis é chamada de. NET *Compact Framework* [MSDN 2008]. O .NET *Compact Framework* é um componente integral dos dispositivos que possuem os sistemas operacionais *Windows Mobile* e *Windows Embedded* CE, que habilitam a construção e execução de aplicações e a utilização de *web services*.

O .NET *Compact Framework* suporta as linguagens *Visual Basic* e *Visual* C#. e inclui uma máquina virtual otimizada chamada CLR. É, portanto, projetado para obter um desempenho ótimo mesmo respeitando as restrições dos recursos dos dispositivos limitados. Além de classes criadas exclusivamente para a plataforma, possui um subconjunto de bibliotecas de classes do .NET *Framework* completo, o que dá suporte a características tais como WCF e *Windows Forms*.

Os principais aspectos arquiteturais da plataforma, ilustrada na figura 2.17, são os seguintes:

- *Windows* CE. Sistema operacional base da arquitetura. Fornece as funcionalidades centrais e diversas características específicas dos dispositivos, além de telas, gráficos, desenhos e *web services*, que foram reescritos para eficientemente rodarem sobre dispositivos. A interoperabilidade com a plataforma .NET *Compact Framework* inclui compatibilidade com a segurança nativa, integração completa com programas de configuração nativos e interoperabilidade com código nativo usando COM.
- CLR. Máquina virtual da plataforma, reescrita para permitir que recursos restritos executem sobre memória limitada e para eficientemente usar a energia da bateria.
- *Framework*. É um subconjunto do .NET *Framework* e também contém características exclusivamente projetadas para a plataforma .NET *Compact Framework*. Fornece recursos que facilitam que desenvolvedores do *framework* nativo desenvolvam aplicações para o .NET *Compact Framework* e vice-versa.
- *Visual Studio*. O desenvolvimento de aplicações para dispositivos móveis através do IDE que acompanha a plataforma é tão fácil quanto o desenvolvimento para aplicações *desktop*. O IDE inclui um conjunto de tipos de projetos e emuladores que suportam o desenvolvimento para *Pocket* PC, *smartphone* e *Windows* CE embarcado.

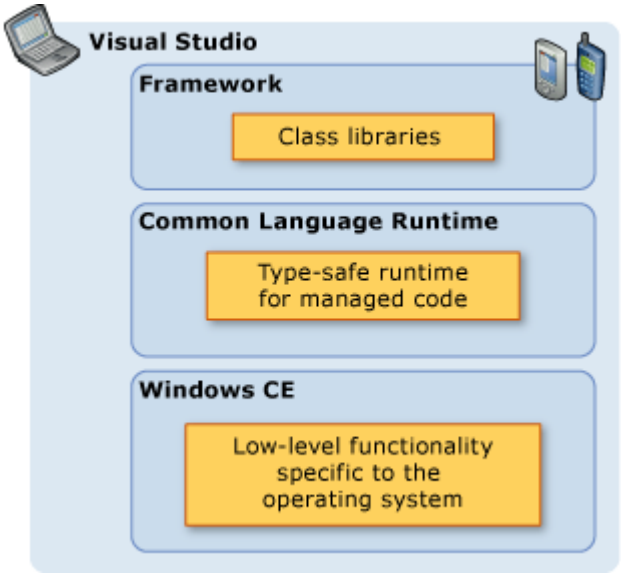


**Figura 2.17 – Arquitetura .Net *Compact Framework* [MSDN 2008]**

A plataforma .NET *Compact Framework* fornece suporte a *web services* embutido que inclui o suporte às seguintes funcionalidades:

- Protocolos baseados em HTTP;
- Autenticação NTLM;
- Conteúdo XML codificado por SOAP: esse suporte inclui a passagem de conjuntos de dados ADO .NET;
- Métodos de requisições e respostas *web* que podem enviar mensagens HTTP SOAP e receber mensagens SOAP em resposta;
- Bibliotecas e métodos SOAP que podem serializar e de-serializar chamadas de métodos e objetos arbitrários em e de mensagens XML SOAP.

### 2.4.2 Java *Platform, Micro Edition* (Java ME)

Java ME é a plataforma da Sun para o desenvolvimento de aplicações para computação móvel e ubíqua. Conforme a própria fabricante, Java ME é a mais ubíqua plataforma de aplicações para dispositivos móveis. Ainda segundo ela, Java ME fornece um ambiente robusto e flexível para a execução de aplicações embarcadas em dispositivos, móveis ou não, tais como telefones celulares, assistentes digitais pessoais (PDA's), *set-top-boxes* de televisão e impressoras. Java ME inclui interfaces para o usuário flexíveis, segurança robusta, protocolos de redes incluídos e suporte para aplicações em rede e *offline* que podem ser baixadas dinamicamente. Aplicações baseadas em Java ME são portáveis através de muitos dispositivos, já considerando as capacidades nativas dos dispositivos [JME 2008].

A plataforma Java ME é uma coleção de tecnologias e especificações que podem ser combinadas para a construção de um ambiente de execução Java. A construção de uma aplicação Java ME leva em conta para qual dispositivo ela está sendo construída e é baseada em três elementos:

- Uma configuração, que fornece as bibliotecas mais básicas e capacidades de uma máquina virtual para uma grande quantidade de dispositivos. A configuração de pequenos dispositivos móveis é chamada de CLDC e a configuração de dispositivos mais capazes como *smartphones* e *set-top-boxes* é chamada de CDC.
- Um perfil, que é um conjunto de API's que suportam uma quantidade menor de dispositivos, chamado de MIDP.
- Um pacote opcional, que é um conjunto de API's específicas de uma tecnologia.

A figura 2.18 representa uma visão geral dos componentes da tecnologia Java ME e como eles se relacionam com as outras tecnologias Java.

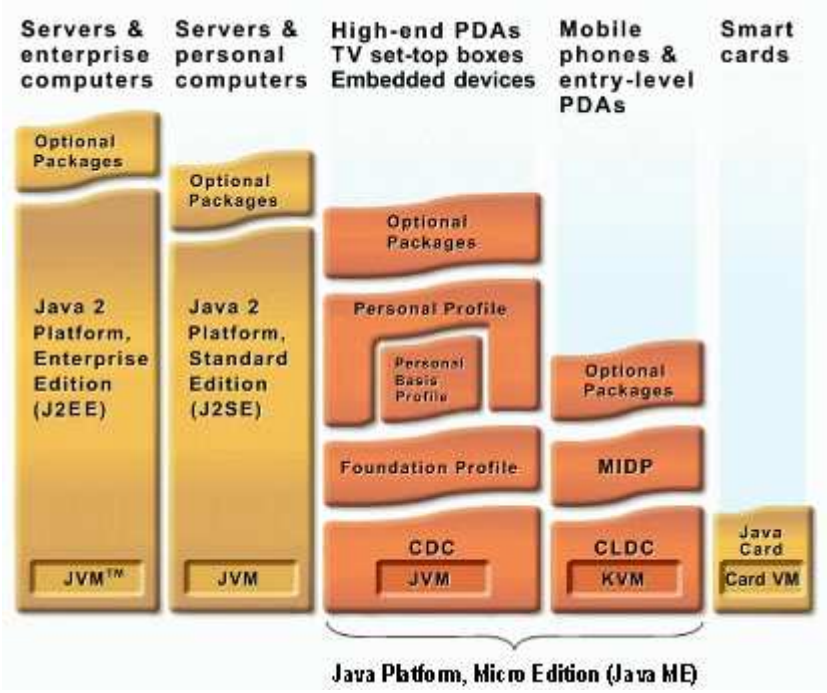


**Figura 2.18 – Arquitetura Java ME [JME 2008]**

A solução proposta pela plataforma Java ME para trabalhar com *web services* é chamada de Java ME *Web Services Specification*, ou JSR 172. Essa especificação define um subconjunto de API's que a JAXP define para a manipulação básica de dados estruturados XML. A especificação ainda identifica o subconjunto de API's e convenções que a JAX-RPC usa para permitir chamadas RPC baseadas em XML para a plataforma JME. Ou seja, JSR 172 especifica:

- API's para processamento do XML, a partir de um subconjunto de JAXP;
- Acesso a *web services* baseados em RPC, a partir de um subconjunto de JAX-RPC.

Juntos, os subconjuntos JAXP e JAX-RPC habilitam que aplicações JME forneçam capacidades básicas de processamento de XML e um modelo de desenvolvimento consistente para todos os clientes JME comunicarem-se com *web services*. As duas novas tecnologias também garantem a interoperabilidade de clientes JME com *web services* e habilitam que desenvolvedores reusem componentes *web services*, permitindo que o mesmo serviço seja implantado sobre muitos tipos de clientes e plataformas Java.

JSR 172 também fornece um compilador de *stub* que cria todo o código que aplicações JME necessitam para executar simples chamadas programáticas para um *web service* existente. A especificação também permite que *stubs* sejam independentes da implementação,

de forma que essas aplicações possam ser dinamicamente fornecidas para qualquer plataforma JME que suporte essa tecnologia. Resumindo, aplicações JSR 172 são portáveis.

A especificação JSR 172, no entanto, contém algumas limitações notáveis, a maioria delas devidas aos requisitos do WS-I *Basic Profile* [Ballinger 2004]. A conformidade com esse perfil promove a interoperabilidade, mas também previne a utilização de métodos alternativos [Tergujeff 2007].

Outra abordagem dentro da plataforma Java ME é a utilização das bibliotecas kXML [Haustein 2005] e kSOAP [Haustein 2006]. Essas bibliotecas, de código-aberto, estão disponíveis há algum tempo para ambientes Java e são indicadas principalmente quando JSR 172 não está disponível. Para justificar a utilização dessas bibliotecas, principalmente kSOAP, [Tergujeff 2007] afirma que a adequação a dispositivos sem suporte a JSR 172 e a garantia de habilidade para acessar serviços não conformes ao WS-I torna kSOAP ainda relevante atualmente.

### 2.4.3 Android

Android é uma plataforma de *software* para dispositivos móveis baseada em Linux, que permite o desenvolvimento de aplicações em Java e que utiliza bibliotecas desenvolvidas pelo Google [Android 2008]. Embora seja um projeto da Open Handset Alliance, o Android é usualmente ligado ao Google, um dos fundadores dessa aliança e o seu principal participante, tanto que muitas vezes é chamado de Google Android.

Efetivamente, o Android é uma junção de diversos componentes que já existiam e de novos componentes criados especificamente para a plataforma. Por isso, é apresentado também como uma pilha de *softwares* que inclui sistema operacional, *middleware* e aplicações-chave, conforme ilustra a figura 2.19.

Na base da pilha de *softwares* do Android encontra-se o sistema operacional, baseado em Linux, juntamente com os componentes e *drivers* necessários para a correta utilização dos componentes de *hardware* dos dispositivos móveis. Na camada central da plataforma está o *middleware*, que se divide em três partes: Android *Runtime*, onde se encontram as bibliotecas de núcleo e a máquina virtual; bibliotecas gerais, utilizadas para o desenvolvimento das aplicações; *framework* de aplicação, onde se encontram os componentes de mais alto nível que são utilizados para a construção das aplicações. Na camada superior da pilha encontram-se as aplicações-chave disponibilizadas com o Android, tais como Contatos, Telefone e

*Browser*, entre outros, que podem ser utilizadas para o desenvolvimento de outras aplicações, que também são inseridas nessa camada.

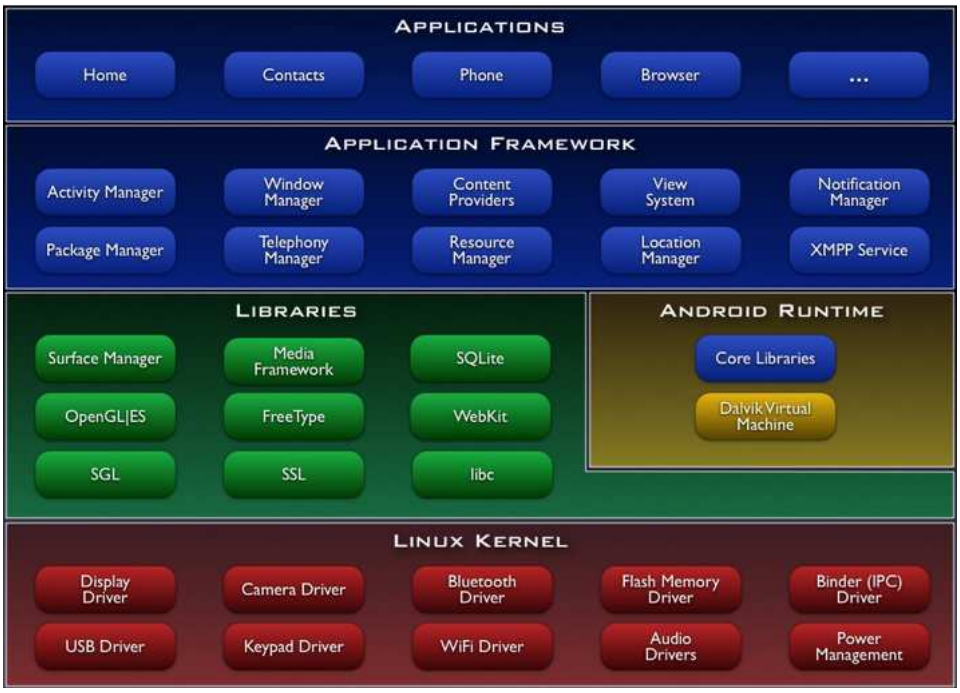


**Figura 2.19 – Arquitetura Android [Android 2008]**

O Android é uma plataforma criada especificamente para dispositivos móveis, o que pode ser facilmente percebido através da análise de suas características já embarcadas:

- Layouts dos dispositivos: adaptável para telas com *layouts* tradicionais e com *layouts* maiores, sensíveis ao toque (*touch screen*);
- Conectividade: principais padrões do mercado, tais como GSM, CDMA, Bluetooth, EDGE, EV-DO, 3G e Wi-Fi;
- Mídias suportadas: principais mídias do mercado, tais como MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF;
- Troca de mensagens: SMS e MMS;
- Máquina virtual otimizada para dispositivos móveis: *Dalvik Virtual Machine*;
- Suporte a *hardwares* adicionais: câmera, GPS, bússola, acelerômetro;
- Armazenamento: banco de dados SQLite;
- *Web browser*: WebKit.

Pelo fato do Android ter sido construído exclusivamente para o desenvolvimento de aplicações para dispositivos móveis, algumas características foram definidas tendo esse

objetivo em mente, como os quatro tipos de blocos de construção, por exemplo. Aplicações Android são construídas através de uma combinação desses blocos:

- Atividade (*Activity*). É o bloco de construção mais comum em aplicações Android. Normalmente é uma simples tela da aplicação, sendo que deve mostrar a interface com o usuário e responder a eventos. Por exemplo, uma aplicação de mensagens de texto com duas telas possui 1 *activity* para a tela que mostra lista de contatos e 1 *activity* para a tela para escrever a mensagem para o contato escolhido.
- Intenção (*Intent*). O Android usa uma classe especial chamada *Intent* para mover de uma tela para outra. Um *intent* descreve a intenção da aplicação, ou seja, o que a aplicação quer fazer. Por exemplo, a intenção de chamar outra atividade. Usa-se um Receptor de Intenção (*Intent Receiver*) para executar uma reação a um evento externo. Por exemplo, quando o telefone toca, quando a rede de dados fica disponível, quando for determinado horário. A aplicação não precisa estar rodando para que o *intent receiver* seja chamado. Se necessário, o sistema inicia a aplicação caso o *intent receiver* tenha sido disparado.
- Serviço (*Service*). Um *service* é código de longa duração que é executado sem a existência de uma tela. Por exemplo, em um *media player* tocando sons de uma lista de músicas, a aplicação *media player* possui uma ou mais *activities* que permitem que o usuário escolha as músicas e inicie ou pare o tocador. No entanto, a ação de tocar música não deve ser implementada como *activity*, mas sim como *service*, porque deve permanecer tocando as músicas mesmo quando o usuário sair da tela e entrar em outra aplicação;
- Fornecedor de Conteúdo (*Content Provider*). Um *content provider* é útil para disponibilizar dados de uma aplicação para as outras aplicações. Aplicações podem armazenar dados em arquivos, banco de dados SQLite ou algum outro mecanismo. Por exemplo, a aplicação Contatos utiliza o bloco de construção *content provider*.

Até o momento, o Android não possui bibliotecas específicas para trabalhar com *web services*. Dessa forma, o desenvolvedor deve implementar o seu próprio código para habilitar suas aplicações a trabalharem com *web services*. Como a linguagem de programação do Android é Java, para contornar essa dificuldade pode-se utilizar as bibliotecas kXML e kSOAP.

# 3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados alguns trabalhos relacionados com a proposta de dissertação que é descrita no capítulo 4 e que comprovam a relevância do assunto escolhido. Esses trabalhos ilustram desde o consumo e do fornecimento de *web services* em dispositivos móveis, até motores de execução de composições de *web services*, tema central desta proposta.

## 3.1 Arquitetura *Mobile* SOA

Um exemplo de arquitetura de acesso a *web services* por dispositivos móveis é apresentado por [Tergujeff 2007]. Em um sentido técnico, a demonstração consiste de uma descrição de serviço (WSDL), uma implementação de serviço e três aplicações clientes, conforme ilustrado pela figura 3.1. JSR 172 é usado para consumo do serviço em dois clientes móveis de funcionalidades idênticas: um para dispositivos com suporte nativos para JSR 172 e outro para dispositivos sem essa API. Para comparação, uma aplicação cliente de PC baseado em Apache Axis também foi implementado.

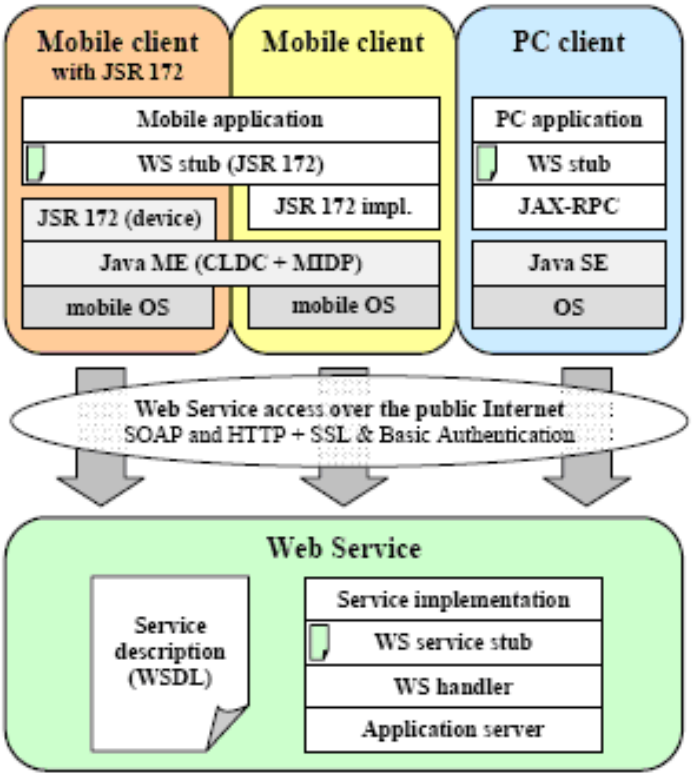


**Figura 3.1 – Arquitetura *Mobile* SOA**

A demonstração de conceito da arquitetura proposta identificou que serviços com processamento longo não podem ser usados diretamente, já que JSR 172 somente suporta mensagens síncronas. Operações assíncronas foram simuladas através do uso de uma técnica onde o cliente repetidamente busca por uma resposta, através do envio de requisições de serviço, até que uma resposta seja disponibilizada. A assincronicidade é mais propriamente atingida pelo uso da característica de empurrar registro do MIDP, permitindo que aplicações possam ser iniciadas de fora de uma requisição, via SMS, por exemplo. Em um ambiente onde os dispositivos possuam endereços IP alcançáveis, essa característica de enviar registro do MIDP pode ser combinada com uma pequena versão de um manipulador de *web services* para permitir o fornecimento de serviços por dispositivos leves.

## 3.2 Arquitetura *Mobile Host*

A arquitetura *Mobile Host* [Srirama 2006] apresenta uma abordagem para o fornecimento de *web services* móveis. Em seus estudos, os desafios chave encontrados foram:

- Manter os *web services* compatíveis de tal forma que os clientes não notassem a diferença.
- Projetar o hospedeiro móvel com uma pequena parcela dos recursos disponíveis em um dispositivo móvel (em seu protótipo, 130 KB).
- Limitar a sobrecarga de performance das funcionalidades para que nem os serviços, nem as funções normais do *smartphone* fossem impedidas de funcionar.

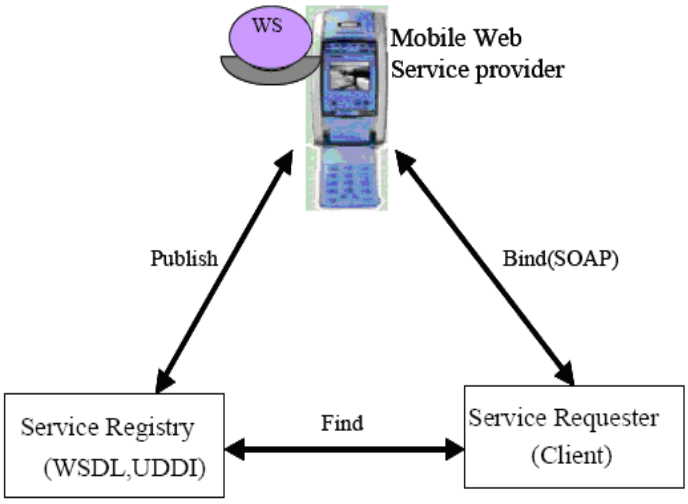


**Figura 3.2 – Configuração arquitetural básico do *Mobile Host***

Considerando esses desafios, juntamente com os requisitos arquiteturais e de casos de uso, a arquitetura *Mobile Host* fornece as seguintes características:

- Suporte às requisições HTTP padrão de servidor *web*;
- Fornecimento de *web service* para requisições de clientes usando SOAP sobre HTTP;
- Capacidade de manipular requisições concorrentes;
- Suporte à implantação de serviços em tempo de execução;
- Suporte à análise de desempenho.

*Mobile Host* foi desenvolvido como um manipulador de *web service* construído sobre o topo de um servidor *web* normal. As requisições para o *web service* enviadas por tunelamento HTTP são desviadas e manuseadas por um manipulador de *web service*.

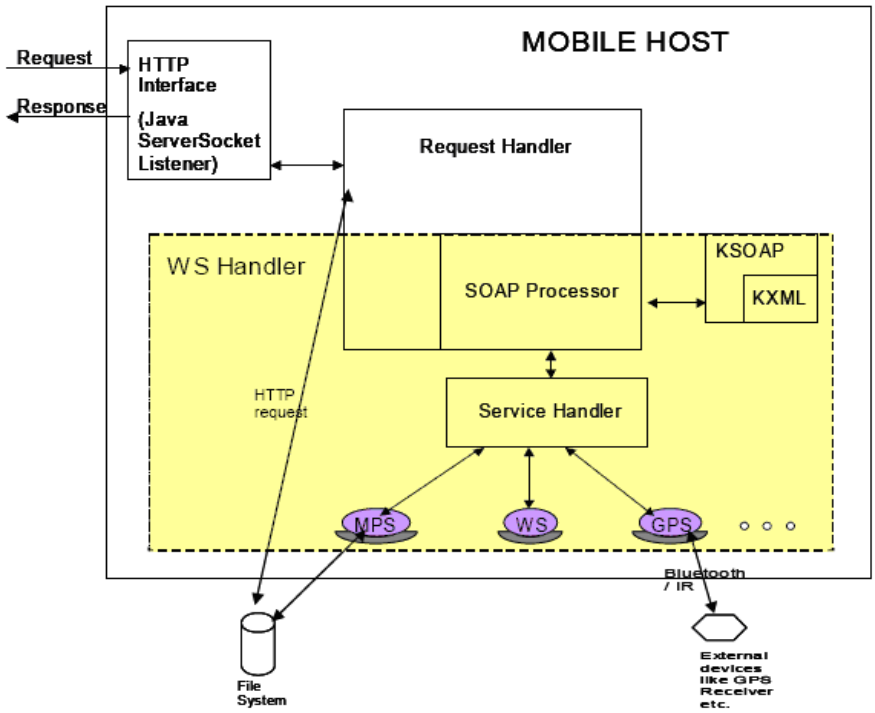


**Figura 3.3 – Arquitetura do núcleo do *Mobile Host***

A figura 3.3 apresenta o núcleo da arquitetura *Mobile Host*. Na interface HTTP, *Mobile Host* fica esperando por requisições HTTP GET/POST em um soquete do servidor. Quando *Mobile Host* recebe uma requisição, o soquete do servidor a aceita, cria um soquete para a comunicação e inicia uma nova *thread* de execução, criando uma instância do manipulador de requisições (*request handler*). Este extrai a mensagem de entrada do canal de entrada do soquete e checa por requisições para o *web service*. Em caso afirmativo, o

manipulador de requisições as processa e envia as respostas escrevendo no canal de saída do soquete.

Se a mensagem contém uma requisição *web service*, o componente manipulador de *web service* (*Web Service Handler*) processa a mensagem, lendo o corpo da mensagem HTTP e de-serializando a requisição SOAP de objetos Java através do processador SOAP. Esses objetos são passados para o manipulador de serviço (*service handler*), que extrai seus detalhes e invoca o respectivo serviço. Os *web services* implantados no *Mobile Host* podem acessar o sistema de arquivos local ou qualquer dispositivo externo, como um receptor GPS, usar infravermelho, *bluetooth*, etc., e podem implementar lógica de negócio. O manipulador de requisições serializa a resposta e prepara uma mensagem de resposta HTTP, que retorna ao cliente pelo canal de saída do soquete.

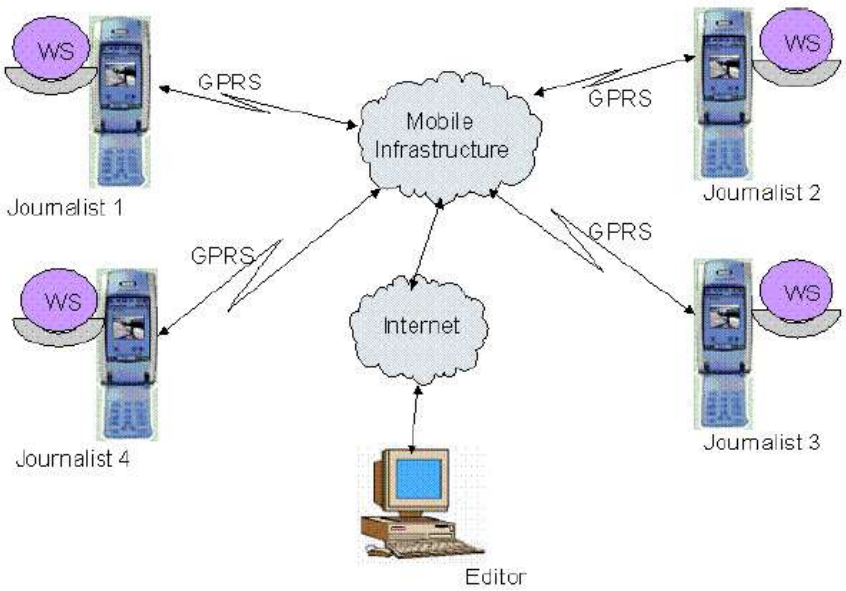


**Figura 3.4 – Cenário de utilização do *Mobile Host***

Um exemplo de cenário de uso da arquitetura *Mobile Host*, ilustrado na figura 3.4, envolve a coordenação entre jornalistas e suas respectivas organizações. Jornalistas podem estar em diferentes localizações ao redor do globo, cobrindo diferentes eventos. Um editor sempre pode manter-se informado da localização de seus jornalistas e do conteúdo que eles recolheram. Por exemplo, podem navegar pelas fotos tiradas pelo jornalista. Aplicações clientes podem ser desenvolvidas pelo editor, as quais sincronizam as informações armazenadas pelo editor e os dados do *Mobile Host*. A diferença chave das soluções tradicionais onde jornalistas enviam seu conteúdo, para um servidor suportado pelo editor é

que o acesso paralelo ao *Mobile Host* é possível por ambos, editor e jornalista, e até mesmo outros jornalistas.

## 3.3 *Peer-to-Peer web services*

Em sistemas *ad-hoc* sem qualquer infra-estrutura, o paradigma *Peer-to-Peer* é atrativo. Assim, uma arquitetura de *web services* utilizando o paradigma P2P [Gehlen 2005] apresenta-se como uma solução interessante de infra-estrutura para esse tipo de sistema, sendo mais um passo em direção à computação ubíqua. Nesta infra-estrutura, cada *peer* pode desempenhar uma ou mais funções dentro de uma arquitetura orientada a serviço, conforme demonstram as 4 próximas figuras. De acordo com o contexto deste trabalho relacionado, a arquitetura clássica de SOA é ilustrada na figura 3.5. Nessa arquitetura existem 3 atores principais:

- Provedor de Serviços (P);
- Requisitor de Serviços (R);
- *Broker* de Serviços (B): repositório com informações sobre serviços (UDDI); contém informações sobre o provedor de serviços (*white pages*), categorias que caracterizam os serviços (*yellow pages*) e a interface dos serviços (*green pages*).

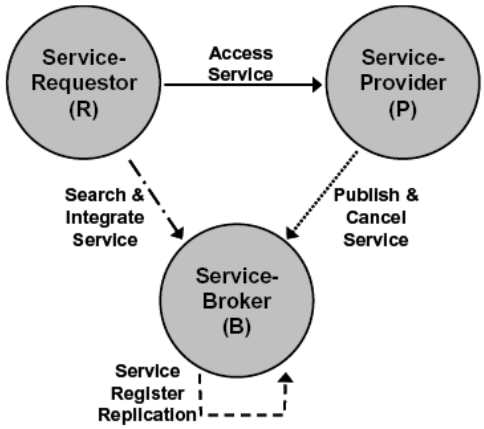


**Figura 3.5 – Arquitetura orientada a serviço simples (P-B-R)**

Dando seqüência à explicação da arquitetura P2P *web services*, a figura 3.6 ilustra diversos *peers* em um cenário clássico de SOA, onde diversos fornecedores de serviços (P) publicam seus serviços no repositório (B), os quais são encontrados pelos requisitores (R) através de consultas a B e, finalmente, R acessa os serviços de P.

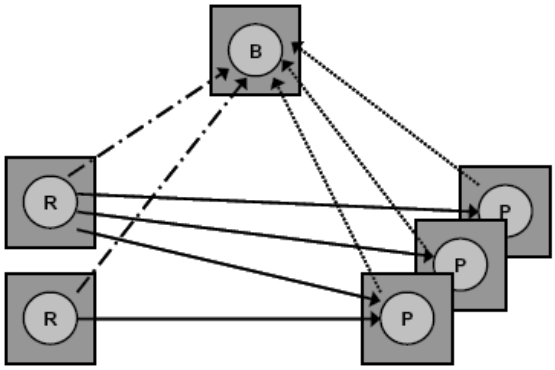


**Figura 3.6 – Cenário SOA cliente-servidor**

Em uma arquitetura de serviços P2P, cada nó, que pode ser qualquer tipo de dispositivo, móvel ou não, deve ser capaz de publicar e consumir serviços, como ilustra a figura 3.7. Neste cenário, a infra-estrutura é fixa e utiliza padrões da indústria como DHCP e DNS, além de um repositório de *web services* UDDI.

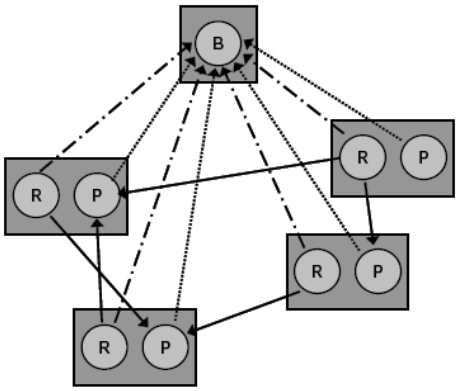


**Figura 3.7 – Cenário P2P SOA com infra-estrutura fixa**

Segundo o autor, embora o cenário apresentado acima pareça simples, em um cenário móvel real, composto por rede *ad-hoc*, existem algumas dificuldades:

- Como e onde o serviço deve ser publicado e descoberto (centralizado ou descentralizado)?
- Os nós devem ser endereçáveis, mas na maioria das redes móveis atuais como GSM/GPRS o endereço IP fica escondido através de um NAT.
- O limitado poder de processamento e a pouca memória dos dispositivos móveis atuais causam dificuldades no fornecimento de funcionalidades de servidor.
- A comunicação móvel não é confiável não oferece QoS contínuo.

Para resolver essas questões, um segundo cenário P2P SOA é apresentado, conforme ilustra a figura 3.8. Este cenário é uma rede *ad-hoc* (MANET) pura, sem nenhuma infra-

estrutura, com nós entrando e saindo da rede constantemente. Neste cenário, não existe um único *broker* centralizado (B). Ao invés disso, alguns nós incluem um B. Alguns dispositivos são somente consumidores de serviços (R), como, por exemplo, controles-remotos, terminais e *tablet* PC's. Outros são somente fornecedores de serviços (P), como, por exemplo, controlador de luz e servidor de mídia. Já outros são tanto consumidores como fornecedores de serviços. Pelo menos um nó dentro do ambiente tem que ser um *broker* (B). Esse ambiente heterogêneo é caracterizado por nós com uma grande quantidade de capacidades e poder computacional.

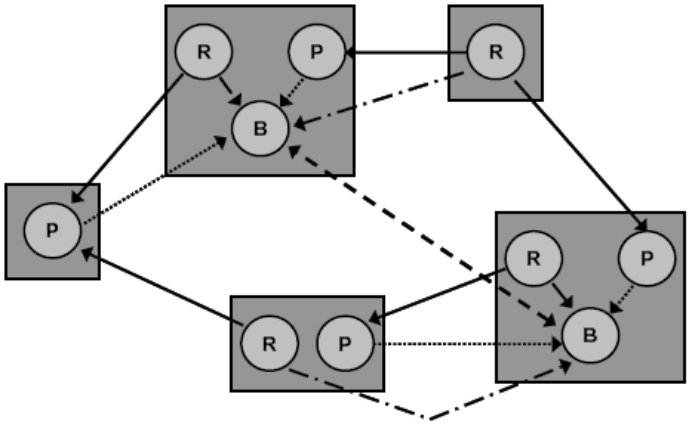


**Figura 3.8 – Cenário *ad-hoc***

A arquitetura para habilitar P2P *web services* é ilustrada na figura 3.9. O componente de *software* central é o componente SOAP, que fornece um *parser* XML, protocolos de transporte (HTTP, WAP e BEEP), algoritmos de segurança e as portas para o cliente e o servidor. Esse componente é a interface do ambiente distribuído. Os objetos do tipo *client proxy* são representações dos serviços remotos e são gerados e atualizados pelo *Remote Service Repository*, dentro do componente de gerenciamento. Os serviços implantados dentro do *framework* são publicados em WSDL no *Local Service Repository*, também dentro do componente de gerenciamento. Esses dois repositórios fornecem os mecanismos de registro e descoberta de serviços. O componente QoS é responsável por monitorar as interfaces de rede (banda disponível, latência) e o carregamento do dispositivo.

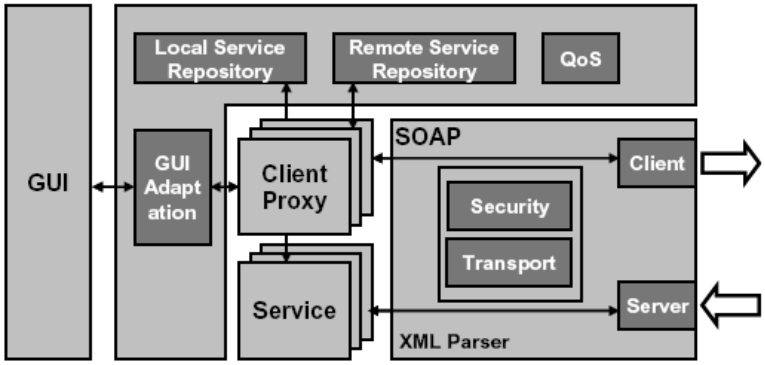


**Figura 3.9 – Arquitetura de *software* que habilita P2P *web services***

A implementação para dispositivos móveis é baseada nos pacotes kSOAP e kXML. Como kSOAP somente suporta vinculação HTTP (HTTP-*Binding*), para publicar serviços de um dispositivo para outro é necessário um servidor para HTTP-*Binding*. A figura 3.10 apresenta com mais detalhes a pilha de protocolos sobre os dispositivos móveis, que é explicada abaixo.

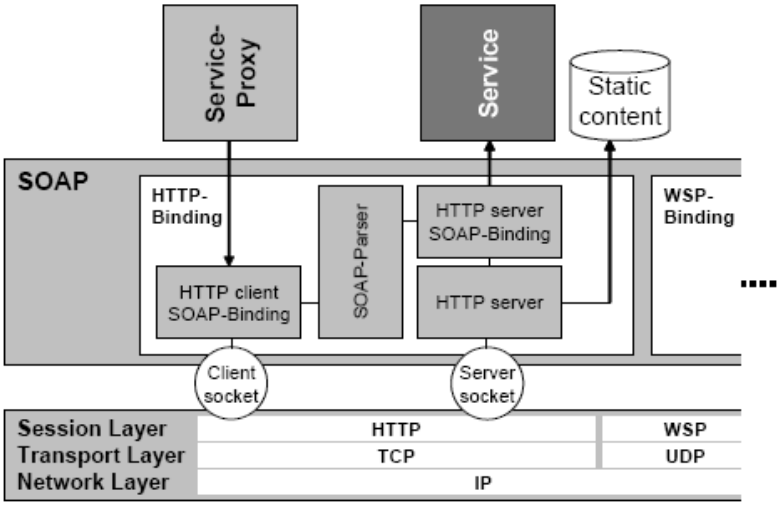


**Figura 3.10 – Servidor HTTP-*Binding***

A infra-estrutura considerada é baseada em IP, camada mais baixa da arquitetura. Dentro do *middleware* móvel, a camada SOAP é acoplada aos protocolos de sessão e de transporte. HTTP define os papéis fixos de cliente e servidor, portanto, dentro de uma sessão HTTP, um dispositivo possui o papel de cliente, enquanto outro possui papel de servidor. No *middleware* P2P *web services*, kSOAP foi estendido para possibilitar que cada dispositivo possa atuar em ambos os papéis. No *middleware* há um servidor HTTP esperando por requisições de entrada através do soquete do servidor. As requisições que contêm conteúdo SOAP são encaminhadas para o servidor HTTP-*Binding*, o qual acopla os serviços para

SOAP. As mensagens de entrada SOAP são correspondidas dentro do HTTP-*Binding* para invocações de métodos dos serviços correspondentes. O serviço de *proxy* (*Service-Proxy*) é acoplado do cliente do HTTP-*Binding* para SOAP. Todos os métodos locais para o *proxy* são mapeados em chamadas SOAP remotas.

Para suportar essa arquitetura, foi criado um *framework* de aplicação P2P que possibilita *web services* em nós P2P. Diversos componentes fazem parte desse *framework*, sendo que o principal deles é o servidor SOAP, ilustrado na figura 3.11. O *framework* foi implementado em Java ME. O *middleware* baseado em *web services* móveis é apresentado em detalhes por [Pham 2005].

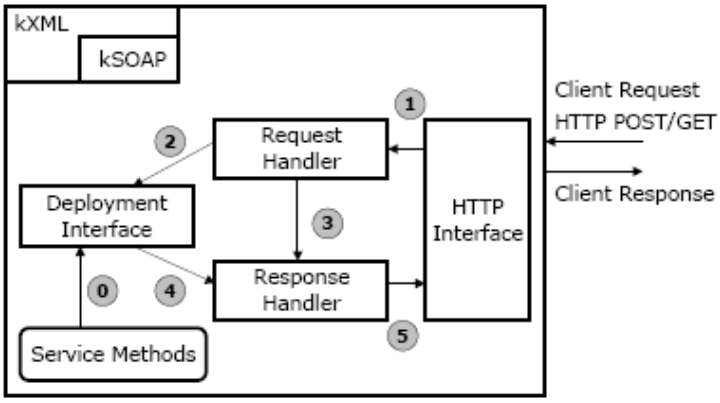


**Figura 3.11 – Estrutura do núcleo do servidor SOAP**

De forma simplificada, o funcionamento do servidor SOAP é o seguinte:

- Para servir requisições de clientes, métodos de serviços são iniciados na fase de inicialização do servidor SOAP e enviados ao *Deployment Interface* (ponto 0).
- O servidor SOAP torna os serviços disponíveis através de uma interface HTTP.
- Quando um cliente conecta ao servidor SOAP enviando uma requisição HTTP POST/GET, o soquete do servidor aceita a conexão cliente e retorna um soquete contendo um *stream* com os dados de entrada.
- A partir desse *stream* com dados de entrada, uma mensagem de requisição HTTP é extraída e passada ao *Request Handler* para processamento (ponto 1).
- O *Request Handler* usa kSOAP e kXML para processar a mensagem, de-serializando o XML em objetos Java e passando-os juntamente com outras informações necessárias ao *Deployment Interface* (ponto 2).
- Na seqüência, o *Request Handler* chama o *Response Handler* para formatar uma resposta ao cliente (ponto 3).

- O *Response Handler* pega o resultado do *Deployment Interface* (ponto 4).
- Para finalizar, o *Response Handler* e envia o resultado para o cliente através de um *stream* de dados de saída correspondente ao soquete de conexão do cliente (ponto 5).

O leve servidor SOAP foi implementado para executar sobre as plataformas Java ME CDC e CDLC/MIDP e Java SE. Ele requer uma baixa utilização de memória e é capaz de suportar requisições de clientes concorrentes. Principalmente sob as limitações da API Java ME, foram implementadas duas estratégias para manipulação de *threads* concorrentes. A primeira deles segue o seguinte algoritmo:

1. Inicia gerando n *threads* para manipulação de n requisições de clientes em paralelo (n é igual a 4, neste caso).
2. Aguarda até que a última *thread* (a enésima *thread*) termina. Nenhuma *thread* é criada durante esse intervalo.
3. Inicia novamente a geração de outras n *threads* e repete o passo 2.

A segunda estratégia é uma implementação de um *thread pool*, no qual são definidos alguns números fixos de *threads* (um vetor de n *threads* trabalhadoras, n é igual a 4, neste caso) para utilização. O *thread pool* associará uma tarefa (processamento de uma requisição de cliente) para cada uma dessas *threads* trabalhadoras. Quando qualquer uma dessas *threads* termina sua tarefa, uma nova tarefa é associada imediatamente. Se o número de requisições clientes concorrentes é maior que n, essas requisições são colocadas em uma fila para processamento futuro por qualquer uma das *threads* trabalhadoras.

Em comparação com a primeira, a segunda estratégia usa mais eficientemente os recursos dos dispositivos porque permite um número fixo de *threads* trabalhadoras para manipulação de requisições de clientes continuamente. No entanto, como o *thread pool* cria um vetor com n objetos *threads*, mais memória é alocada para este vetor. Por exemplo, para manter o estado de espera para requisições de clientes, o servidor SOAP com o mecanismo de *thread pool* consome aproximadamente 125 KB com 5 *threads*, enquanto que a primeira estratégia requer em média 105 KB de memória, não importando a quantidade de *threads*.

## 3.4 Sliver

O projeto Sliver (http://mobilab.wustl.edu/projects/sliver/) [Hackmann 2006] se auto-define como um motor de execução SOAP e BPEL para dispositivos móveis.

Simplificadamente, o Sliver levou o BPEL de grandes servidores para pequenos dispositivos móveis, além da disponibilização de serviços e da criação de composições com eles. O Sliver suporta muitos dos requisitos da computação ubíqua, como o tratamento à desconexão, por exemplo, visto que os *web services* participantes de uma composição podem estar ou não *online*. Ainda, permite a comunicação entre os serviços utilizando HTTP ou *bluetooth*, entre outros aspectos.

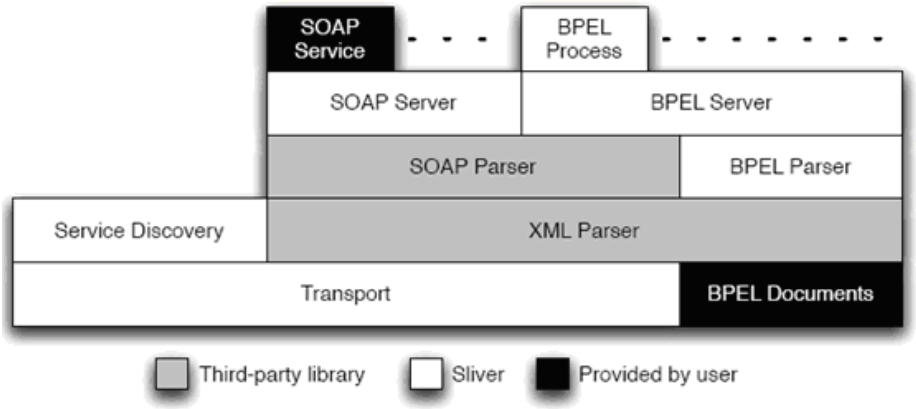


**Figura 3.12 – Arquitetura do motor de execução Sliver**

Sliver é um motor de execução de processos de *workflow* BPEL que suporta uma grande variedade de dispositivos, desde telefones celulares até *desktops*. A arquitetura do motor de execução Sliver, ilustrado na figura 3.12, usa uma série de *parsers* escritos à mão desenvolvidos utilizando os leves pacotes kXML  e kSOAP. [Hackmann 2006] afirma que o desenvolvimento de motores de *middleware* como Sliver é um passo importante em direção ao objetivo de longo prazo de levar *groupware* para dispositivos móveis.

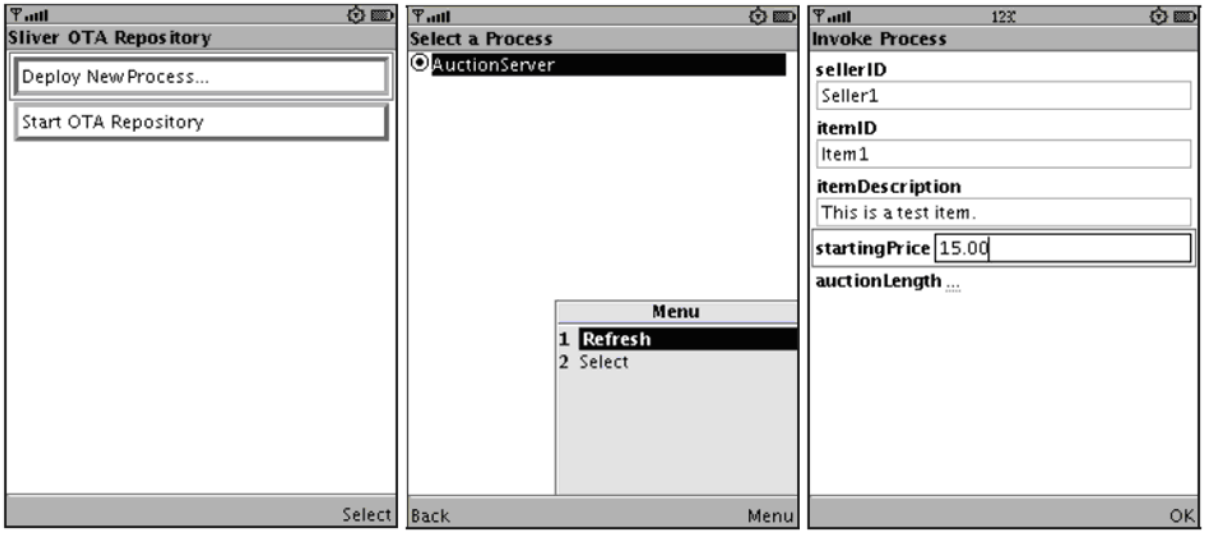


**Figura 3.13 – Protótipo Sliver**

Como justificativas para o projeto [Hackmann 2007], no que diz respeito a aplicações de computação ubíqua, a incompatibilidade de arquiteturas de *hardware* e *software* está desencorajando interações *ad-hoc* entre os dispositivos. Ainda, o modelo de comunicação inflexível do BPEL efetivamente proíbe a sua utilização sobre os tipos de redes sem fio dinâmicas usadas pela maioria dos dispositivos de computação ubíqüos. Por isso, propuseram extensões ao BPEL, implementadas no Sliver, que resolvem essas restrições, transformando o BPEL em uma plataforma versátil para aplicações computacionais ubíquas interoperáveis. Como prova de conceito da arquitetura Sliver, foi desenvolvido um protótipo, ilustrado em três imagens na figura 3.13.

### 3.4.1 Analisador BPEL

Ao contrário dos serviços SOAP padrões, que geralmente são entidades *stand-alone* implementadas em alguma das muitas linguagens existentes, processos BPEL usam esquemas XML padronizados para descrever interações entre outros serviços SOAP. O analisador BPEL (BPEL *Parser*) do Sliver representa cada *tag* BPEL com uma classe Java correspondente; por exemplo, a *tag* <reply> é representada pela classe Reply. Cada classe tem um construtor que usa a biblioteca kXML para analisar a parte correspondente das descrições do processo. Esses construtores tomam decisões de validação local, na qual verificam a validade do documento BPEL; por exemplo, o construtor da classe Reply lança uma exceção se encontra uma tag <reply> sem um atributo partnerLink ou partnerGroup. Essas decisões locais eliminam muito da necessidade de uma pesada biblioteca de análise para validação completa de XML.

BPEL subdivide o processo em escopos aninhados; informação de estado (como variáveis e *links* de comunicação com *web services*) é compartilhada entre todas as atividades dentro de um escopo. Sliver mantém toda a informação conhecida sobre cada escopo em tempo de análise (por exemplo, os tipos e nomes das variáveis) em objetos de dados de escopo. Um objeto de instância de processo também é instanciado para cada instância do *workflow*; ele monitora a informação que varia através das instâncias (por exemplo, os valores das variáveis).

Sliver suporta todas as construções de atividades básicas e estruturadas em BPEL, com a exceção da atividade de compensação, e suporta consultas e transformações de dados básicas expressadas usando a linguagem XPath. Sliver também suporta o uso de escopos BPEL e permite a definição, dentro deles, de variáveis locais, manipuladores de erros e manipuladores de eventos.

### 3.4.2 Servidor BPEL

Sliver implementa um motor de execução de composições de *web services* através de um servidor BPEL (BPEL *Server*), implementado como uma classe Java chamada BPELServer. Essa classe hospeda os processos BPEL que são gerados pelo analisador BPEL. A classe BPELServer estende o servidor SOAP (SOAP *Server*) para invocar esses processos no lugar de serviços SOAP. Ele também adiciona diversos métodos adicionais para sua API pública. O método addProcess cria um processo BPEL no *namespace* especificado. Ele lê a especificação do processo BPEL fornecida pelo *stream* de entrada. Esse método invoca o analisador BPEL descrito acima, que encapsula o *workflow* completo em um objeto do processo chamado Process.

O método bindIncomingLink mapeia os *links* parceiros de chegada para os tipos de mensagens que eles aceitam como entrada. Igualmente, o método bindOutgoingLink mapeia *links* de parceiros de saída para *endpoints* concretos. Esses métodos permitem que aplicações que embarquem o Sliver apliquem uma ampla variedade de *links* de parceiros mapeando políticas flexíveis, de acordo com as necessidades da aplicação. Tais políticas podem variar em complexidade, desde simples mapeamentos fortemente codificados até o re-mapeamento dinâmico de *links* parceiros em tempo de execução usando a camada de descoberta de serviço.

A camada do servidor BPEL do Sliver é capaz de hospedar uma ampla variedade de processos de *workflow* úteis. Um *framework* foi proposto para permitir a análise de linguagens de *workflow* em termos de um conjunto de 20 padrões de *workflow* [Van der Aalst 2003] comumente recorrentes. Sliver atualmente suporta 14 desses 20 padrões de forma completa, e mais um deles parcialmente. Até mesmo sobre PDA e telefones celulares, *hardwares* com recursos limitados, o custo da execução da maioria dos padrões no Sliver é na ordem de 100 ms.

## 3.5 Outros motores de execução

Componente essencial para a execução de *web services* em dispositivos móveis, os motores de execução estão sendo portados para plataformas com menos recursos. Relacionando com a arquitetura que está sendo proposta, são relevantes os seguintes motores de execução:

- Servidores *web*;
- Servidores de *web services*;

- Servidores de composições de *web services*.

Alguns exemplos desses tipos de componentes, que geraram publicações científicas, foram ilustrados acima. Além desses, no entanto, alguns trabalhos de caráter mais comercial podem ser referenciados e são apresentados a seguir.

### 3.5.1 JME SOAP *Server*

JME SOAP *Server* (http://sourceforge.net/projects/jmesoapserver/) é um *framework* de servidor SOAP para a plataforma Java ME. Com ele é possível executar *web services* sobre dispositivos móveis. Suporta SOAP 1.2, mas sem anexos. Roda sobre MIDP 2.0 / CLDC 1.1 ou dispositivos compatíveis com CDC com máquinas virtuais com memória suficiente.

### 3.5.2 *Mobile Web Server*

*Mobile Web Server* (http://research.nokia.com/research/projects/mobile-web-server/) é uma solução da Nokia de servidor *web* para dispositivos móveis. Consiste de um servidor *web* produzido a partir do servidor Apache httpd e módulos da linguagem Python portados para o sistema operacional Symbian, juntamente com uma solução tecnológica para fornecer acesso URL e HTTP globais ao dispositivo móvel. Essa solução consiste de um módulo no dispositivo móvel e outro em um servidor *gateway* [Nokia 2007][Wikman 2006]. O MWS já está disponível comercialmente através do endereço http://mymobilesite.net/.

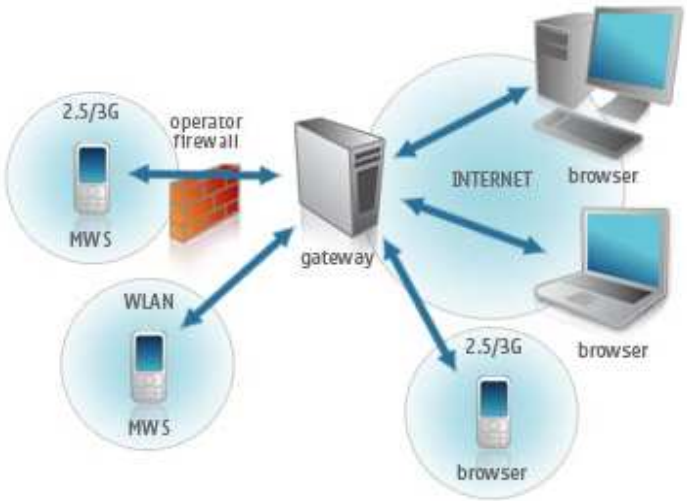


**Figura 3.14 – Visão geral da arquitetura do MWS**

### 3.5.3 i-Jetty

i-Jetty (http://code.google.com/p/i-jetty/) é um projeto que portou o popular servidor *web* Jetty (http://jetty.mortbay.com/jetty-6/), implementado em Java, para a plataforma Android. Com ele, um telefone celular com Android pode funcionar como um servidor *web standalone* ou, até mesmo, como parte de uma pequena rede através da sincronização com outros telefones com Android e i-Jetty. Testes preliminares mostraram que o i-Jetty possui um desempenho muito bom no Android.


**Figura 3.15 – i-Jetty em execução no emulador do Android**

Com i-Jetty é possível disponibilizar *servlets* Java como aplicações *web* na plataforma Android. i-Jetty não pretende ser um servidor de *web services* ou de composição de *web services*, mas é uma primeira abordagem de provimento de aplicações *web* para essa plataforma.

# 4 SMALLSOA

Este capítulo descreve o motor para execução de composições de serviços em ambientes móveis chamado SmallSOA. A seção 4.1 introduz SmallSOA e a arquitetura maior na qual ele está inserido chamada U-SOA (http://sourceforge.net/projects/usoa/), juntamente com as justificativas para a criação de ambos. Uma visão geral do motor através de suas principais características é apresentada na seção 4.2. Na seção 4.3 são apresentados os novos papéis nas arquiteturas orientadas a serviços. A seção 4.4, por sua vez, apresenta os principais requisitos que foram observados para construção do motor.

## 4.1 O projeto U-SOA

O motor SmallSOA, anteriormente chamado de SOA *Engine* [Zanuz 2008], faz parte de um projeto maior de nosso grupo de pesquisa de nome U-SOA [Zanuz 2008b]. U-SOA projeta um *framework* para a construção de sistemas colaborativos ubíquos baseados em arquiteturas orientadas a serviços. U-SOA prevê também estruturas capazes de fornecerem mapas das composições de serviços e dos respectivos serviços utilizados nessas composições, permitindo, dessa forma, que diversas análises de impacto e de QoS possam ser construídas a partir desses mapas. O *framework* baseia-se em uma ontologia, que representa o modelo proposto através de uma especificação formal.

U-SOA (*A ubiquitous SOA framework*), conforme ilustra a figura 4.1, pode ser visto como uma pilha de tecnologias separadas em camadas. As camadas envolvidas na arquitetura contemplam desde níveis mais baixos, no qual se encontra o sistema operacional Android, até o topo da pilha, onde se encontram as aplicações colaborativas ubíquas suportadas pela arquitetura.

Atualmente, os seguintes trabalhos já foram ou estão sendo desenvolvidos:

- Servidor de *web services* SOAP *Server*;
- Motor de execução SmallSOA;
- Linguagem de composição de serviços inSOA, anteriormente chamada de SQLSOA [Zanuz 2008b];
- Ferramenta para análise de impacto na composição de serviços gImpact;
- Robô para descobrimento de serviços WSBot;

- Interface para composição de serviços em dispositivos móveis.

No topo da arquitetura são encontradas as aplicações colaborativas ubíquas que poderão ser criadas a partir do suporte dado pelos componentes desenvolvidos nas camadas inferiores de U-SOA.

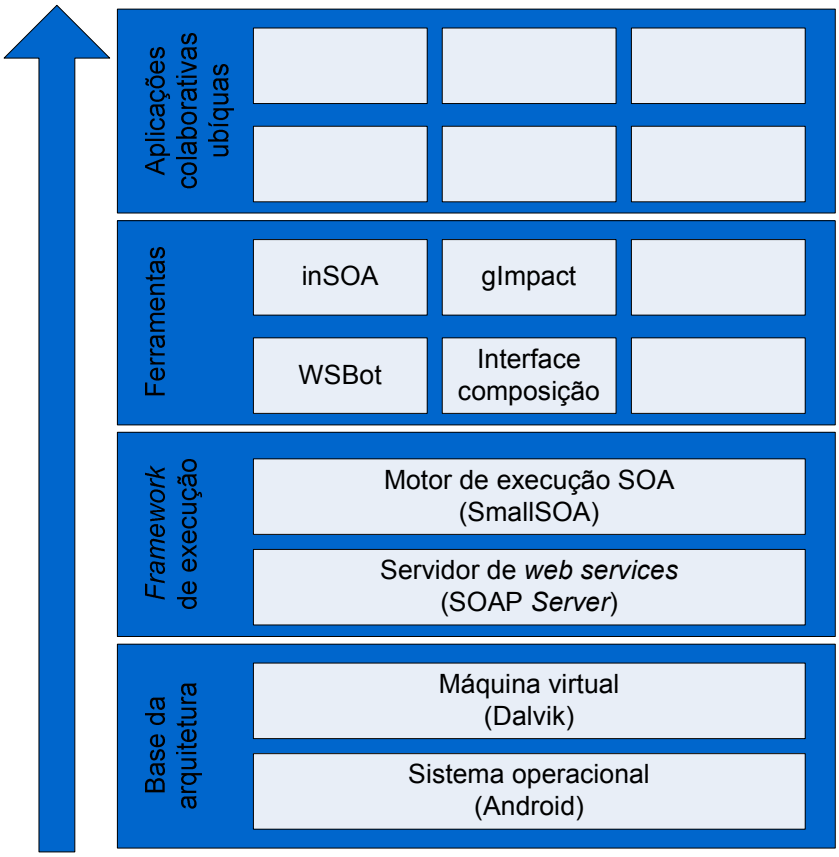


**Figura 4.1 – Arquitetura U-SOA**

### 4.1.1 SmallSOA – Introdução

Arquiteturas orientadas a serviço estão sendo indicadas para serem a base de sistemas colaborativos [Laso-Ballesteros 2006] e de sistemas móveis e ubíqüos. Isto decorre do baixo acoplamento e da independência de plataforma que SOA oferece, permitindo a integração perfeita de sistemas construídos sobre plataformas diferentes. SOA abstrai as camadas mais baixas de uma aplicação, o que a torna candidata ideal à maioria dos novos sistemas que estão sendo desenvolvidos e que necessitam interação com outros sistemas.

SOA baseia-se em três principais componentes: fornecedor de serviço, consumidor de serviço e repositório de serviços. Todo o tráfego entre esses componentes utiliza padrões da indústria baseados em XML, tais como SOAP. Os serviços, que na abordagem atual de SOA

são implementados usualmente como *web services*, são descritos através da linguagem WSDL. SOA propõe a composição desses serviços em novas aplicações compostas, que conceitualmente também são *web services* e, conseqüentemente, são descritos em WSDL. Essa composição é feita através de linguagens de composição de serviços, sendo que BPEL [Weerawarana 2002] está se tornando o padrão desse tipo de linguagem.

Atualmente, WSDL descreve os *web services* através de seus parâmetros de entrada e de saída. Ou seja, o *web service* é uma aplicação caixa-preta. Como a aplicação de composição de serviços é um novo *web service*, também descrita através de WSDL, não é possível identificar os serviços que fazem parte dela. Evidentemente, a aplicação composta é implantada em um servidor de aplicação que, através da descrição da composição feita em BPEL, conhece quais são e onde estão esses serviços. No entanto, essas informações não ficam visíveis externamente ao servidor. Dessa forma, é difícil determinar uma grande variedade de informações a respeito das composições de serviços como, por exemplo, a análise do impacto que a exclusão de determinado serviço acarretaria ao conjunto de aplicações compostas de uma empresa e informações relevantes a QoS. U-SOA e SmallSOA, entre outros aspectos, surgem para resolver esse tipo de limitação.

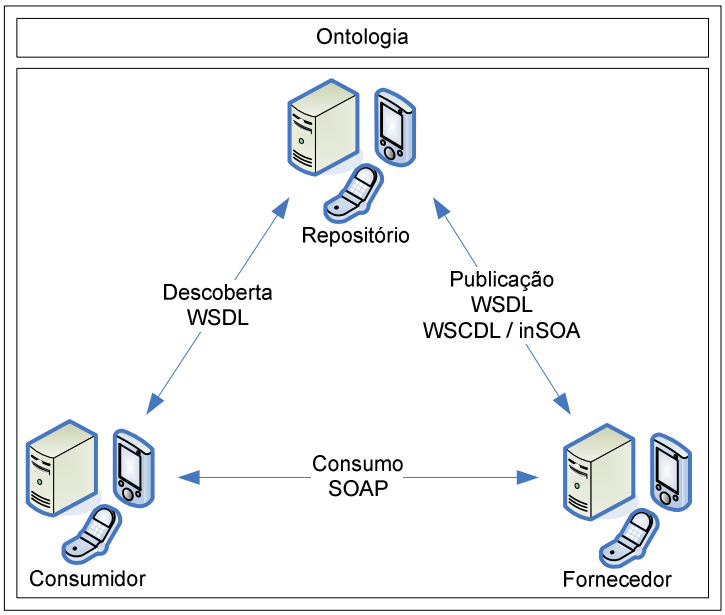


**Figura 4.2 – Arquitetura orientada a serviços ubíquos**

A figura 4.2 ilustra algumas melhorias na estrutura atual de SOA que estão disponíveis a partir da arquitetura que está sendo proposta. A primeira delas é a possibilidade de que cada nó do tradicional triângulo de relacionamentos Fornecedor-Repositório-Consumidor seja qualquer dispositivo, inclusive pequenos dispositivos móveis. A segunda é a utilização de

alguma linguagem de descrição de composições de serviços (WSCDL), no caso, inSOA, para a publicação das composições. Essa nova forma de descrição contém as informações adicionais necessárias para a identificação dos serviços participantes das composições, permitindo o mapeamento desses serviços, conforme ilustra a figura 4.3, e, conseqüentemente, diversas novas capacidades, entre elas a análise de impacto citada no parágrafo anterior.

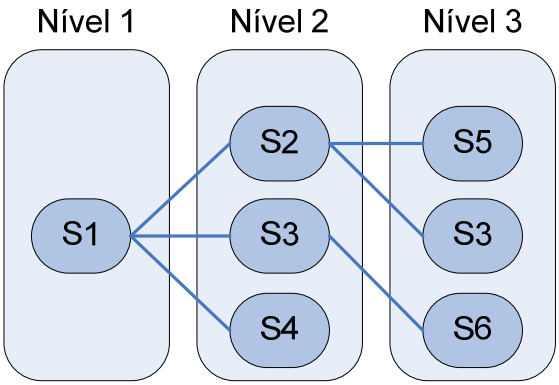


**Figura 4.3 – Serviços compostos apresentados em formato de árvore**

## 4.2 Visão geral do motor

SmallSOA é um motor para execução de composições de serviços (*web services*) em ambientes móveis, um dos componentes centrais da arquitetura U-SOA. O motor se caracteriza por oferecer condições para a execução de composições de serviços em ambientes móveis e distribuídos, conforme ilustra a figura 4.4.

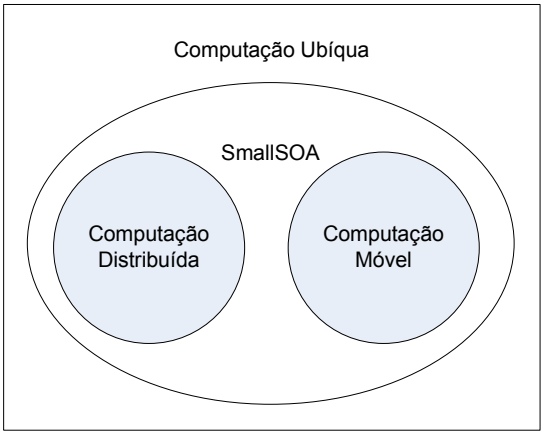


**Figura 4.4 – Contexto atual de SmallSOA**

Um motor para execução de composições de serviços é um componente de *software* que, dada uma linguagem de composição de serviços, interpreta uma descrição de composição escrita nessa linguagem e processa-a, normalmente retornando algo para um usuário cliente, conforme ilustra a figura 4.5. A descrição da composição processada pelo motor é escrita através de uma linguagem própria criada para o projeto U-SOA conhecida como inSOA.

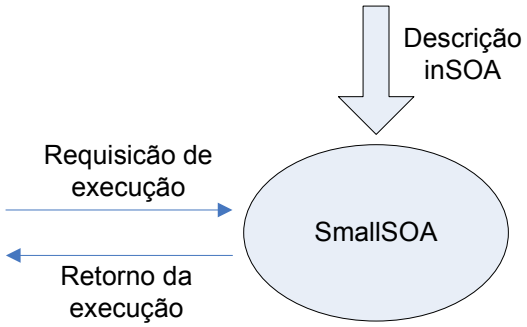


**Figura 4.5 – Interpretação de linguagem de descrição de serviços**

O motor SmallSOA deve, portanto, ser capaz de interpretar uma descrição de composição de serviços escritos através da linguagem inSOA. Os principais aspectos descritos pela linguagem e que devem ser interpretados pelo motor são:

- Chamadas a serviços e a outras composições de serviços;
- Manipulação dos retornos dos serviços e dos retornos ao usuário cliente;
- Manipulação de variáveis;
- Tratamento de exceções.

Além da sua função de interpretador ou executor de uma linguagem de composição de serviços, SmallSOA possui outras duas funções adicionais relacionadas e que dão apoio à principal:

- Publicação das composições de serviços;
- Disponibilização das composições de serviços.

Para que essas duas funções possam ser executadas, o motor SmallSOA precisa manipular um repositório de composições de serviços, onde estas possam ser armazenadas, quando da publicação, e disponibilizadas, quando da consulta à uma composição. Conceitualmente, a função principal do motor não exigia necessariamente a utilização de um repositório de composições, visto que poderia ser utilizada alguma outra técnica ou forma de execução. Porém, a utilização desse repositório é a abordagem mais indicada e permite uma

maior gama de funcionalidades ao motor. O relacionamento de SmallSOA com o repositório de composições de serviços é ilustrado na figura 4.6.

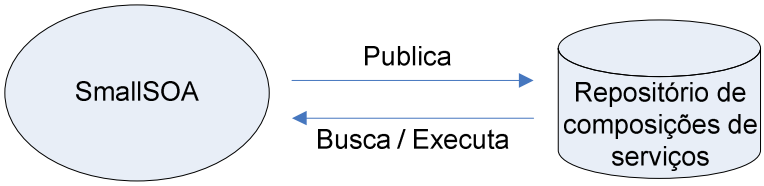


**Figura 4.6 – Repositório de composições de serviços de SmallSOA**

A execução de uma composição de serviços em ambientes móveis é uma tarefa especialista que consome bastantes recursos computacionais. Por sua natureza, dispositivos móveis possuem maiores deficiências nesse aspecto do que dispositivos para outros ambientes. Embora essas deficiências tenham diminuído nos últimos anos, elas ainda são relevantes e devem ser levadas em consideração na concepção de *software* para esse tipo de dispositivo. Por essa razão, o motor SmallSOA foi planejado para evitar o consumo excessivo de recursos computacionais. As principais limitações dos dispositivos móveis são:

- Poder de processamento;
- Memória disponível;
- Consumo de bateria;
- Espaço em disco.

Além das limitações específicas dos dispositivos, questões próprias do ambiente móvel são dificuldades adicionais e foram levadas em consideração na construção do motor SmallSOA. As principais limitações de ambientes móveis são relacionadas à questão da mobilidade em si, tais como:

- Áreas sem cobertura de sinal de rede ou com “sombras”;
- Quedas no sinal de rede.

Além de cuidados na construção do motor, para melhorar o seu desempenho algumas tarefas que poderiam ser feitas por ele foram destinadas ao compilador da linguagem de composição de serviços. Embora ambos os componentes sejam executados em dispositivos móveis, essa abordagem é adequada também pelo fato de que, uma vez publicada, uma composição pode ser executada *n* vezes. Então, é preferível consumir mais recursos no momento anterior à publicação do que nos momentos das execuções. As tarefas que foram destinadas ao compilador da linguagem são:

- Definição da ordem das chamadas dos *web services*, evitando que o motor consuma tempo e recursos para encontrar essa informação, uma vez que o usuário pode escrever a composição em qualquer ordem;
- Verificação das dependências entre os *web services* a serem chamados, indicando explicitamente quais *web services* podem ser invocados de forma paralela.

Com isso, SmallSOA pode focar especificamente na execução da composição de serviços, sua tarefa principal, sem se preocupar com questões referentes à manipulação da descrição da composição.

Em outros ambientes de composição de serviços, tais como BPEL, isso não ocorre porque o desenvolvedor da composição define essas informações manualmente em tempo de desenvolvimento, especificando exatamente a ordem de execução dos *web services* e informando através de instruções específicas se estes podem ser executados de forma paralela.

Quanto às peculiaridades dos ambientes móveis onde SmallSOA está inserido, a principal preocupação do motor diz respeito às possíveis quedas de sinais de rede. Essas quedas podem ocorrer de duas formas:

- Queda da conexão entre o usuário cliente e o motor SmallSOA;
- Queda da conexão entre o motor SmallSOA e os *web services*.

## 4.3 Novos papéis nas arquiteturas orientadas a serviços

U-SOA possui um servidor próprio de *web services*, que foi customizado a partir do JME SOAP *Server* [Gerlach 2006]. Com isso, U-SOA permite que qualquer dispositivo móvel atue como fornecedor de serviços, o que atualmente está limitado a plataformas de *hardware* de alto poder de processamento. Além desse novo papel, esses dispositivos mantêm o recém adquirido papel de consumidores de serviços. Devido ao motor SmallSOA, U-SOA passa também a possuir seu próprio servidor de composições de *web services*, o que incrementa ainda mais a quantidade de papéis que cada participante da arquitetura orientada a serviço possa atuar. Essa nova característica de fornecimento de serviços a partir de pequenos dispositivos móveis, também chamados de pares (*peers*), é essencial para que sistemas colaborativos ubíquos tornem-se realidade. A figura 4.7 ilustra alguns desses papéis dentro de um cenário de uma arquitetura orientada a serviços modificada.

Na figura é possível visualizar que se trata de um cenário bastante heterogêneo, contendo *web services* e composições de *web services* em ambientes móveis e não-móveis. Os

ícones que representam servidores e possuem a legenda Axis significam que têm instalado um servidor de *web services* Apache Axis [Axis 2008]. O outro ícone de servidor contém um repositório UDDI. Todos os servidores Axis possuem os seus *web services* e composições de *web services* publicados no repositório UDDI. Os ícones que representam dispositivos móveis, (telefones celulares e *smartphones*), possuem instalados, dependendo do caso, o SOAP *Server* customizado para U-SOA e o motor SmallSOA. Existe ainda um dispositivo móvel que contém um repositório de composições SmallSOA, com as publicações das composições inSOA dos dispositivos. Os *web services* dos dispositivos que possuem SOAP *Server* estão publicados no repositório UDDI.

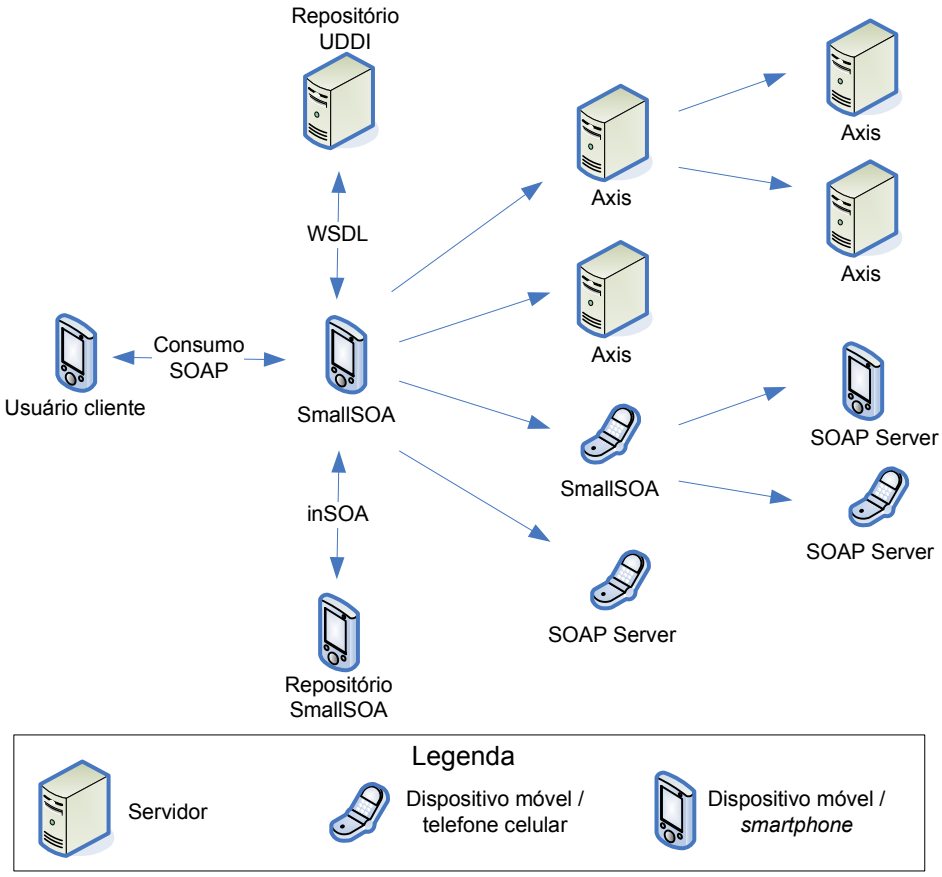


**Figura 4.7 – Cenário de utilização de SmallSOA**

O cenário inicia com um usuário cliente consumindo uma composição de serviços através de uma chamada SOAP a um dispositivo móvel com SmallSOA instalado. O dispositivo então busca a descrição da composição em um repositório SmallSOA, que pode estar implantado nele mesmo ou em outro local da Internet. Feito isso, o dispositivo terá disponível a descrição e poderá iniciar a execução. A execução significa chamadas a *web services* ou a outras composições de *web services*, de forma transparente para o dispositivo.

Como motor SmallSOA o dispositivo está atuando com o papel de fornecedor e ao executar *web services* está atuando com o papel de cliente. O dispositivo pode ainda atuar como o repositório de composições inSOA.

**Quadro 4.1 – Papéis dentro de arquiteturas orientadas a serviços**

| | **SOA** | | **SmallSOA** | | **U-SOA** | |
|---|---|---|---|---|---|---|
| | **Tradicional** | **Móvel** | **Tradicional** | **Móvel** | **Tradicional** | **Móvel** |
| **Fornecedor** | WS/C | | | C | WS/C | WS/C |
| **Consumidor** | WS/C | | WS/C | C | WS/C | WS/C |
| **Repositório** | WS/C | | | C | WS/C | WS/C |

Legenda WS: *web services*
C: composição de *web services*

O quadro 4.1 apresenta de forma resumida os novos e antigos papéis existentes dentro das arquiteturas orientadas a serviço. Conforme o quadro, uma arquitetura orientada a serviço clássica (SOA) possui os papéis de fornecedor, consumidor e repositório, de *web services* e de composições de *web services* (WS/C), mas apenas em ambientes tradicionais, não-móveis. SmallSOA introduz SOA em dispositivos móveis, possuindo os papéis de fornecedor, consumidor e repositório mas somente de composições de *web services* (C). SmallSOA também atua como consumidor de *web services* e de composições de *web services* (WS/C) em ambientes não-móveis. U-SOA, por sua vez, permite os papéis de fornecedor, consumidor e repositório de *web services* e de composições de *web services* (WS/C) em ambientes móveis e não-móveis.

## 4.4 Principais requisitos implementados de SmallSOA

Requisitos são capacidades e condições às quais o sistema deve atender [Jacobson 1999]. Para que o motor SmallSOA possa ser utilizado como o componente que U-SOA necessita para esta função, inúmeros requisitos funcionais e não-funcionais precisam ser atendidos, sendo que alguns deles são apresentados nesta seção. A forma mais comum para representar requisitos funcionais é através de casos de uso. Requisitos não-funcionais, por sua vez, normalmente são descritos agrupados por categorias, tais como usabilidade, desempenho e segurança. Nesta seção, no entanto, os requisitos do motor são listados sob a forma de características do sistema [Larman 2004].

As características de SmallSOA listadas a seguir dizem respeito a requisitos do ponto de vista do projeto U-SOA.

### 4.4.1 Interpretar a linguagem inSOA

O motor funciona como um interpretador de código XML gerado pelo compilador da linguagem inSOA. A linguagem de composição de serviços inSOA escrita pelo usuário é apresentada na figura 4.8.

```
invoke http://unisinos.br/ws/masters.do as a, -- Invoke 1
    http://sqlsoa.br:8090/ws/researchers.name as b, -- Invoke 2
    http://receita.fazenda.gov.br/validCPF.done as c -- Invoke 3
into /unisinos/master/masters as master,
    /sqlsoa/names as names,
    /names as CPF
 set a.firstname := names/name || names/first,
     b.age := names/age * -1,
     c.cpf := master/cpf
where not names/lastname <> ''
 and master/cpf = names/cpf
return master/firstname, master/lastname, CPF/valid
fail a: invoke http://sqlsoa.br:8091/ws/sendmail.do as d
         set d.to := names/name || '@sqlsoa.br'
     other:
      invoke http://sqlsoa.br:8091/ws/crash.done as e;
```

**Figura 4.8 – Linguagem de composição de serviços inSOA**

Um exemplo de XML interpretado pelo motor SmallSOA é ilustrado na figura 4.9. Esse XML é uma tradução da linguagem inSOA e é formatado de modo que facilite a sua execução. O motor deve suportar todas as funcionalidades descritas na linguagem.

```
- <inSOA id="airportComposition" tags="travel, money" hash="67dd30dfa352b3ea329e2fbb762bf034abb5e50e">
    <inSOAScript/>
  - <Invokes>
    - <Invoke Method="SOAP" Namespace="http://www.webserviceX.NET"
      EndPoint="http://www.webserviceX.NET/airport.asmx" Operation="getAirportInformationByAirportCode"
      Alias="a" Pipe="/Envelope/Body/getAirportInformationByAirportCodeResponse
      /getAirportInformationByAirportCodeResult" Into="airport" ParallelismLevel="1">
      - <Parameters>
          <Parameter Name="airportCode" Value="POA" Type=""/>
        </Parameters>
      </Invoke>
    </Invokes>
    <Wheres> </Wheres>
  - <Returns>
      <Return ReturnExpression="airport" TypeReturnExpression=""/>
    </Returns>
    <Fails> </Fails>
  </inSOA>
```

**Figura 4.9 – XML gerado a partir de uma composição inSOA**

### 4.4.2 Executar o conjunto de padrões de composição previstos em inSOA

A linguagem inSOA implementa uma série de padrões de composição (*composition patterns*) [Van der Aalst 2003] através da combinação de seus comandos. SmallSOA deve executar todos eles. O quadro 4.2 apresenta uma comparação entre os padrões de composição suportados por BPEL e por inSOA. No quadro, o sinal "+" significa que a linguagem suporta o padrão, o sinal "-" significa que não suporta o padrão descrito, enquanto o sinal "+/-" significa que a linguagem não implementa de forma nativa o padrão, mas é possível representá-lo de forma indireta.

**Quadro 4.2 – Padrões de composição implementados nas linguagens BPEL e inSOA**

| # | Padrão de composição | BPEL | inSOA |
|---|---|---|---|
| **1** | *Sequence* | + | + |
| **2** | *Parallel Split* | + | + |
| **3** | *Synchronization* | + | + |
| **4** | *Exclusive Choice* | + | + |
| **5** | *Simple Merge* | + | + |
| **6** | *Multi-choice* | + | + |
| **7** | *Synchronizing Merge* | + | + |
| **8** | ***Multi-merge*** | **-** | **+** |
| **9** | ***Discriminator*** | **-** | **+** |
| **10** | ***Arbitrary Cycles*** | **-** | **+** |
| **11** | *Implicit Termination* | + | + |
| **12** | *Multiple Instances Without Synchronization* | + | + |
| **13** | *Multiple Instances With a Priori Design Time Knowledge* | + | +/- |
| **14** | *Multiple Instances With a Priori Runtime Knowledge* | - | +/- |
| **15** | ***Multiple Instances Without a Priori Runtime Knowledge*** | **-** | **+** |
| **16** | *Deferred Choice* | + | + |
| **17** | *Interleaved Parallel* | +/- | + |
| **18** | *Milestone* | - | +/- |
| **19** | *Cancel Activity* | + | + |
| **20** | *Cancel Case* | + | + |
| **21** | *Exception Handling* | + | + |

### 4.4.3 Execução paralela de *web services*

SmallSOA permite a execução de mais de um *web service* simultaneamente, utilizando o indicador de nível de paralelismo constante na descrição da composição. Para isso, é o utilizado o conceito de *thread*. Uma *thread* é uma unidade de execução concorrente que possui sua própria pilha de chamadas de métodos e seus parâmetros [Sun 2008]. Na plataforma Android, *threads* em uma mesma máquina virtual interagem e sincronizam através da utilização de objetos compartilhados [Android 2008].

A técnica utilizada por SmallSOA é a de disparar uma nova *thread* para cada tarefa de invocação de *web service*, independente de seu nível de paralelismo. Quando o XML da descrição da composição indicar que um *web service* não pode ser executado paralelamente com outro, o motor sincroniza essas execuções de modo o *web service* que deve ser executado posteriormente só seja executada após o término do anterior.

### 4.4.4 Tratar a possibilidade de desconexão

O motor trata eventuais desconexões ocorridas durante uma execução de uma composição de serviços. As desconexões podem ocorrer das seguintes formas:

- Queda do sinal de rede do usuário cliente;
- Queda do sinal de rede do motor SmallSOA;
- Queda do sinal de rede algum servidor de *web service*, móvel ou não;
- Indisponibilidade de algum servidor de *web service*, móvel ou não;
- Falta de bateria de algum dispositivo envolvido.

Para controlar desconexões, SmallSOA utiliza técnica similar a usada por servidores *web* para gravarem as sessões dos usuários logados em um sistema. Um servidor *web*, a partir do primeiro acesso do usuário ao sistema, cria um número de sessão único e mantém um registro com os valores das variáveis daquela e de todas as comunicações daquele usuário naquela sessão. SmallSOA, da mesma forma, utiliza os identificadores de usuário e senha informados em uma execução de composição e, juntamente com o identificador da composição, mantém registros dos valores das variáveis da execução. Diferentemente do do servidor *web*, não há comunicações adicionais entre cliente e SmallSOA. Caso caia a conexão, na próxima execução o cliente deve informar que quer continuar a partir da sessão anterior. Os parâmetros da execução são detalhados no capítulo seguinte.

# 5 CONSTRUÇÃO DO MOTOR SMALLSOA

Este capítulo descreve os aspectos referentes à implementação do motor para execução de composições de serviços em ambientes móveis SmallSOA. A seção 5.1 apresenta os aspectos arquiteturais de SmallSOA, desde sua relação externa com os demais componentes da arquitetura U-SOA até o relacionamento entre os componentes de seu núcleo. Na seção 5.2 é detalhado o WSDL do *web service* SmallSOA. Na seção 5.3 é apresentado o diagrama de classes da implementação do motor. A seção 5.4 descreve como foi feita a construção dos componentes. A seção 5.5, por sua vez, apresenta a modelagem do banco de dados utilizado em SmallSOA.

## 5.1 Aspectos arquiteturais

Arquiteturalmente, SmallSOA é um *middleware* dentro de uma arquitetura maior chamada U-SOA. Na pilha de componentes ilustrada na figura 5.1, SmallSOA está no quarto nível, logo abaixo do topo. A pilha de componentes se inicia pelo sistema operacional para dispositivos móveis Android. Logo acima existe uma abstração de *hardware* através da máquina virtual Dalvik, ainda pertencente à arquitetura Android. No terceiro nível aparece o um servidor de *web services* próprio da arquitetura U-SOA, chamado de SOAP *Server*. SmallSOA aparece logo em seguida, atuando como um servidor de composições de *web services* da arquitetura U-SOA. O topo da pilha é o nível de aplicação, onde aparecem as aplicações criadas a partir da composição de *web services*.

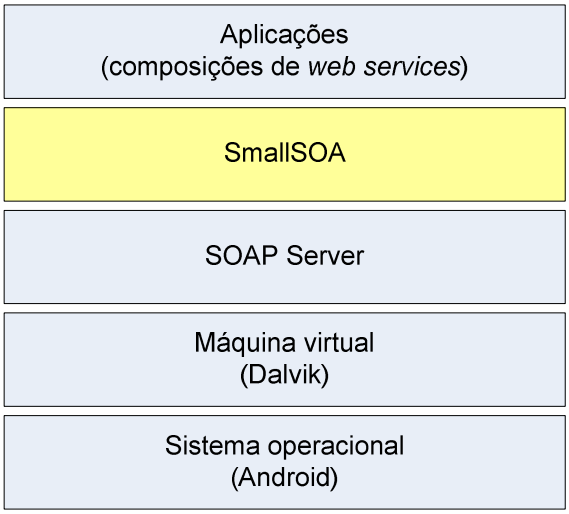


**Figura 5.1 – SmallSOA na pilha de componentes de U-SOA**

A abordagem de implementação de SmallSOA poderia se dar de inúmeras formas, sendo as principais as seguintes:

- *Software standalone* a ser chamado quando da necessidade de executar a composição de serviços;
- Servidor de composições de serviços nos mesmos moldes do servidor de serviços SOAP *Server*;
- Aplicativo implantado como um *service* exclusivo da arquitetura Android;
- *Web service* implantado no servidor SOAP *Server* incluído na arquitetura de U-SOA.


**Figura 5.2 – SOAP *Server* de U-SOA**

Optou-se por implementar SmallSOA como um *web service* . Desta forma, se tem um motor para execução de composições de serviços portátil que, customizado, poderia ser implantado em qualquer servidor de *web services* baseado em SOAP. A sua única restrição é que atualmente ele deve ser executado no sistema operacional Android. Essa restrição decorre

da arquitetura de U-SOA, onde SmallSOA está inserido, que utiliza esse sistema operacional. A figura 5.2 mostra a tela do servidor de *web services* SOAP *Server*, onde SmallSOA está implantado.

### 5.1.1 Integração com o ambiente externo

A integração de SmallSOA com o ambiente externo é bastante simples e ocorre basicamente através da comunicação do SOAP *Server* com os outros componentes de U-SOA, conforme ilustra a figura 5.3. Na figura é possível identificar uma aplicação se comunicando com o a interface do servidor SOAP *Server* através de requisições SOAP. A partir dessa interface, o SOAP *Server* direciona as mensagens para os respectivos *web services*. Após a execução, o *web service* retorna uma mensagem de resposta para o servidor, que encaminha para a aplicação que a requisitou.

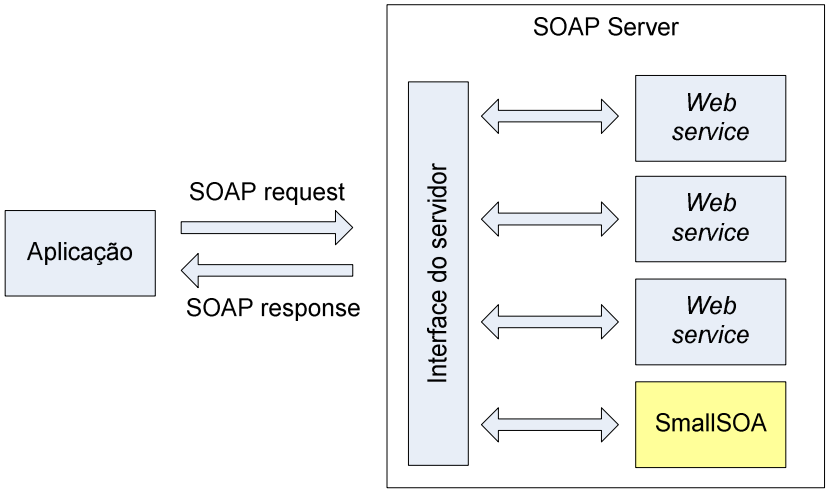


**Figura 5.3 – Comunicação de SmallSOA**

### 5.1.2 Núcleo do motor

Internamente, o motor SmallSOA permanece sempre no estado aguardando requisições, conforme ilustra a figura 5.4. A partir do recebimento de uma requisição SOAP na interface do servidor destinada ao *web service* SmallSOA, o SOAP *Server* encaminha-a para o seu destino. Após sua chegada, dentro de SmallSOA, a mensagem é direcionada a um dos três métodos existentes no *web service*, que são as funções executadas por ele: publicação, consulta ou de execução de uma composição de *web services*.

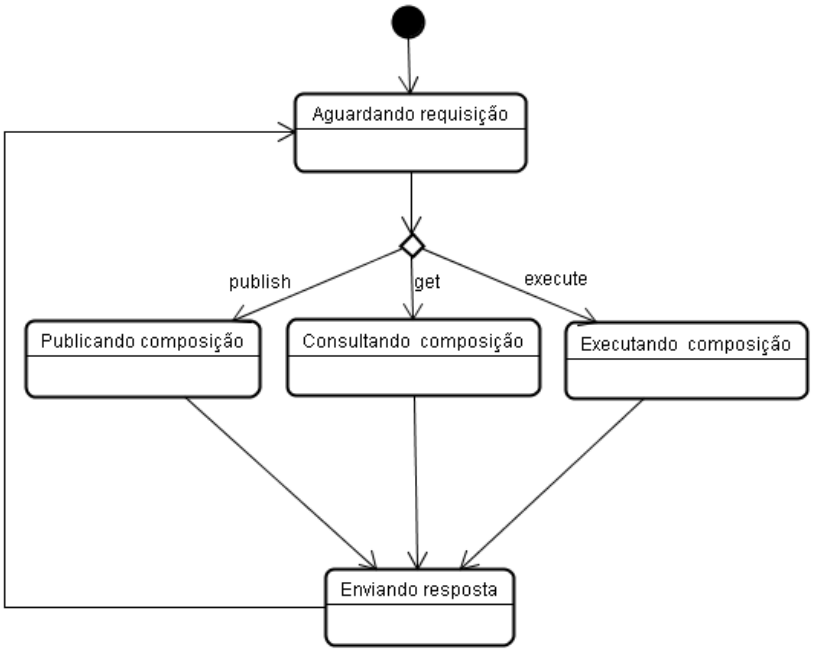


**Figura 5.4 – Estados de alto-nível de SmallSOA**

O fato de o motor permanecer no estado aguardando requisições não significa custo adicional para o dispositivo móvel. A utilização desse estado sem custo é conseqüência de SmallSOA ter sido implementado como um *web service* do SOAP *Server*. Este sim consome recursos para ficar aguardando as requisições. SmallSOA, por sua vez, permanece nesse estado utilizando o mesmo custo já contabilizado do SOAP *Server*.

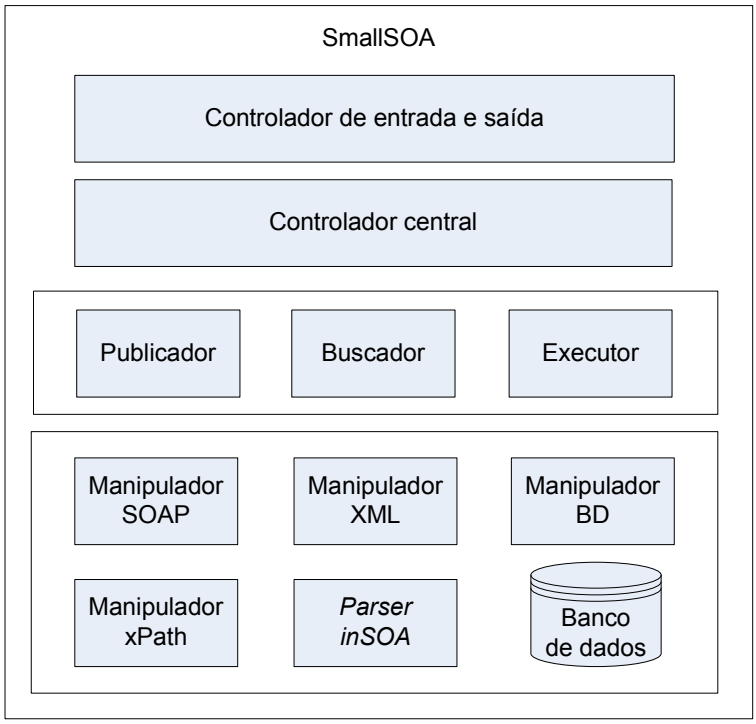


**Figura 5.5 – Núcleo de SmallSOA**

Os componentes internos do motor SmallSOA são apresentados na figura 5.5 e explicados a seguir.

- **Controlador de entrada e saída**. Responsável por receber a requisição direcionada pelo SOAP *Server* e encaminhá-la ao controlador central, na entrada, e por receber a mensagem do controlador central e encaminhá-la ao SOAP *Server*, na saída.
- **Controlador central**. Responsável por manipular as requisições em estado bruto e popular as estruturas internas do motor, preparando-o para a execução. No momento da entrada, identifica o método a ser executado pela requisição e direciona ao componente responsável. No momento da saída, direciona a mensagem de retorno ao controlador de entrada e saída.
- **Publicador**. Responsável pela publicação das composições de serviços no repositório de composições.
- **Buscador**. Responsável pela consulta das composições de serviços publicadas no repositório de composições.
- **Executor**. Responsável pela execução das composições de serviços. É o interpretador da descrição da composição em XML gerada pelo compilador da linguagem inSOA. Faz todo o gerenciamento da composição e procede à execução, de acordo a sua descrição, o que significa que é capaz de compreender e executar todos os comandos da composição, incluindo chamadas a *web services*, desvios de fluxo, iterações, etc.
- **Manipulador SOAP**. Componente responsável pela comunicação SOAP entre o motor SmallSOA e os *web services*.
- **Manipulador XML**. Componente responsável pela manipulação de XML dentro do motor.
- **Manipulador xPath**. Componente responsável pela execução de comandos xPath em determinados momentos nos XML's manipulados durante as execuções.
- ***Parser* inSOA**. Componente que transforma a descrição da composição em XML gerada pelo compilador da linguagem inSOA em estruturas computacionais internas ao motor.
- **Manipulador BD**. Componente responsável pela manipulação das bases de dados utilizadas pelo motor SmallSOA.

- **Banco de dados**. É o gerenciador físico do banco de dados, utilizado para diversas finalidades dentro do motor.

Na figura 5.5 não estão ilustrados os relacionamentos entre os componentes, mas apenas a listagem simples dos mesmos. A relação entre eles será demonstrada mais adiante em outros diagramas.

### 5.1.3 Benefícios da implementação de SmallSOA como *web service*

As principais vantagens de SmallSOA ter sido implementado como um *web service* são:

- Como SmallSOA preferencialmente necessitaria ficar escutando alguma porta, funcionando como servidor, implantando-o como *web service* em um servidor SOAP essa camada de comunicação não precisou ser implementada.
- SmallSOA tornou-se um servidor de composições de serviços portátil que poderia ser implantado em qualquer servidor de *web services* baseado em SOAP e no sistema operacional Android.
- *Web services* são baseados em um conjunto de padrões da Internet definidos pelo W3C e não requerem configurações especiais nos *firewalls* porque utilizam o protocolo HTTP para se comunicar.
- As aplicações podem facilmente se integrarem com SmallSOA, devido às características intrínsecas dos *web services*.
- SmallSOA pode fazer parte de composições de *web services*.

Por outro lado, é possível citar algumas desvantagens do fato de SmallSOA ter sido implementado como um *web service*:

- A performance dos *web services* costuma ser fraca, principalmente devido à constante manipulação de XML.
- Ao mesmo tempo em que a utilização de HTTP é uma vantagem, visto que permite passar por *firewalls* sem configurações adicionais, ela pode ser um problema já que os *firewalls* não evitariam eventuais ataques ao motor.
- O suporte a transações em *web services* ainda não está maduro o suficiente em comparação a outros padrões de computação distribuída.

## 5.2 WSDL do motor SmallSOA

Como SmallSOA foi implementado sob a forma de um *web service*, ele também pode ser descrito através de um WSDL, sendo que sua descrição completa encontra-se no apêndice A. A figura 5.6 apresenta o *web service* SmallSOA de uma forma ampla, podendo ser identificados o seu endereço, que no exemplo é *localhost*, e os seus três métodos: *Publish*, *Get* e *Execute*. Cada um dos métodos é explicado mais detalhadamente a partir da identificação de seus parâmetros de entrada e de saída, o que é feito a seguir.

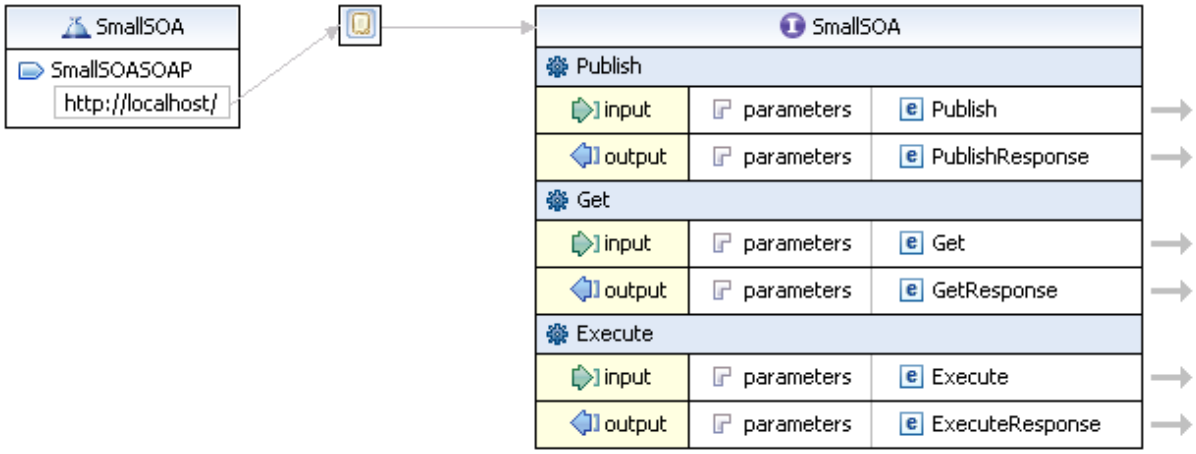


**Figura 5.6 – WSDL do motor SmallSOA**

### 5.2.1 *Publish*

O método *Publish* é o responsável pela publicação de uma composição de serviços descrita através da linguagem inSOA. A publicação compreende a armazenagem da descrição inSOA em um repositório interno do motor. Além da composição em si, informações de configuração e segurança recebidos como parâmetros de entrada do *web service* são armazenados.

O método *Publish* possui os seguintes parâmetros de entrada, todos eles do tipo *string*, ilustrados na figura 5.7:

- *idComposition*: identificação da composição;
- *idUser*: identificação do usuário;
- *Password*: senha do usuário;
- *Rule*: regras de acesso à composição;
- *inSOA*: descrição da composição utilizando a linguagem inSOA.

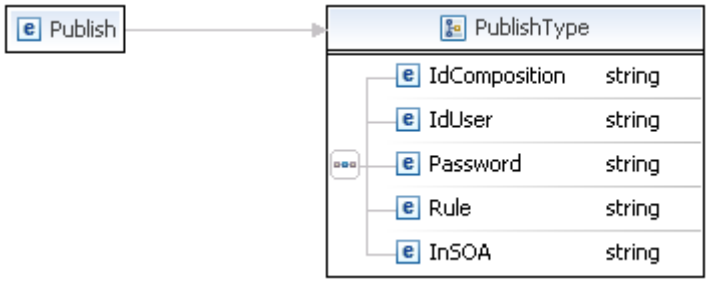


**Figura 5.7 – Parâmetros de entrada do método *Publish***

O único parâmetro de saída do método *Publish* é chamado de *Result* e também é do tipo *string*. O seu conteúdo pode ser uma das seguintes opções:

- *OK*: composição publicada com sucesso;
- *NOK*: não foi possível publicar a composição.

### 5.2.2 *Get*

O método *Get* é o responsável pela consulta a uma composição de serviços descrita através da linguagem inSOA e armazenada em SmallSOA. O método atualmente possui somente um parâmetro de entrada, do tipo *string*, conforme ilustra a figura 5.8:

- *idComposition*: identificação da composição.

A partir do parâmetro *idComposition* informado, SmallSOA busca a respectiva publicação no repositório e retorna para o usuário cliente. Em um segundo momento, será disponibilizada uma consulta através de *tags*, o que possibilitará ao usuário pesquisar as composições de determinada categoria. Por exemplo: finanças. Atualmente, essas *tags* já fazem parte da descrição da composição inSOA e também são armazenadas em um banco de dados interno de SmallSOA, sendo que cada composição pode ter mais de uma *tag*.


**Figura 5.8 – Parâmetros de entrada do método *Get***

O método *Get* possui somente um único parâmetro de saída, que é chamado de *inSOA*, e também é do tipo *string*. O seu conteúdo é a descrição da composição de serviços escrita em inSOA. Caso não haja uma composição armazenada em SmallSOA com o *idComposition* informado, o retorno será vazio.

### 5.2.3 *Execute*

O método *Execute* é o responsável pela execução de determinada composição de serviços previamente armazenada em SmallSOA. O método possui os seguintes parâmetros de entrada, ilustrados na figura 5.9:

- *idComposition*: identificação da composição; tipo *string*;
- *idUser*: identificação do usuário; tipo *string*;
- *Password*: senha do usuário; tipo *string*;
- *Timeout*: tempo em milisegundos que o usuário aceita esperar na chamada de cada *web service* da composição, antes do motor disparar uma exceção; tipo *int*;
- *NumberOfTries*: número de tentativas de chamada a cada *web service* da composição que o usuário deseja que o motor faça antes de disparar uma exceção; tipo *int*;
- *Parameters*: parâmetros da execução da composição de serviços; é um tipo especial composto de *n* repetições do par *Name* e *Value*, ambos tipo *string*.

O único parâmetro de saída do método *Execute* é chamado de *Result* e é do tipo *string*. O seu conteúdo pode conter o seguinte:

- Palavra *Denied*, que significa que o usuário cliente não possui acesso de execução à composição;
- XML contendo o retorno descrito na composição inSOA, com os valores obtidos através de seu processamento;
- Conteúdo vazio, caso tenha ocorrido falha no acesso aos *web services* da composição.

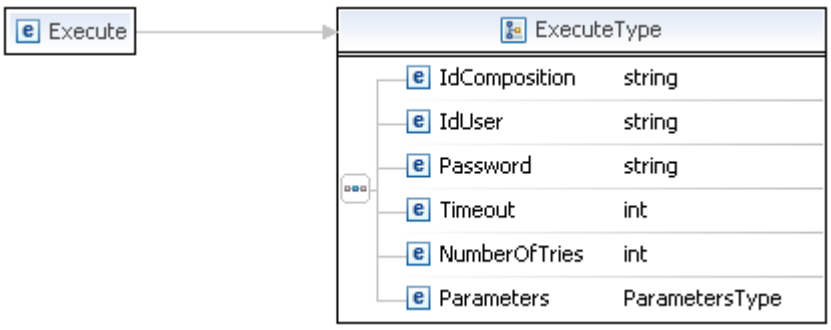


**Figura 5.9 – Parâmetros de entrada do método *Execute***

## 5.3 Diagrama de classes

Um diagrama de classes de projeto ilustra as especificações para classes de *software* e interfaces em uma aplicação [Larman 2004]. A informação típica inclui classes, associações, atributos, métodos, entre outros. Na figura 5.10 é ilustrado de forma resumida o diagrama de classes de SmallSOA, apresentando as principais classes implementadas. Para facilitar a leitura, foram removidas do diagrama classes, atributos e métodos que não são relevantes para a compreensão das funcionalidades principais de SmallSOA. Pela figura, é possível identificar o seguinte:

- Classe principal *SmallSOA*.
- Classes responsáveis pela manipulação das três funções intrínsecas do motor:
  - Publicação de composições de serviço: *PublishComposition*;
  - Consulta de composições de serviço: *GetComposition*;
  - Execução de composições de serviço: *ExecuteComposition*.
- Classe responsável pela implementação de paralelismo através de *threads*: *ExecuteThread*.
- Classe responsável pela construção da resposta do motor: *SmallSOAResponse*.
- Classes responsáveis pela manipulação do XML da composição inSOA:
  - Manipulação do XML da composição inSOA: *InSOAParser*;
  - Manipulação do *tag* do XML <inSOA>: *InSOA*;
  - Manipulação do *tag* do XML <Invoke>: *Invoke*;
  - Manipulação do *tag* do XML <Parameter>: *Parameter*;
  - Manipulação do *tag* do XML <Where>: *Where*;
  - Manipulação do *tag* do XML <Return>: *Return*;
  - Manipulação do *tag* do XML <Fail>: *Fail*.
- Classes auxiliares:
  - Manipulação do histórico: *Historic*;
  - Manipulação de banco de dados: *DataBase*.
- Pacotes auxiliares:
  - Manipulação de envelopes SOAP: *KSOAP2*;
  - Manipulação de XML: *kXML2*;
  - Manipulação de xPath: *MiniXPath*.

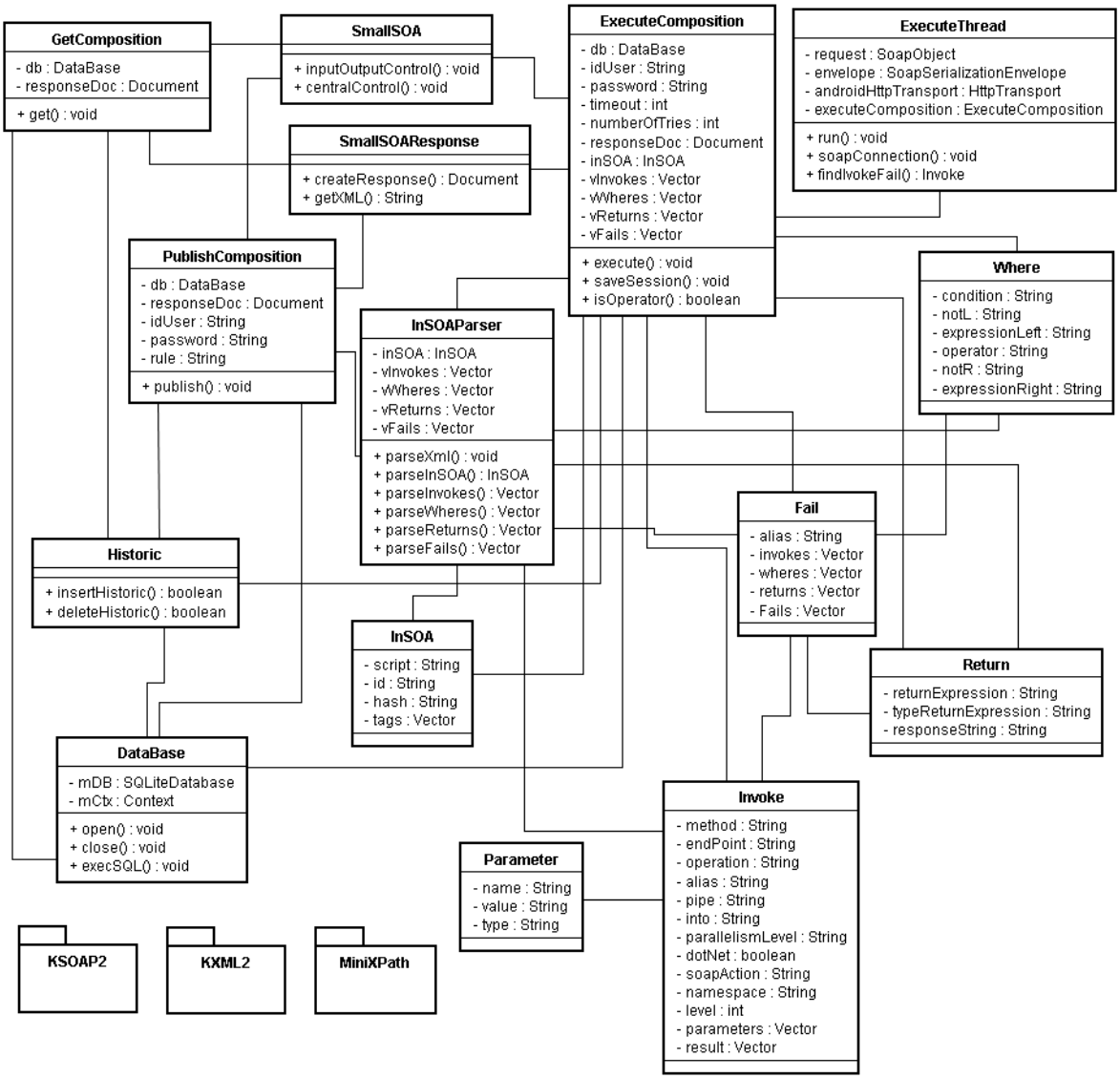


**Figura 5.10 – Diagrama de classes de SmallSOA**

## 5.4 Implementação do motor

A implementação do motor para execução de composições de serviços para ambientes móveis procedeu das seguintes formas:

1. Construção de novos componentes de *software*;
2. Utilização de componentes de *software* existentes;
3. Customização de componentes de *software* existentes.

Portanto, pode-se considerar o motor SmallSOA como uma coletânea de tecnologias para dispositivos móveis, organizadas e arquiteturadas com o objetivo de executar composições de *web services* nesses dispositivos.

A seguir são explicados cada um dos componentes do motor. Primeiramente, porém, é apresentado o ambiente de desenvolvimento utilizado.

### 5.4.1 Ambiente de desenvolvimento

Para construir o motor SmallSOA foi utilizada a seguinte estrutura de *hardware* e *software*:

- Processador Intel Pentium M 1,73GHz;
- 1GB de memória RAM;
- Sistema operacional Microsoft Windows XP *Professional* versão 2002 *Service Pack* 2;
- Linguagem de programação Java;
- Android SDK Windows versão1.0 *release* 1;
- Plataforma Eclipse Europa versão 3.3.0 com Java *Runtime Environment* versão 1.6.0.11;
- Servidor de *web services* customizado a partir do JME SOAP *Server* versão 0.1.1.

### 5.4.2 Controlador de entrada e saída

Componente implementado como um método na classe principal *SmallSOA* que implementa uma fachada entre o ambiente externo e o controlador central.

### 5.4.3 Controlador central

Também implementado como um método na classe principal *SmallSOA*, a principal função deste componente é atuar como um tradutor da mensagem de entrada, direcionando-a para o respectivo método interno do motor.

### 5.4.4 Publicador

O componente Publicador foi mapeado para a classe *PublishComposition*, cujo algoritmo funciona da seguinte forma:

1. Recebe o XML com os parâmetros do método *Publish* do *web service* SmallSOA.

2. Manipula o XML da mensagem, obtendo o identificador da composição, o identificador do usuário, a senha do usuário, a regra de acesso e a descrição XML da composição inSOA.
3. Manipula o XML de inSOA, obtendo o seu identificador, as *tags* e outras informações da composição.
4. Verifica se composição existe no repositório:
    1. Se a composição não existe, são incluídos registros nas seguintes tabelas: *composition*, *tag*, *user* e *permission*.
    2. Se a composição existe, são alteradas as seguintes tabelas: *composition*, *tag*, *user* e *permission*.
5. Insere um registro na tabela *historic*, informando o resultado da execução: OK ou NOK.
6. Utilizando o componente *SmallSOAResponse*, é criada uma resposta para o usuário cliente.

A figura 5.11 ilustra um trecho do código da classe.

```
InSOAParser parser = new InSOAParser();
parser.parseXml(compositionDescription, "PublishComposition");
String id = parser.getInSOA().getId();
Vector<String> tags = parser.getInSOA().getTags();

// If composition already exists, then update it. Else insert it.
sql = new String("select * from composition where idComposition = '" + id +
"'");
Cursor cursor = db.fetchElementSQL(sql);
if (cursor.getCount()>0) {
      sql = "update composition set insoa = \"" + compositionDescription +
"\", date = \"" + c.getTime().toString() + "\" where idComposition = \"" +
id + "\"";
} else {
      sql = "insert into composition (idComposition, insoa, date) values
(\"" + id + "\", \"" + compositionDescription + "\", \"" +
c.getTime().toString() + "\")";
}
cursor.close();
db.execSQL(sql);
```

**Figura 5.11 – Trecho de código da classe *PublishComposition***

A figura 5.12 ilustra o retorno do componente publicador, recebido por um usuário cliente de SmallSOA.


**Figura 5.12 – Retorno do componente publicador**

### 5.4.5 Buscador

O componente Buscador foi mapeado para a classe *GetComposition*, cujo algoritmo funciona da seguinte forma:

1. Recebe o XML com os parâmetros do método *Get* do *web service* SmallSOA.
2. Manipula o XML da mensagem, obtendo o identificador da composição.
3. Busca a composição do repositório a partir de seu identificador recebido como parâmetro.
4. Insere um registro na tabela *historic*, informando o resultado da execução: OK ou NOK.

5. Utilizando o componente *SmallSOAResponse*, é criada uma resposta para o usuário cliente.
    1. Se a composição existe no repositório, ela é retornada ao usuário cliente.
    2. Se a composição não existe no repositório, é retornado vazio ao usuário cliente.

A figura 5.13 ilustra um trecho do código da classe.

```
Calendar c = Calendar.getInstance();

String sql = new String("select * from composition where idComposition = '"
+ id + "'");
Cursor cursor = db.fetchElementSQL(sql);

if (cursor.moveToFirst()) {
      responseString =
cursor.getString(cursor.getColumnIndexOrThrow("insoa"));

      sql = "insert into historic (idComposition, method, result, date)
values (\"" + id + "\", \"GetComposition\", \"OK\", \"" +
c.getTime().toString() + "\")";
      db.execSQL(sql);
} else {
      sql = "insert into historic (idComposition, method, result, date)
values (\"" + id + "\", \"GetComposition\", \"NOK\", \"" +
c.getTime().toString() + "\")";
      db.execSQL(sql);
}
cursor.close();

// Create XML response
SoaEngineResponse soaResponse = new SoaEngineResponse();
responseDoc = soaResponse.createResponse("GetComposition", id,
responseString);
```

**Figura 5.13 – Trecho de código da classe *GetComposition***

A figura 5.14 ilustra o retorno do componente buscador, recebido por um usuário cliente de SmallSOA.


**Figura 5.14 – Retorno do componente buscador**

### 5.4.6 Executor

O componente Executor foi mapeado para a classe *ExecuteComposition*, cujo algoritmo executa da seguinte forma:

1. Recebe o XML com os parâmetros do método *Execute* do *web service* SmallSOA.
2. Manipula o XML da mensagem, obtendo o identificador da composição, o identificador do usuário, a senha do usuário, o tempo de espera, o número de tentativas e os parâmetros da composição inSOA.

3. Busca a composição do repositório a partir de seu identificador recebido como parâmetro.
4. Manipula o XML de inSOA, obtendo os seguintes dados:
    1. Informações da composição: *tag* <inSOA>;
    2. Vetor de chamadas de *web services*: *tag* <Invoke>;
    3. Vetor de manipulação de variáveis e dos retornos dos *web services*: *tag* <Where>;
    4. Vetor de manipulação dos retornos ao usuário cliente: *tag* <Return>;
    5. Vetor de exceções: *tag* <Fail>.
5. Executa efetivamente a composição de *web services*.
    1. Executa *web services* de forma paralela, quando possível;
    2. Salva as sessões de execução permitindo a continuação caso haja alguma desconexão.
6. Insere um registro na tabela *historic*, informando o resultado da execução: OK ou NOK.
7. Utilizando o componente *SmallSOAResponse*, é criada uma resposta para o usuário cliente, contendo a especificação contida na *tag* <Return>.

A figura 5.15 ilustra um trecho do código da classe.

```
request = new SoapObject(oInvoke.getNamespace(), oInvoke.getOperation());

Vector<Parameter> parameters = oInvoke.getParameters();
for (int j=0; j<parameters.size(); j++) {
      request.addProperty(parameters.elementAt(j).getName(),
parameters.elementAt(j).getValue());
}

envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.dotNet = oInvoke.isDotNet();
envelope.setOutputSoapObject(request);
//HttpTransport androidHttpTransport = new
HttpTransport(oInvoke.getEndPoint().toLowerCase());
androidHttpTransport = new
HttpTransport(oInvoke.getEndPoint().toLowerCase());
androidHttpTransport.debug = true;

boolean invokeDone = true;

// ---------- Begin call with timeout
// Create a timer task that closes the connection
TimerTask task = new TimerTask() { public void run()
{androidHttpTransport.reset(); } };

try {
      new Timer().schedule(task, timeout); // milliseconds
      androidHttpTransport.call(oInvoke.getSoapAction(), envelope);
      task.cancel(); // cancel the timeout task

} catch (InterruptedIOException e) {
```

**Figura 5.15 – Trecho de código da classe *ExecuteComposition***

A figura 5.16 ilustra o retorno do componente executor, recebido por um usuário cliente de SmallSOA.

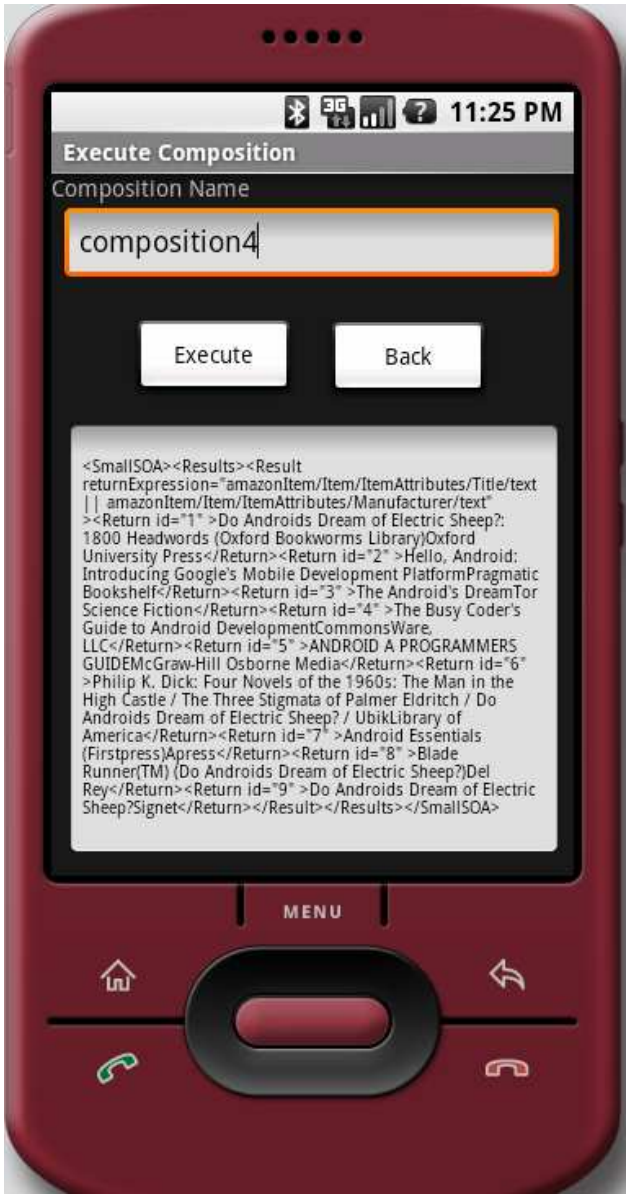


**Figura 5.16 – Retorno do componente executor**

### 5.4.7 Manipulador SOAP

O manipulador SOAP foi mapeado para o pacote *KSOAP2*. Esse pacote contém as classes da conhecida e leve biblioteca Java para dispositivos móveis kSOAP2 [Haustein 2006]. kSOAP2 suporta HTTP e HTTPS, ambos com ou sem autenticação básica. A versão do protocolo SOAP suportadas pela biblioteca são 1.0, 1.1 e 1.2. O manipulador SOAP de SmallSOA foi implementado utilizando a versão 1.1.

kSOAP2, no entanto, não funciona no Android, visto que o SDK do Android não suporta todas as classes contidas na plataforma JME, faltando, entre outros, alguns

componentes utilizados por ela. Dessa forma, foi necessário fazer algumas alterações para que fosse possível utilizá-la.

### 5.4.8 Manipulador XML

O componente manipulador XML foi mapeado para o pacote de classes *KXML2*, que contém a também conhecida biblioteca Java para dispositivos móveis kXML2 [Haustein 2005].

### 5.4.9 Manipulador xPath

O manipulador xPath foi mapeado para o pacote de classes *MiniXPath*, que foi implementado a partir da customização da biblioteca minixpath [Cremarenco 2003]. Com esse componente é feita a navegação pelos resultados das execuções dos *web services*, até que se encontre a variável solicitada.

```
− <soap:Envelope>
  − <soap:Body>
    − <GetQuoteResponse>
      − <GetQuoteResult>
          <StockQuotes><Stock><Symbol>GOOG</Symbol><Last>433.75</Last><Date>9/11/2008</Date>
          <Time>4:00pm</Time><Change>+19.59</Change><Open>408.93</Open><High>435.09</High>
          <Low>406.38</Low><Volume>6442813</Volume><MktCap>136.4B</MktCap>
          <PreviousClose>414.16</PreviousClose><PercentageChange>+4.73%</PercentageChange>
          <AnnRange>409.68 - 747.24</AnnRange><Earns>15.222</Earns><P-E>27.21</P-
          E><Name>GOOGLE</Name></Stock></StockQuotes>
        </GetQuoteResult>
      </GetQuoteResponse>
    </soap:Body>
  </soap:Envelope>
```

**Figura 5.17 – Envelope SOAP de retorno de um *web service* com resultado em XML**

No exemplo ilustrado pela figura 5.17, é possível visualizar o envelope SOAP de retorno da execução de um *web service* de cotações de ações da bolsa de valores americana. No exemplo, para encontrar o valor da última cotação das ações do Google (GOOG), deve-se utilizar a seguinte instrução xPath, que retornaria o valor 433.75:

/Envelope/Body/GetQuoteResponse/GetQuoteResult/StockQuotes/Stock/Last

Utilizou-se xPath para a manipulação dos resultados dos *web services* em substituição ao tipo de dado SOAP *Object*, próprio da biblioteca kSOAP2. Embora SOAP *Object* seja a opção indicada pela documentação, o retorno de determinados *web services* ainda

necessitariam a utilização de xPath ou outra técnica para encontrar valores dentro de um XML, como é o caso da figura 5.17. Diferentemente do que ocorre em retornos de *web service* mais simples, como o exemplo apresentado na figura 5.18, onde o valor de retorno é o número 1.818.

```
- <soap:Envelope>
  - <soap:Body>
    - <ConversionRateResponse>
        <ConversionRateResult>1.818</ConversionRateResult>
      </ConversionRateResponse>
    </soap:Body>
  </soap:Envelope>
```

**Figura 5.18 – Envelope SOAP de retorno de um *web service* com resultado direto**

Além desse aspecto, o motor SmallSOA possui uma peculiaridade em relação a outras aplicações que executam *web services*: é um motor de execução padrão, que não tem conhecimento prévio do que irá executar (*on the fly*). SmallSOA deve suportar todo o tipo de retorno de *web service*, desde os mais simples, como o apresentado na figura 5.18, até os mais complexos, como o apresentado na figura 5.17 ou aquele que consulta a loja *online* Amazon, por exemplo. Portanto, para permitir maior flexibilidade, optou-se pela utilização de um manipulador xPath.

### 5.4.10 *Parser* inSOA

O *Parser* inSOA foi mapeado para a classe *InSOAParser* e é utilizado pelas classes *PublishComposition* e *ExecuteComposition*. Como o próprio nome diz, esse componente faz o *parser* do XML da linguagem inSOA para sua utilização dentro do motor. *Parser* inSOA utiliza as seguintes estruturas auxiliares armazenar o conteúdo do XML:

- Classe *InSOA* para a *tag* <inSOA>;
- Classe *Invoke* para a *tag* <Invoke>;
- Classe *Parameter* para a *tag* <Parameter>;
- Classe *Where* para a *tag* <Where>;
- Classe *Return* para a *tag* <Return>;
- Classe *Fail* para a *tag* <Fail>.

### 5.4.11 Manipulador BD

Para simplificar a construção do motor, o componente manipulador BD foi implementado em uma única classe para manipular toda a base de dados de SmallSOA, chamada de *DataBase*. Manipulador BD relaciona-se com a classe *Historic* e com as três classes responsáveis pelas funções intrínsecas do motor: *PublishComposition*, *GetComposition* e *ExecuteComposition*.

## 5.5 Modelagem do banco de dados

SmallSOA faz uso de um banco de dados interno que serve como repositório das composições de serviço e também para outros controles próprios do motor. O banco de dados escolhido é o SQLite [SQLite 2008], opção de *software* livre que já vem embutido na plataforma Android.

O SQLite é um motor de banco de dados SQL embarcado. Diferentemente da maioria dos bancos de dados SQL, não possui um processo servidor separado. SQLite foi criado para ser extremamente simples e ao invés de competir com outros grandes bancos de dados, compete com ferramentas de manipulação de arquivos. Um banco de dados SQL completo, com múltiplas tabelas, índices, gatilhos (*triggers*) e visões está contido dentro de um simples arquivo no disco. É importante citar que SQLite é um banco de dados “sem tipos”. Na prática, SQLite utiliza tipagem dinâmica, sendo que o conteúdo pode ser armazenado como *integer*, *real*, *text*, *blob*, ou como *null*. Devido à tipagem dinâmica, esses tipos definidos na criação de uma tabela são apenas uma “dica” do que é esperado em determinada coluna, ou seja, não são tipos restritivos. Por exemplo, SQLite permite incluir uma *string* em uma coluna definida como *integer*.

A modelagem da base de dados do motor SmallSOA contempla seis tabelas, apresentadas na figura 5.19.

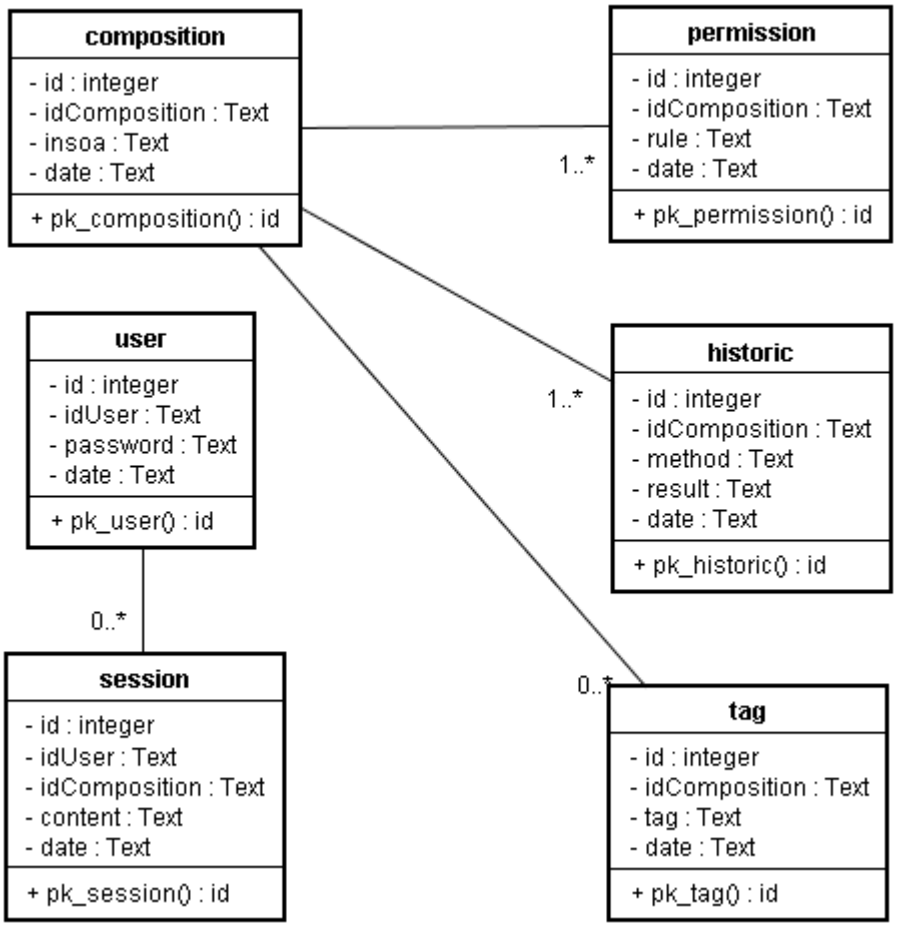


**Figura 5.19 – Modelagem do banco de dados**

Todas as cinco tabelas possuem uma chave-primária chamada *id*, que é um inteiro autoincrementável e serve apenas para identificar unicamente os registros. O restante do conteúdo das tabelas da modelagem de SmallSOA é detalhado abaixo:

- *composition*. Tabela que contém as composições de serviço. É o próprio repositório de composições de serviço inSOA. Contém o identificador da composição, recebido como parâmetro do usuário cliente, a composição inSOA em formato XML, também recebida como parâmetro no momento da publicação da composição, e a data e hora da publicação.
- *tag*. Tabela que contém as *tags* das composições de serviço, para uma posterior busca por categorias. Contém o identificador da composição, recebido como parâmetro do usuário cliente, o nome da *tag*, também recebida como parâmetro no momento da publicação da composição, e a data e hora da publicação. A tabela *tag* possui uma relação 0 para *n* com a tabela *composition*, ou seja, é uma parâmetro opcional na publicação de uma composição.

- *permission*. Tabela que contém as permissões das composições de serviço. Contém o identificador da composição e a regra, recebidos como parâmetros do usuário cliente, a composição inSOA em formato XML, também recebida como parâmetro no momento da publicação da composição, e a data e hora da publicação.
- *historic*. Tabela que contém o histórico da utilização do moto, funcionando como um *log*. Contém o identificador da composição, recebido como parâmetro do usuário cliente, o método que foi executado e o resultado da execução, além data e hora do acontecimento.
- *user*. Tabela que contém os usuários donos das composições, armazenados no momento da publicação das mesmas. Contém o identificador do usuário e a sua senha (*password*), recebidos como parâmetros do usuário cliente no momento da publicação da composição, e a data e hora de atualização dos dados do usuário.
- *session*. Tabela que contém os dados das sessões de execução de uma composição. Contém o identificador da sessão, o identificador do usuário, o identificador da composição, o conteúdo da sessão e a data e hora da atualização da sessão.

# 6 RESULTADOS

Neste capítulo são apresentados os resultados obtidos a partir da execução do motor SmallSOA. A seção 6.1 mostra a metodologia utilizada para a a validação de SmallSOA. A seção 6.2 apresenta os resultados obtidos com a validação simples do motor. Já na seção 6.3 e 6.4 são descritos dois cenários de teste e os respectivos resultados, que visam validar as funcionalidades do motor SmallSOA, bem como de parte da arquitetura U-SOA. Para finalizar, a seção 6.5 apresenta uma visão geral dos resultados e um comparativo com os trabalhos relacionados.

## 6.1 Metodologia

A primeira validação de SmallSOA é o próprio término de sua implementação, visto que esta se deu por finalizada somente quando os requisitos e características descritos anteriormente foram cumpridos. Isso significa que SmallSOA está funcional dentro da arquitetura completa para suportar ambientes colaborativos ubíqüos U-SOA, que está sendo construída. Destacam-se a integração com outros dois trabalhos: a linguagem de composição de serviços inSOA e a ferramenta para análise de impacto na composição de serviços gImpact.

A metodologia para a validação do trabalho inclui os seguintes itens:

- Validação simples utilizando composições de serviços criadas a partir de *web services* profissionais de um mesmo fornecedor, visando ao controle de tempos e à comparação dos resultados entre eles. Testa as seguintes características do motor SmallSOA:
  - Execução paralela de *web services*;
  - Tratamento de desconexão.
- Definição e execução de dois cenários de teste que utilizam composições com uma quantidade razoável de *web services* e de composições de *web services*. Testa as seguintes características do motor SmallSOA:
  - Repositório de composições de serviços;
  - Chamadas a *web services* e composições de *web services*;
  - Interpretação da linguagem inSOA;

  - Suporte ao conjunto de padrões de composição previstos em inSOA.
- Construção de uma aplicação cliente para consumir o *web service* SmallSOA e apresentar os resultados na tela do emulador do Android.
- Comparação das características de SmallSOA com os trabalhos relacionados, sem análise comparativa de desempenho devido às seguintes diferenças:
  - Plataformas;
  - Linguagens de composição de serviços;
  - Funcionalidades.

## 6.2 Validação simples

A validação simples do motor SmallSOA força sua execução buscando testar certos aspectos de sua construção. Para que isso seja possível, é indispensável a utilização de um ambiente controlado e confiável. Por isso, optou-se pela utilização de composições de serviços criadas a partir dos *web services* profissionais da Amazon [Amazon 2008], que estão sempre disponíveis. Para a validação simples de SmallSOA, a descrição da composição em si não é o principal. O que importa, nesse momento, é o sucesso na validação de aspectos específicos do motor, tais como tempos de execução, execução paralela de *web services* e tratamento de desconexão, entre outros. O *web service* da Amazon utilizado para esta validação é o Amazon *Associates Web Service*, cuja descrição encontra-se no seguinte endereço:

http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl?

As sub-seções a seguir apresentam os resultados da validação simples detalhadamente.

### 6.2.1 Composição com um único *web service*

O primeiro teste foi feito utilizando-se uma composição de serviços descrita conforme a figura 6.1. Essa composição invoca um único serviço, que é o *web service* da loja *online* Amazon, passando os seguintes parâmetros, que já estão transcritos no XML:

- ***AWSAccessKeyId***. Chave de acesso para o *web service*, previamente cadastrada. Valor: número da chave de acesso.
- ***SearchIndex***. Seleciona a parte do conteúdo da Amazon que se deseja pesquisar. Valor: livros (*Books*).

- ***Sort***. Informa a forma de ordenação dos resultados. Valor: nível de relevância (*relevancerank*).
- ***ItemPage***. Página de resultados que será retornada pelo *web service*. Valor: 1 (somente a primeira página dos resultados será retornada).
- ***Keywords***. Palavras-chaves utilizadas na consulta à base de dados da Amazon. Valor: android (livros que contenham a palavra chave android).

```
- <inSOA id="composition4" tags="finance, money" hash="67dd30dfa352b3ea329e2fbb762bf034abb5e50e">
    <inSOAScript/>
  - <Invokes>
    - <Invoke Method="SOAP" EndPoint="http://soap.amazon.com/onca/soap?Service=AWSECommerceService"
      Operation="ItemSearch" Alias="a" Pipe="/Envelope/Body/ItemSearchResponse/Items/Item"
      Into="amazonItem" ParallelismLevel="1">
      - <Parameters>
          <Parameter Name="AWSAccessKeyId" Value="07Z9SS44NH4X1XS7NHR2"
          Type="">teste</Parameter>
          <Parameter Name="SearchIndex" Value="Books" Type=""/>
          <Parameter Name="Sort" Value="relevancerank" Type=""/>
          <Parameter Name="ItemPage" Value="1" Type=""/>
          <Parameter Name="Keywords" Value="android" Type=""/>
        </Parameters>
      </Invoke>
    </Invokes>
  - <Wheres>
      <Where Condition="" NotL="" ExpressionLeft="amazonItem/Item/ItemAttributes/Manufacturer/text"
      Operator="NE" NotR="" ExpressionRight="Wrox"/>
    </Wheres>
  - <Returns>
      <Return ReturnExpression="amazonItem/Item/ItemAttributes/Title/text || amazonItem/Item/ItemAttributes
      /Manufacturer/text" TypeReturnExpression=""/>
    </Returns>
    <Fails> </Fails>
  </inSOA>
```

**Figura 6.1 – Composição com um único *web service***

A referida composição de serviços foi executada dez vezes consecutivas, sendo que os tempos de sua execução são apresentados na tabela 6.1. A tabela apresenta os horários iniciais e finais de cada execução, bem como o respectivo tempo de execução, que é calculado a partir da diferença entre os horários. Ao final é apresentada a média dos tempos de execução, que ficou em 2,107 segundos. Também se encontram destacados na tabela os seguintes valores:

- Tempo da primeira execução: 2,369 segundos;
- Menor tempo de execução: 1,529 segundos;
- Maior tempo de execução: 4,521 segundos.

O primeiro tempo de execução encontra-se grifado na tabela devido à peculiaridade própria de uma execução inicial. Há, nesses casos, um consumo de tempo um pouco maior para colocação das estruturas do motor na memória. Como são feitas dez execuções consecutivas, as demais possuem essa vantagem em relação à primeira. O *log* completo das execuções está disponível no apêndice B.

**Tabela 6.1 – Tempos de execução da composição com um único *web service***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 00:38:30,394 | 00:38:32,763 | **2,369** |
| **2** | 00:38:38,073 | 00:38:39,782 | 1,709 |
| **3** | 00:38:41,363 | 00:38:43,152 | 1,789 |
| **4** | 00:38:44,663 | 00:38:46,352 | 1,689 |
| **5** | 00:38:47,683 | 00:38:49,502 | 1,819 |
| **6** | 00:38:50,823 | 00:38:52,352 | **1,529** |
| **7** | 00:38:53,582 | 00:38:55,353 | 1,771 |
| **8** | 00:38:56,672 | 00:38:58,483 | 1,811 |
| **9** | 00:38:59,703 | 00:39:01,762 | 2,059 |
| **10** | 00:39:03,012 | 00:39:07,533 | **4,521** |
| **Média de tempo de execução** | | | **2,107** |

### 6.2.2 Composição com quatro *web services* sem paralelismo

No segundo teste foi executada uma composição de serviços que invoca quatro *web services* da Amazon, similares ao ilustrado pela figura 6.1. Novamente, a referida composição de serviços foi executada dez vezes consecutivas, sendo que os tempos de sua execução são apresentados na tabela 6.2, que apresenta os horários iniciais e finais de cada execução, bem como o respectivo tempo de execução, que é calculado a partir da diferença entre os horários. Ao final é apresentada a média dos tempos de execução, que ficou em 7,731 segundos. Encontram-se destacados na tabela os seguintes valores:

- Tempo da primeira execução: 8,470 segundos;
- Menor tempo de execução: 6,060 segundos;
- Maior tempo de execução: 12,010 segundos.

**Tabela 6.2 – Tempos de execução da composição com quatro *web services* sem paralelismo**

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 00:42:49,302 | 00:42:57,772 | **8,470** |
| **2** | 00:43:03,263 | 00:43:10,453 | 7,190 |
| **3** | 00:43:13,053 | 00:43:19,478 | 6,425 |
| **4** | 00:43:21,433 | 00:43:27,633 | 6,200 |
| **5** | 00:43:29,923 | 00:43:37,763 | 7,840 |
| **6** | 00:43:41,362 | 00:43:48,123 | 6,761 |
| **7** | 00:43:52,123 | 00:43:58,183 | **6,060** |
| **8** | 00:44:00,403 | 00:44:06,858 | 6,455 |
| **9** | 00:44:09,013 | 00:44:18,912 | 9,899 |
| **10** | 00:44:21,383 | 00:44:33,393 | **12,010** |
| **Média de tempo de execução** | | | **7,731** |

Os parâmetros da invocação do *web service* SmallSOA, *Timeout* e *NumberOfTries*, foram, respectivamente, 2000 e 2, o que significa que o motor deveria aguardar 2000 milisegundos e fazer 2 tentativas de execução de cada *web service* participante da composição. Devido a falhas nas conexões, foram encontrados alguns problemas nas execuções, relacionados justamente a *timeouts* e a número de tentativas esgotadas. O *log* completo das execuções está disponível no apêndice C. Por sua vez, os tempos de execução são ilustrados graficamente na figura 6.2.

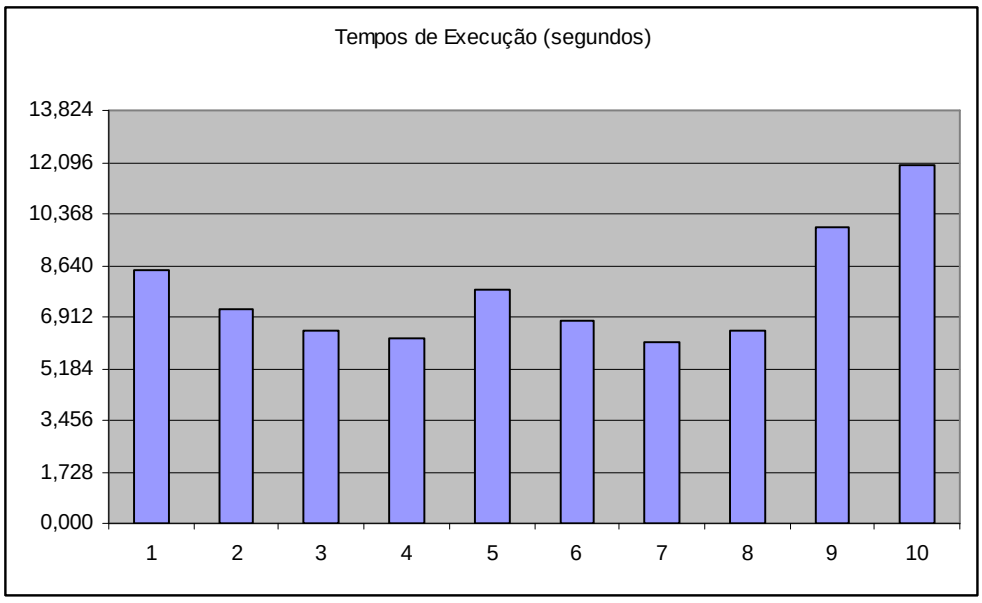


**Figura 6.2 – Tempos de execução da composição com quatro *web services* sem paralelismo**

Esta validação testa uma composição de serviços com chamadas a quatro *web services* sem utilização de paralelismo, ou seja, foi simulado que cada invocação a *web service* deveria

ocorrer após o término da invocação do *web service* anterior. A tabela 6.3 apresenta os tempos de execução de cada *web service* referentes à primeira execução de composição simulada. Os *web services* são identificados pelos *aliases* “a”, “b”, “c” e “d”, juntamente com uma marca de início e final de execução. É identificado também o início e final de execução da composição como um todo. Fica visível, na tabela, que cada *web service* foi executado de forma não paralela, somente iniciando a execução do seguinte após o término da execução do anterior.

**Tabela 6.3 – Tempos da primeira execução sem paralelismo**

| # | Horário inicial | Horário final |
|---|---|---|
| **1** | 00:42:49,302 | ------ INICIO ------ |
| **2** | 00:42:49,359 | RUN INI: a |
| **3** | 00:42:51,673 | RUN FIM: a |
| **4** | 00:42:51,685 | RUN INI: b |
| **5** | 00:42:53,392 | RUN FIM: b |
| **6** | 00:42:53,413 | RUN INI: c |
| **7** | 00:42:55,892 | RUN FIM: c |
| **8** | 00:42:55,902 | RUN INI: d |
| **9** | 00:42:57,772 | RUN FIM: d |
| **10** | 00:42:57,772 | ------ FIM ------ |

A tabela 6.4 apresenta as mesmas informações da tabela 6.3, mas de forma que seja possível comparar os tempos de execução gastos com cada *web service* e também o tempo total gasto pela composição.

**Tabela 6.4 – Comparação dos tempos da primeira execução sem paralelismo**

| Execução | Horário inicial | Horário final | Tempo gasto (s) |
|---|---|---|---|
| ***Web service* a** | 00:42:49,359 | 00:42:51,673 | 2,314 |
| ***Web service* b** | 00:42:51,685 | 00:42:53,392 | 1,707 |
| ***Web service* c** | 00:42:53,413 | 00:42:55,892 | 2,479 |
| ***Web service* d** | 00:42:55,902 | 00:42:57,772 | 1,870 |
| **Composição** | 00:42:49,302 | 00:42:57,772 | 8,470 |

De acordo com a tabela, o tempo total efetivamente gasto pela composição de *web services* foi um pouco maior do que a soma dos tempos gastos por cada *web service*, que foi de 8,370 segundos.

### 6.2.3 Composição com quatro *web services* com paralelismo

No terceiro teste foi executada uma composição de serviços que invoca quatro *web services* da Amazon, similares ao ilustrado pela figura 6.1. Mais uma vez, a referida composição de serviços foi executada dez vezes consecutivas, sendo que os tempos de sua execução são apresentados na tabela 6.5, que apresenta os horários iniciais e finais de cada execução, bem como o respectivo tempo de execução, que é calculado a partir da diferença entre os horários. Ao final é apresentada a média dos tempos de execução, que ficou em 5,437 segundos. Encontram-se destacados na tabela os seguintes valores:

- Tempo da primeira execução: 5,180 segundos;
- Menor tempo de execução: 4,009 segundos;
- Maior tempo de execução: 6,900 segundos.

**Tabela 6.5 – Tempos de execução da composição com quatro *web services* com paralelismo**

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 00:45:13,353 | 00:45:18,533 | **5,180** |
| **2** | 00:45:21,813 | 00:45:28,713 | **6,900** |
| **3** | 00:45:30,603 | 00:45:35,543 | 4,940 |
| **4** | 00:45:37,393 | 00:45:42,233 | 4,840 |
| **5** | 00:45:44,293 | 00:45:48,302 | **4,009** |
| **6** | 00:45:50,303 | 00:45:56,593 | 6,290 |
| **7** | 00:45:58,813 | 00:46:04,803 | 5,990 |
| **8** | 00:46:06,343 | 00:46:11,822 | 5,479 |
| **9** | 00:46:14,363 | 00:46:19,612 | 5,249 |
| **10** | 00:46:21,733 | 00:46:27,222 | 5,489 |
| **Média de tempo de execução** | | | **5,437** |

Da mesma forma que a validação anterior, os parâmetros da invocação do *web service* SmallSOA, *Timeout* e *NumberOfTries*, foram, respectivamente, 2000 e 2. Devido a falhas nas invocações dos *web services* foram encontrados alguns problemas nas execuções das composições, também relacionados a *timeouts* e a número de tentativas esgotadas, como na validação anterior. O *log* completo das execuções está disponível no apêndice D. Já a figura 6.3 ilustra graficamente os tempos das dez execuções da composição de serviços.

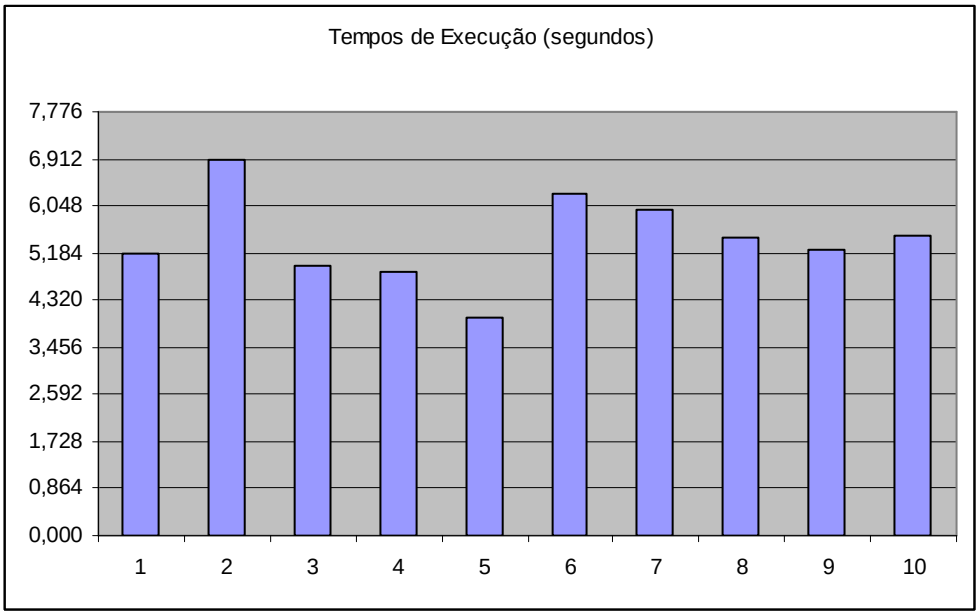


**Figura 6.3 – Tempos de execução da composição com quatro *web services* com paralelismo**

Esta validação testa uma composição de serviços com chamadas a quatro *web services* com utilização de paralelismo. A composição executada é exatamente igual ao da validação anterior, com exceção ao nível de paralelismo dos *web services*. Mais especificamente, os *web services* cujos *aliases* são "a" e "b" possuem o nível de paralelismo 1, já os *web services* cujos *aliases* são "c" e "d" possuem o nível de paralelismo 2. Isso significa que os *web services* "a" e "b" são executados paralelamente e, quando terminarem, os *web services* "c" e "d" são executados, também paralelamente. A tabela 6.6 apresenta os tempos de execução de cada *web service* referentes à primeira execução de composição simulada. Fica visível, na tabela, o paralelismo de execução entre os *web services* "a" e "b" e entre os *web services* "c" e "d", que são disparados simultaneamente dentro dos grupos definidos pelo nível de paralelismo.

**Tabela 6.6 – Tempos da primeira execução com paralelismo**

| # | Horário inicial | Horário final |
|---|---|---|
| **1** | 00:45:13,353 | ------ INICIO ------ |
| **2** | 00:45:13,363 | RUN INI: a |
| **3** | 00:45:13,382 | RUN INI: b |
| **4** | 00:45:15,823 | RUN FIM: b |
| **5** | 00:45:15,873 | RUN FIM: a |
| **6** | 00:45:15,894 | RUN INI: c |
| **7** | 00:45:16,052 | RUN INI: d |
| **8** | 00:45:18,512 | RUN FIM: c |
| **9** | 00:45:18,523 | RUN FIM: d |
| **10** | 00:45:18,533 | ------ FIM ------ |

A tabela 6.7 apresenta as mesmas informações da tabela 6.6, mas de forma que seja possível comparar os tempos de execução gastos com cada *web service* e também o tempo total gasto pela composição.

**Tabela 6.7 – Comparação dos tempos da primeira execução com paralelismo**

| Execução | Horário inicial | Horário final | Tempo gasto (s) |
|---|---|---|---|
| ***Web service* a** | 00:45:13,363 | 00:45:15,873 | 2,510 |
| ***Web service* b** | 00:45:13,382 | 00:45:15,823 | 2,441 |
| ***Web service* c** | 00:45:15,894 | 00:45:18,512 | 2,618 |
| ***Web service* d** | 00:45:16,052 | 00:45:18,523 | 2,471 |
| **Composição** | 00:45:13,353 | 00:45:18,533 | 5,180 |

De acordo com a tabela, o tempo total efetivamente gasto pela composição de *web services* foi bem menor do que a soma dos tempos gastos por cada *web service*, que foi de 10,040 segundos, chegando a ser quse 50% menor. Esse tempo confirma a expectativa de redução pretendida pela execução paralela de dois grupos de dois *web services* cada. Nesta validação, ocorreu situação exatamente oposta à verificada na validação anterior, onde o tempo total gasto pela composição foi maior do que a soma dos tempos gastos por cada *web service*.

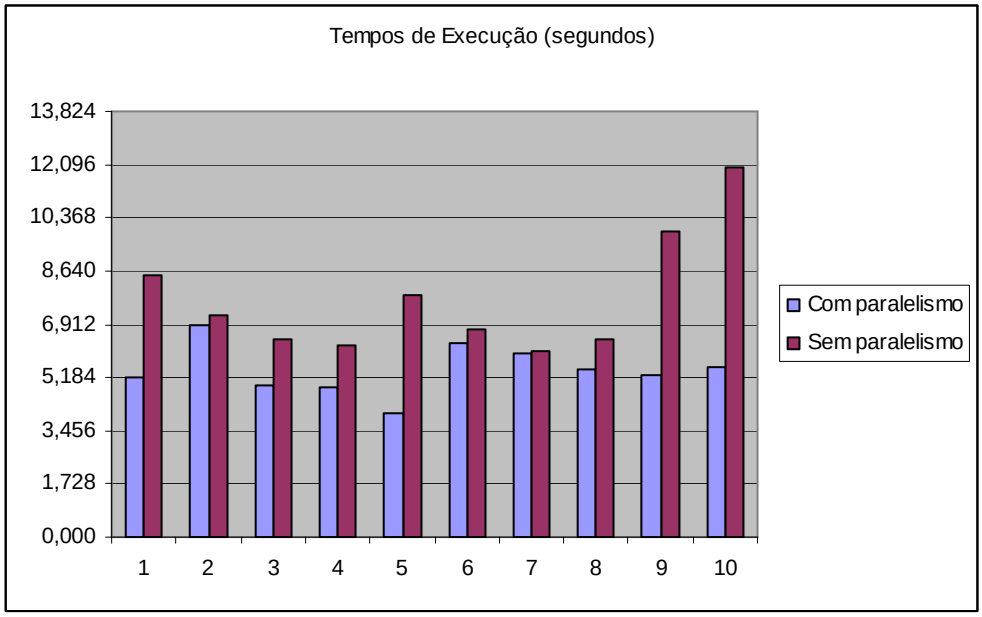


**Figura 6.4 – Comparação dos tempos de execução com e sem paralelismo**

Neste exemplo, para efeito de simulação, o nível de paralelismo foi definido manualmente no XML. Em uma situação real, no entanto, o nível de paralelismo dos *web services* participantes de uma composição é gerado pelo compilador da linguagem de composição inSOA, que gera o XML executado por SmallSOA com essa informação. Para encontrar os níveis de paralelismo, o compilador leva em conta os pré-requisitos de cada *web service* e os padrões de composição utilizados.

A partir da comparação dos tempos de execução da mesma composição de serviços, com e sem paralelismo, apresentada na figura 6.4, é possível comprovar que todas as dez execuções foram mais rápidas naquela que previa essa característica.

## 6.3 Cenários de teste

Os dois cenários de teste que auxiliam na validação deste trabalho foram feitos em conjunto com outros dois trabalhos que estão sendo implementados por nosso grupo de pesquisa, conforme ilustra a figura 6.5. O seu fluxo de funcionamento é o seguinte:

1. Usuário edita uma composição de serviços escrita através da linguagem inSOA;
2. Usuário compila o *script* através de uma ferramenta do pacote inSOA;
3. Usuário ajusta eventuais erros;
4. Através de uma ferramenta do pacote inSOA, o usuário gera um código XML referente à composição de serviços;
5. Usuário publica a composição em SmallSOA;
6. Usuário busca a composição em SmallSOA;
7. Usuário executa a composição em SmallSOA;
8. A ferramenta gImpact localiza a composição em SmallSOA e efetua suas análises;
9. A ferramenta gImpact executa a composição em SmallSOA para coletar os tempos de execução;
10. A partir das análises feitas pela ferramenta gImpact, o usuário altera o *script* inSOA e o fluxo recomeça.

O foco dos testes do ponto de vista de SmallSOA que é descrito neste trabalho é o item 7, referente à execução de uma composição de serviços. Dessa forma, dois cenários e

seus respectivos resultados referentes à execução de composições são apresentados nas próximas subseções.

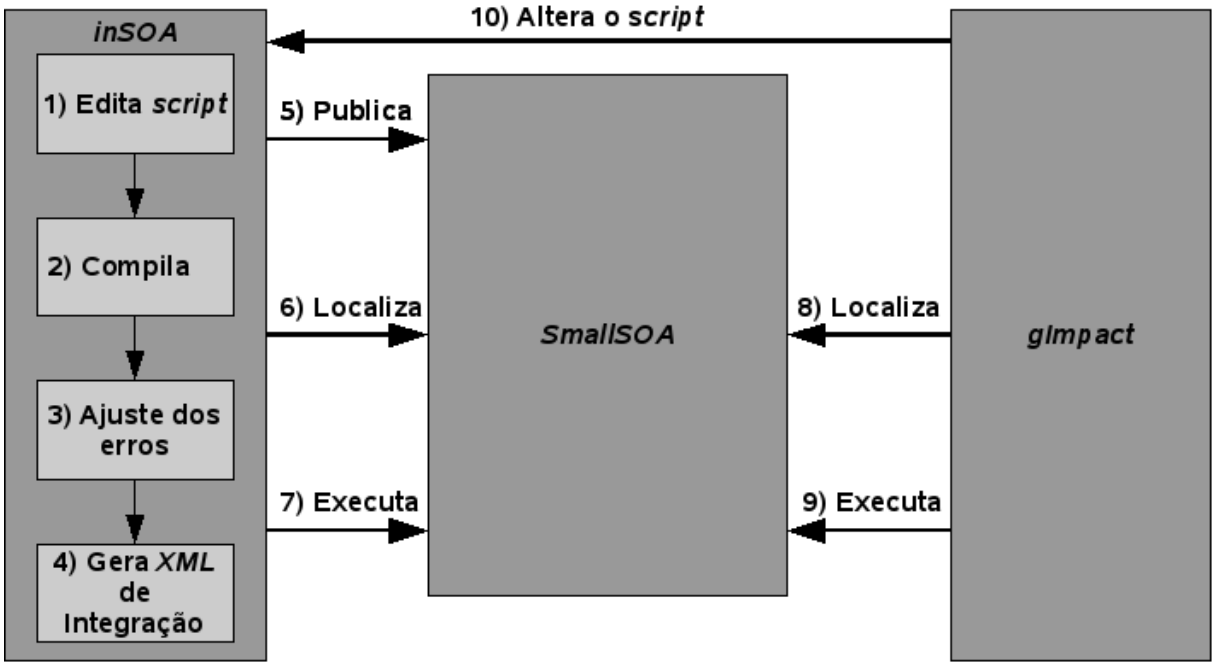


**Figura 6.5 – Fluxo dos cenários de teste**

### 6.3.1 Primeiro cenário: informações sobre viagens

O primeiro cenário de teste é ilustrado na figura 6.6 e é composto de um conjunto de *web services* e composições de *web services* orquestrados de forma tal que retornem informações sobre viagens. Neste exemplo, o usuário trabalha com informações sobre o país ao qual deseja viajar, incluindo informações sobre o tempo, horário, moeda e a cidade destino. O usuário trabalha também com informações sobre o aeroporto onde irá pousar e com a validação do endereço destino.

Neste cenário, foram processadas as seguintes composições de serviços, escritas através da linguagem inSOA:

1. *getInformationTravel*;
2. *getInformationCountry*;
3. *getAirportInformation*;
4. *validateAddress*;
5. *getInformationWeather*;
6. *getInformationGMT*;

7. *getInformationMoney*;
8. *getInformationUK*;
9. *validateUS*;
10. *validateUK*.

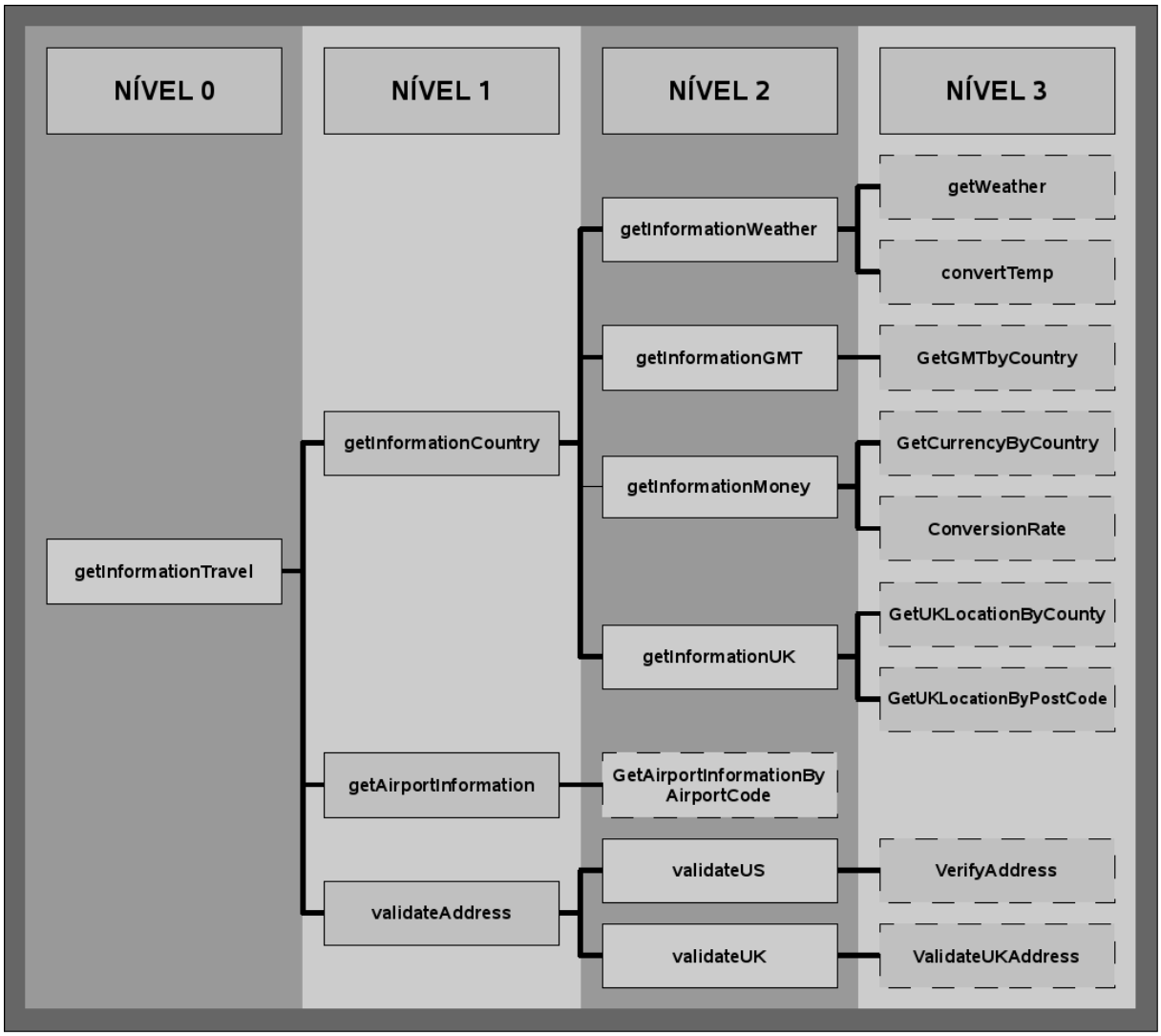


**Figura 6.6 – Primeiro cenário de teste**

A tabela 6.8 apresenta os resultados obtidos em dez execuções da composição *getInformationWeather*, que inclui os *web services getWeather* e *convertTemp*.

**Tabela 6.8 – Resultado da composição *getInformationWeather***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:06:45,233 | 01:06:50,273 | 5,040 |
| **2** | 01:06:54,903 | 01:06:59,483 | 4,580 |
| **3** | 01:07:01,193 | 01:07:03,304 | 2,111 |
| **4** | 01:07:04,473 | 01:07:09,643 | 5,170 |
| **5** | 01:07:10,743 | 01:07:21,445 | 10,702 |
| **6** | 01:07:22,413 | 01:07:30,423 | 8,010 |
| **7** | 01:07:31,562 | 01:07:36,443 | 4,881 |
| **8** | 01:07:37,723 | 01:07:41,323 | 3,600 |
| **9** | 01:07:43,053 | 01:07:54,013 | 10,960 |
| **10** | 01:07:55,123 | 01:08:03,363 | 8,240 |
| **Média de tempo de execução** | | | **6,329** |

A tabela 6.9 apresenta os resultados obtidos em dez execuções da composição *getInformationGMT*, que inclui o *web service getGMTByCountry*.

**Tabela 6.9 – Resultado da composição *getInformationGMT***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:11:40,683 | 01:11:41,873 | 1,190 |
| **2** | 01:11:43,243 | 01:11:44,113 | 0,870 |
| **3** | 01:11:45,023 | 01:11:46,074 | 1,051 |
| **4** | 01:11:46,873 | 01:11:47,713 | 0,840 |
| **5** | 01:11:48,503 | 01:11:49,473 | 0,970 |
| **6** | 01:11:50,283 | 01:11:51,213 | 0,930 |
| **7** | 01:11:52,043 | 01:11:52,954 | 0,911 |
| **8** | 01:11:53,853 | 01:11:54,683 | 0,830 |
| **9** | 01:11:55,573 | 01:11:56,563 | 0,990 |
| **10** | 01:11:58,053 | 01:11:58,982 | 0,929 |
| **Média de tempo de execução** | | | **0,951** |

A tabela 6.10 apresenta os resultados obtidos em dez execuções da composição *getInformationMoney*, que inclui os *web services getCurrencyByCountry* e *convertionRate*.

**Tabela 6.10 – Resultado da composição *getInformationMoney***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:12:25,943 | 01:12:27,703 | 1,760 |
| **2** | 01:12:28,693 | 01:12:30,773 | 2,080 |
| **3** | 01:12:31,663 | 01:12:33,994 | 2,331 |
| **4** | 01:12:35,023 | 01:12:37,663 | 2,640 |
| **5** | 01:12:38,663 | 01:12:40,823 | 2,160 |
| **6** | 01:12:41,723 | 01:12:43,533 | 1,810 |
| **7** | 01:12:44,613 | 01:12:46,413 | 1,800 |
| **8** | 01:12:47,273 | 01:12:49,263 | 1,990 |
| **9** | 01:12:50,053 | 01:12:51,934 | 1,881 |
| **10** | 01:12:52,753 | 01:12:54,873 | 2,120 |
| **Média de tempo de execução** | | | **2,057** |

A tabela 6.11 apresenta os resultados obtidos em dez execuções da composição *getInformationUK*, que inclui os *web services getUKLocationByCountry* e *getUKLocationByPostCode*.

**Tabela 6.11 – Resultado da composição *getInformationUK***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:13:26,983 | 01:13:33,553 | 6,570 |
| **2** | 01:13:34,643 | 01:13:40,143 | 5,500 |
| **3** | 01:13:41,213 | 01:13:47,103 | 5,890 |
| **4** | 01:13:48,593 | 01:13:54,053 | 5,460 |
| **5** | 01:13:55,093 | 01:13:57,043 | 1,950 |
| **6** | 01:14:00,283 | 01:14:02,273 | 1,990 |
| **7** | 01:14:03,273 | 01:14:08,752 | 5,479 |
| **8** | 01:14:09,723 | 01:14:11,673 | 1,950 |
| **9** | 01:14:12,743 | 01:14:18,413 | 5,670 |
| **10** | 01:14:19,513 | 01:14:21,473 | 1,960 |
| **Média de tempo de execução** | | | **4,242** |

A tabela 6.12 apresenta os resultados obtidos em dez execuções da composição *validateUS*, que inclui o *web service verifyAddress*.

**Tabela 6.12 – Resultado da composição *validateUS***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:14:59,373 | 01:15:00,193 | 0,820 |
| **2** | 01:15:01,153 | 01:15:01,943 | 0,790 |
| **3** | 01:15:02,893 | 01:15:03,883 | 0,990 |
| **4** | 01:15:04,834 | 01:15:05,965 | 1,131 |
| **5** | 01:15:06,663 | 01:15:07,713 | 1,050 |
| **6** | 01:15:08,443 | 01:15:09,797 | 1,354 |
| **7** | 01:15:10,543 | 01:15:11,764 | 1,221 |
| **8** | 01:15:12,523 | 01:15:13,533 | 1,010 |
| **9** | 01:15:14,223 | 01:15:15,265 | 1,042 |
| **10** | 01:15:16,023 | 01:15:17,343 | 1,320 |
| **Média de tempo de execução** | | | **1,073** |

A tabela 6.13 apresenta os resultados obtidos em dez execuções da composição *validateUK*, que inclui o *web service validateUKAddress*.

**Tabela 6.13 – Resultado da composição *validateUK***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:15:54,333 | 01:15:55,342 | 1,009 |
| **2** | 01:15:56,164 | 01:15:57,103 | 0,939 |
| **3** | 01:15:58,063 | 01:15:58,744 | 0,681 |
| **4** | 01:15:59,624 | 01:16:00,610 | 0,986 |
| **5** | 01:16:01,443 | 01:16:02,473 | 1,030 |
| **6** | 01:16:03,253 | 01:16:04,173 | 0,920 |
| **7** | 01:16:04,843 | 01:16:05,757 | 0,914 |
| **8** | 01:16:06,663 | 01:16:07,493 | 0,830 |
| **9** | 01:16:08,173 | 01:16:09,253 | 1,080 |
| **10** | 01:16:10,103 | 01:16:11,113 | 1,010 |
| **Média de tempo de execução** | | | **0,940** |

A tabela 6.14 apresenta os resultados obtidos em dez execuções da composição *getInformationCountry*, que inclui as composições de *web services getInformationWeather*, *getInformationGMT*, *getInformationMoney* e *getInformationUK*.

**Tabela 6.14 – Resultado da composição *getInformationCountry***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:16:52,743 | 01:17:09,823 | 17,080 |
| **2** | 01:17:10,823 | 01:17:25,984 | 15,161 |
| **3** | 01:17:26,343 | 01:17:51,553 | 25,210 |
| **4** | 01:18:21,753 | 01:18:37,963 | 16,210 |
| **5** | 01:18:48,623 | 01:19:15,933 | 27,310 |
| **6** | 01:19:19,433 | 01:19:39,073 | 19,640 |
| **7** | 01:19:43,763 | 01:20:04,793 | 21,030 |
| **8** | 01:20:07,833 | 01:20:33,493 | 25,660 |
| **9** | 01:20:43,173 | 01:20:56,153 | 12,980 |
| **10** | 01:21:03,833 | 01:21:20,423 | 16,590 |
| **Média de tempo de execução** | | | **19,687** |

A tabela 6.15 apresenta os resultados obtidos em dez execuções da composição *getAirportInformation*, que inclui o *web service getAirportInformationByAirportCode*.

**Tabela 6.15 – Resultado da composição *getAirportInformation***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:22:31,083 | 01:22:32,222 | 1,139 |
| **2** | 01:22:34,843 | 01:22:35,656 | 0,813 |
| **3** | 01:22:36,163 | 01:22:37,123 | 0,960 |
| **4** | 01:22:38,543 | 01:22:39,353 | 0,810 |
| **5** | 01:22:40,753 | 01:22:41,717 | 0,964 |
| **6** | 01:22:42,653 | 01:22:43,664 | 1,011 |
| **7** | 01:22:44,703 | 01:22:45,547 | 0,844 |
| **8** | 01:22:46,343 | 01:22:47,303 | 0,960 |
| **9** | 01:22:48,133 | 01:22:49,043 | 0,910 |
| **10** | 01:22:49,963 | 01:22:51,033 | 1,070 |
| **Média de tempo de execução** | | | **0,948** |

A tabela 6.16 apresenta os resultados obtidos em dez execuções da composição *validateAddress*, que inclui as composições de *web services validateUS* e *validateUK*.

**Tabela 6.16 – Resultado da composição *validateAddress***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:23:15,813 | 01:23:17,603 | 1,790 |
| **2** | 01:23:17,653 | 01:23:27,467 | 9,814 |
| **3** | 01:23:29,053 | 01:23:39,153 | 10,100 |
| **4** | 01:23:40,333 | 01:23:45,223 | 4,890 |
| **5** | 01:23:52,953 | 01:23:58,143 | 5,190 |
| **6** | 01:24:07,243 | 01:24:09,182 | 1,939 |
| **7** | 01:24:09,343 | 01:24:13,683 | 4,340 |
| **8** | 01:24:24,083 | 01:24:29,803 | 5,720 |
| **9** | 01:24:31,713 | 01:24:33,323 | 1,610 |
| **10** | 01:24:38,713 | 01:24:41,033 | 2,320 |
| **Média de tempo de execução** | | | **4,771** |

A tabela 6.17 apresenta os resultados obtidos em dez execuções da composição principal *getInformationTravel*, que inclui as composições de *web services* *getInformationCountry* e *getAirportInformation* e *validateAddress*.

**Tabela 6.17 – Resultado da composição *getInformationTravel***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 01:25:11,023 | 01:25:36,253 | 25,230 |
| **2** | 01:25:48,543 | 01:26:23,173 | 34,630 |
| **3** | 01:26:32,903 | 01:26:59,393 | 26,490 |
| **4** | 01:29:01,413 | 01:29:28,173 | 26,760 |
| **5** | 01:29:46,323 | 01:30:10,743 | 24,420 |
| **6** | 01:31:13,023 | 01:31:35,543 | 22,520 |
| **7** | 01:32:32,423 | 01:33:01,280 | 28,857 |
| **8** | 01:33:54,883 | 01:34:28,834 | 33,951 |
| **9** | 01:34:36,263 | 01:35:01,863 | 25,600 |
| **10** | 01:35:03,943 | 01:35:33,403 | 29,460 |
| **Média de tempo de execução** | | | **27,792** |

### 6.3.2 Segundo cenário: pesquisa de livros

O segundo cenário de teste é ilustrado na figura 6.7 e é composto de um conjunto de *web services* e composições de *web services* orquestrados para que efetuem uma pesquisa de livros. Neste exemplo, o usuário manipula informações sobre os livros que deseja adquirir, incluindo informações específicas dos livros e os respectivos valores convertidos para a sua moeda; valida as suas credenciais de comprador, incluindo validação do cartão de crédito e de seus dados pessoais; verifica as informações financeiras no SERASA, incluindo validação de CPF, de cheque e das questões legais.

Neste cenário, foram processadas as seguintes composições de serviços, escritas através da linguagem inSOA:

1. *getBooks*;
2. *getInformationBooks*;
3. *validateCredentials*;
4. *validateFinancial*;
5. *getBook*;
6. *sumValueBooks*;
7. *validateCardInformation*;
8. *validatePersonalInformation*;
9. *validateCPF*;
10. *validateCheck*;
11. *validateLawsuits*;
12. *validateInformationSERASA*.

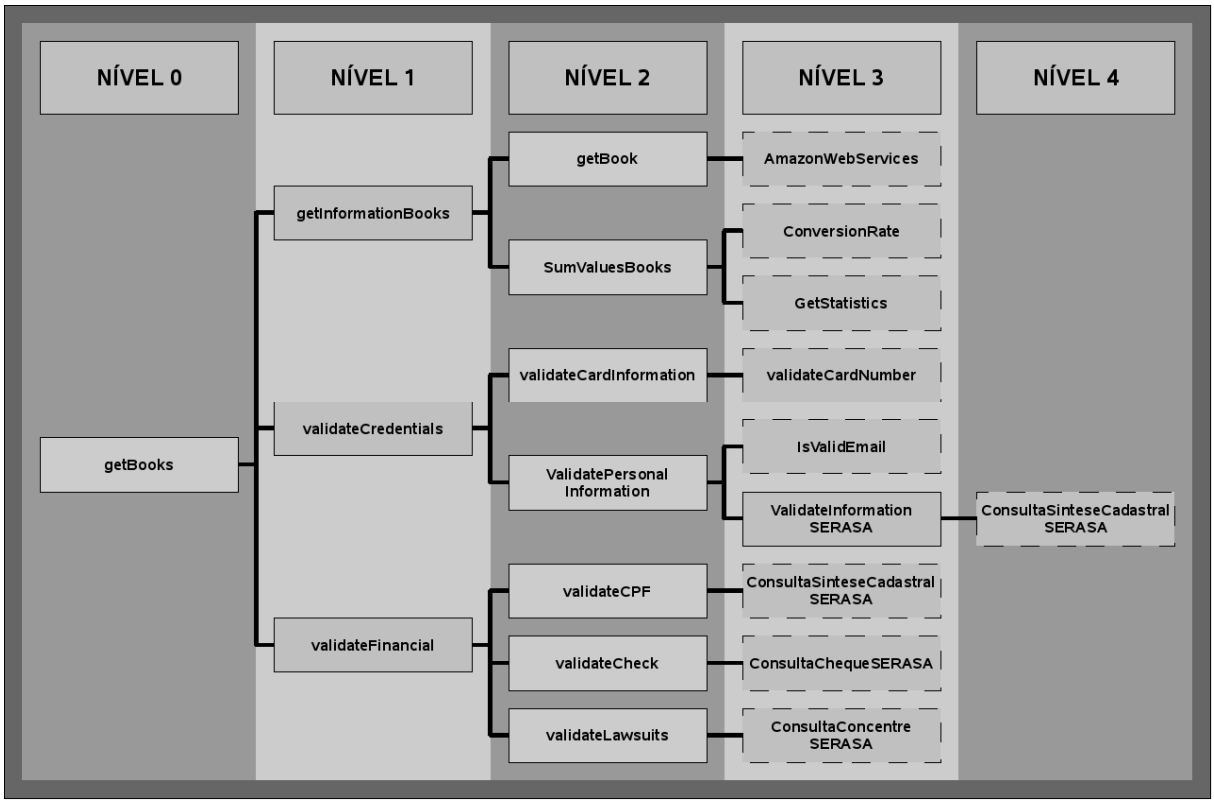


**Figura 6.7 – Segundo cenário de teste**

A tabela 6.18 apresenta os resultados obtidos em dez execuções da composição *validateInformationSERASA*, que inclui o *web service ConsultaSinteseCadastralSERASA*.

**Tabela 6.18 – Resultado da composição *validateInformationSERASA***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:22:44,073 | 03:22:45,653 | 1,580 |
| **2** | 03:22:49,073 | 03:22:52,013 | 2,940 |
| **3** | 03:22:52,933 | 03:22:54,003 | 1,070 |
| **4** | 03:22:54,863 | 03:22:55,792 | 0,929 |
| **5** | 03:22:56,673 | 03:22:57,703 | 1,030 |
| **6** | 03:22:58,563 | 03:22:59,613 | 1,050 |
| **7** | 03:23:02,303 | 03:23:03,423 | 1,120 |
| **8** | 03:23:04,243 | 03:23:05,183 | 0,940 |
| **9** | 03:23:06,763 | 03:23:07,703 | 0,940 |
| **10** | 03:23:29,253 | 03:23:30,273 | 1,020 |
| **Média de tempo de execução** | | | **1,262** |

A tabela 6.19 apresenta os resultados obtidos em dez execuções da composição *getBook*, que inclui o *web service AmazonWebServices*.

**Tabela 6.19 – Resultado da composição *getBook***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:25:26,503 | 03:25:28,493 | 1,990 |
| **2** | 03:25:35,333 | 03:25:36,357 | 1,024 |
| **3** | 03:25:37,643 | 03:25:38,603 | 0,960 |
| **4** | 03:25:39,743 | 03:25:41,092 | 1,349 |
| **5** | 03:25:42,433 | 03:25:43,763 | 1,330 |
| **6** | 03:25:45,093 | 03:25:46,373 | 1,280 |
| **7** | 03:25:47,273 | 03:25:48,642 | 1,369 |
| **8** | 03:25:49,783 | 03:25:50,993 | 1,210 |
| **9** | 03:25:52,092 | 03:25:53,243 | 1,151 |
| **10** | 03:25:54,203 | 03:25:55,553 | 1,350 |
| **Média de tempo de execução** | | | **1,301** |

A tabela 6.20 apresenta os resultados obtidos em dez execuções da composição *sumValueBooks*, que inclui os *web services ConversionRate* e *GetStatistics*.

**Tabela 6.20 – Resultado da composição *sumValueBooks***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:26:21,913 | 03:26:23,223 | 1,310 |
| **2** | 03:26:24,523 | 03:26:25,743 | 1,220 |
| **3** | 03:26:26,884 | 03:26:28,153 | 1,269 |
| **4** | 03:26:29,013 | 03:26:30,263 | 1,250 |
| **5** | 03:26:31,153 | 03:26:32,313 | 1,160 |
| **6** | 03:26:33,513 | 03:26:34,722 | 1,209 |
| **7** | 03:26:35,683 | 03:26:36,993 | 1,310 |
| **8** | 03:26:38,303 | 03:26:39,769 | 1,466 |
| **9** | 03:26:41,183 | 03:26:42,383 | 1,200 |
| **10** | 03:26:43,953 | 03:26:45,226 | 1,273 |
| **Média de tempo de execução** | | | **1,267** |

A tabela 6.21 apresenta os resultados obtidos em dez execuções da composição *validateCardInformation*, que inclui o *web service validateCardNumber*.

**Tabela 6.21 – Resultado da composição *validateCardInformation***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:27:37,543 | 03:27:38,763 | 1,220 |
| **2** | 03:27:39,943 | 03:27:40,983 | 1,040 |
| **3** | 03:27:42,063 | 03:27:43,054 | 0,991 |
| **4** | 03:27:44,063 | 03:27:45,133 | 1,070 |
| **5** | 03:27:46,173 | 03:27:47,433 | 1,260 |
| **6** | 03:27:48,313 | 03:27:49,483 | 1,170 |
| **7** | 03:27:50,493 | 03:27:51,593 | 1,100 |
| **8** | 03:27:52,793 | 03:27:53,784 | 0,991 |
| **9** | 03:27:55,273 | 03:27:56,543 | 1,270 |
| **10** | 03:27:58,003 | 03:27:59,773 | 1,770 |
| **Média de tempo de execução** | | | **1,188** |

A tabela 6.22 apresenta os resultados obtidos em dez execuções da composição *validatePersonalInformation*, que inclui o *web service IsValidEmail* e a composição de *web services validateInformationSERASA*.

**Tabela 6.22 – Resultado da composição *validatePersonalInformation***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:28:28,013 | 03:28:29,513 | 1,500 |
| **2** | 03:28:32,053 | 03:28:33,413 | 1,360 |
| **3** | 03:28:34,863 | 03:28:36,103 | 1,240 |
| **4** | 03:28:37,053 | 03:28:38,363 | 1,310 |
| **5** | 03:28:39,553 | 03:28:40,833 | 1,280 |
| **6** | 03:28:41,864 | 03:28:43,233 | 1,369 |
| **7** | 03:28:44,233 | 03:28:45,713 | 1,480 |
| **8** | 03:28:46,643 | 03:28:48,929 | 2,286 |
| **9** | 03:28:50,373 | 03:28:51,724 | 1,351 |
| **10** | 03:28:53,705 | 03:28:55,413 | 1,708 |
| **Média de tempo de execução** | | | **1,488** |

A tabela 6.23 apresenta os resultados obtidos em dez execuções da composição *validateCPF*, que inclui o *web service ConsultaSinteseCadastralSERASA*.

**Tabela 6.23 – Resultado da composição *validateCPF***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:34:37,373 | 03:34:38,573 | 1,200 |
| **2** | 03:34:43,693 | 03:34:44,563 | 0,870 |
| **3** | 03:34:45,873 | 03:34:46,733 | 0,860 |
| **4** | 03:34:48,703 | 03:34:49,653 | 0,950 |
| **5** | 03:34:51,203 | 03:34:52,163 | 0,960 |
| **6** | 03:34:53,403 | 03:34:54,483 | 1,080 |
| **7** | 03:34:55,713 | 03:34:56,594 | 0,881 |
| **8** | 03:34:58,073 | 03:34:59,143 | 1,070 |
| **9** | 03:35:00,233 | 03:35:01,143 | 0,910 |
| **10** | 03:35:02,153 | 03:35:03,093 | 0,940 |
| **Média de tempo de execução** | | | **0,972** |

A tabela 6.24 apresenta os resultados obtidos em dez execuções da composição *validateCheck*, que inclui o *web service ConsultaChequeSERASA*.

**Tabela 6.24 – Resultado da composição *validateCheck***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:36:12,023 | 03:36:12,913 | 0,890 |
| **2** | 03:36:17,253 | 03:36:18,263 | 1,010 |
| **3** | 03:36:19,233 | 03:36:20,363 | 1,130 |
| **4** | 03:36:21,403 | 03:36:22,443 | 1,040 |
| **5** | 03:36:23,303 | 03:36:24,393 | 1,090 |
| **6** | 03:36:25,143 | 03:36:26,287 | 1,144 |
| **7** | 03:36:27,353 | 03:36:28,466 | 1,113 |
| **8** | 03:36:29,773 | 03:36:30,753 | 0,980 |
| **9** | 03:36:31,683 | 03:36:32,723 | 1,040 |
| **10** | 03:36:33,733 | 03:36:34,813 | 1,080 |
| **Média de tempo de execução** | | | **1,052** |

A tabela 6.25 apresenta os resultados obtidos em dez execuções da composição *validateLawsuits*, que inclui o *web service ConsultaConcentreSERASA*.

**Tabela 6.25 – Resultado da composição *validateLawsuits***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:36:55,044 | 03:36:56,313 | 1,269 |
| **2** | 03:36:57,254 | 03:36:58,163 | 0,909 |
| **3** | 03:36:59,003 | 03:36:59,983 | 0,980 |
| **4** | 03:37:01,163 | 03:37:02,113 | 0,950 |
| **5** | 03:37:03,163 | 03:37:04,083 | 0,920 |
| **6** | 03:37:05,383 | 03:37:06,793 | 1,410 |
| **7** | 03:37:07,803 | 03:37:08,923 | 1,120 |
| **8** | 03:37:09,573 | 03:37:10,774 | 1,201 |
| **9** | 03:37:11,463 | 03:37:12,393 | 0,930 |
| **10** | 03:37:13,503 | 03:37:14,263 | 0,760 |
| **Média de tempo de execução** | | | **1,045** |

A tabela 6.26 apresenta os resultados obtidos em dez execuções da composição *getInformationBooks*, que inclui as composições de *web services getBook* e *sumValueBooks*.

**Tabela 6.26 – Resultado da composição *getInformationBooks***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:37:57,843 | 03:37:59,713 | 1,870 |
| **2** | 03:38:29,133 | 03:38:34,613 | 5,480 |
| **3** | 03:38:49,423 | 03:38:52,793 | 3,370 |
| **4** | 03:40:14,513 | 03:40:21,963 | 7,450 |
| **5** | 03:40:35,933 | 03:40:40,203 | 4,270 |
| **6** | 03:40:56,434 | 03:41:00,923 | 4,489 |
| **7** | 03:41:13,264 | 03:41:20,353 | 7,089 |
| **8** | 03:41:33,553 | 03:41:38,583 | 5,030 |
| **9** | 03:41:49,233 | 03:41:54,283 | 5,050 |
| **10** | 03:42:06,083 | 03:42:10,563 | 4,480 |
| **Média de tempo de execução** | | | **4,858** |

A tabela 6.27 apresenta os resultados obtidos em dez execuções da composição *validateCredentials*, que inclui as composições de *web services validateCardInformation* e *validatePersonalInformation*.

**Tabela 6.27 – Resultado da composição *validateCredentials***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:44:16,263 | 03:44:22,013 | 5,750 |
| **2** | 03:45:12,143 | 03:45:22,169 | 10,026 |
| **3** | 03:46:22,463 | 03:46:33,703 | 11,240 |
| **4** | 03:46:53,724 | 03:47:06,733 | 13,009 |
| **5** | 03:47:22,433 | 03:47:28,304 | 5,871 |
| **6** | 03:47:37,925 | 03:47:49,994 | 12,069 |
| **7** | 03:48:33,843 | 03:48:47,244 | 13,401 |
| **8** | 03:50:26,993 | 03:50:39,133 | 12,140 |
| **9** | 03:51:28,263 | 03:51:34,153 | 5,890 |
| **10** | 03:51:44,713 | 03:51:58,593 | 13,880 |
| **Média de tempo de execução** | | | **10,328** |

A tabela 6.28 apresenta os resultados obtidos em dez execuções da composição *validateFinancial*, que inclui as composições de *web services validateCPF*, *validateCheck* e *validateLawsuits*.

**Tabela 6.28 – Resultado da composição *validateFinancial***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:53:16,312 | 03:53:19,373 | 3,061 |
| **2** | 03:54:32,393 | 03:54:40,763 | 8,370 |
| **3** | 03:54:56,924 | 03:55:05,123 | 8,199 |
| **4** | 03:55:27,372 | 03:55:35,243 | 7,871 |
| **5** | 03:55:50,073 | 03:55:59,243 | 9,170 |
| **6** | 03:56:06,943 | 03:56:14,823 | 7,880 |
| **7** | 03:56:23,802 | 03:56:32,173 | 8,371 |
| **8** | 03:56:40,023 | 03:56:49,153 | 9,130 |
| **9** | 03:57:19,323 | 03:57:27,833 | 8,510 |
| **10** | 03:57:35,623 | 03:57:45,943 | 10,320 |
| **Média de tempo de execução** | | | **8,088** |

A tabela 6.29 apresenta os resultados obtidos em dez execuções da composição principal *getBooks*, que inclui as composições de *web services getInformationBooks*, *validateCredentials* e *validateFinancial*.

**Tabela 6.29 – Resultado da composição *getBooks***

| # | Horário inicial | Horário final | Tempo de execução (s) |
|---|---|---|---|
| **1** | 03:58:33,233 | 03:58:55,583 | 22,350 |
| **2** | 03:59:04,483 | 03:59:27,963 | 23,480 |
| **3** | 03:59:33,773 | 03:59:59,983 | 26,210 |
| **4** | 04:00:27,763 | 04:01:03,963 | 36,200 |
| **5** | 04:01:14,313 | 04:01:51,623 | 37,310 |
| **6** | 04:02:06,413 | 04:02:30,653 | 24,240 |
| **7** | 04:03:34,643 | 04:04:07,283 | 32,640 |
| **8** | 04:05:06,373 | 04:05:33,824 | 27,451 |
| **9** | 04:05:36,253 | 04:06:01,543 | 25,290 |
| **10** | 04:06:11,083 | 04:06:37,076 | 25,993 |
| **Média de tempo de execução** | | | **28,116** |

## 6.4 Avaliação geral dos resultados

Este trabalho foi validado de duas formas: validação simples e cenários de teste. Na validação simples o foco foi na característica da execução paralela de *web services*. Nessa validação, o contexto da composição de serviços não era importante. Optou-se então por utilizar um *web service* profissional, disponibilizado por um fornecedor conhecido. Nessa avaliação comprovou-se que o motor comportou-se como se esperava: melhorando consideravelmente os tempos de execução ao utilizar a característica de paralelismo de execução.

A segunda validação foi feita sobre dois cenários de teste mais complexos, escritos através da linguagem inSOA. Esses cenários foram compostos por aplicações construídas através da composição de uma variedade de serviços e composições de serviços. Nessa validação, da mesma forma que na anterior, os resultados foram avaliados pelo controle do tempo de execução das composições. Neste caso, os resultados acabaram sendo melhores do que se esperava. É verdade que o tempo de execução das composições demorou mais do que a soma dos tempos dos serviços que faziam parte dela. Mesmo assim, o resultado foi bastante satisfatório e validou mais uma vez o motor SmallSOA.

**Quadro 6.1 – Comparação SmallSOA *versus* trabalhos relacionados**

| Projeto | Ano | Função | Plataforma |
|---|---|---|---|
| ***Mobile SOA*** | 2007 | Cliente de *web services* | JME |
| ***Mobile Host*** | 2006 | Servidor de *web services* | JME |
| ***Peer-to-Peer web services*** | 2005 | Servidor, cliente e repositório de *web services* | JME |
| ***Sliver*** | 2006 | Servidor de composições de *web services* | JME |
| ***JME SOAP Server*** | 2006 | Servidor de *web services* | JME |
| ***Mobile Web Server*** | 2006 | Servidor *web* | httpd / Python/ Symbian |
| ***i-Jetty*** | 2008 | Servidor *web* / *servlet* | Android |
| ***SmallSOA*** | 2009 | Servidor e repositório de composições de *web services* | Android |

Para finalizar, o quadro 6.1 apresenta uma comparação das funcionalidades dos trabalhos relacionados com SmallSOA. *Mobile* SOA é o trabalho mais simples, sendo apenas cliente de *web services*, o que SmallSOA também é, visto que executa os *web services* componentes das composições.*Mobile Host* e JME SOAP *Server* são dois servidores de *web services*, algo que SmallSOA não é, uma vez que ele é um *web service*. *Peer-to-Peer web services* oferece servidor, cliente e repositório de *web services* em ambiente P2P, porém não executa composições de serviço, como é o caso de SmallSOA. Sliver, por sua vez, executa composições de serviços BPEL sobre a plataforma JME, sendo que SmallSOA executa composições inSOA sobre a plataforma Android. Já i-Jetty, embora também execute sobre a plataforma Android, é apenas um servidor *web* e não de composições de *web services* como é SmallSOA.

# 7 CONCLUSÕES

O surgimento dos *web services* como tecnologia facilitadora de interoperabilidade entre sistemas de *software* distribuídos proporcionou o ressurgimento de SOA. Se na abordagem anterior os serviços de SOA eram implementados através de DCOM e CORBA, a abordagem atual utiliza *web services*. Com a crescente atividade de colaboração de pessoas apoiadas por computador, ambientes colaborativos estão sendo construídos utilizando uma vasta variedade de dispositivos como porta de entrada para esse tipo de sistema. Entre esses, incluem-se pequenos dispositivos móveis, atualmente bastante popularizados e atualmente com poder de processamento aceitável. Com isso, visualizam-se cenários onde vários tipos de *hardware* e *software* necessitam comunicar-se. Devido às suas características de baixo acoplamento e independência de plataforma, SOA está se tornando a arquitetura mais indicada para esse cenário heterogêneo que se apresenta.

Este trabalho apresentou SmallSOA, um motor para execução de composições de serviços em ambientes móveis. Inicialmente foi feita uma revisão bibliográfica das principais tecnologias relacionadas com este trabalho; na seqüência foram apresentados trabalhos referentes a infra-estruturas para computação ubíqua baseadas em SOA; após, SmallSOA foi introduzido, juntamente com o projeto U-SOA do qual faz parte; para finalizar, foram feitas validações do motor e apresentados os resultados obitdos.

Ao final deste trabalho, é possível concluir que computação ubíqua baseada em arquiteturas orientadas a serviço tornar-se-ão realidade em alguns anos, não somente no meio acadêmico, mas também em aplicações reais. Essa conclusão baseia-se no próprio sucesso obtido neste trabalho, bem como nos sucessos alcançados por trabalhos similares. No entanto, além de arquiteturas e plataformas, *frameworks* para apoiar esse tipo de aplicação, como é o caso do projeto maior U-SOA, devem continuar a ser construídos.

SmallSOA, por sua vez, conseguiu ser o componente responsável pela execução de composições de serviços em ambientes móveis a que se propôs. Devido a complexidade da arquitetura definida por U-SOA, o que é uma prerrogativa de ambientes móveis e ubíquos, SmallSOA precisou implementar um nível de inteligência apurado em comparação a outras ferramentas similares, nas quais toda a informação é explicitamente definida pelos usuários. Entre outros aspectos, SmallSOA toma decisões em tempo de execução a partir de instruções definidas por uma linguagem de composição previamente escritas.

## 7.1 Contribuições

O término da construção do motor SmallSOA permite elencar as contribuições proporcionadas por ele. As principais delas são destacadas a seguir:

- Componente esperado pelo projeto U-SOA;
- Comunicação através de padrões da indústria baseados em XML;
- Implementação de um repositório de composição de serviços em ambientes móveis;
- Publicação de composições de serviços em ambientes móveis;
- Consulta da descrição completa da composição de serviços, possibilitando a implementação de variadas funções sobre ela, entre as quais análises de impacto e QoS;
- Execução de composições de serviços em ambientes móveis;
- Interpretação do XML gerado a partir da descrição de composição de serviços escrita através da linguagem inSOA;
- Execução do conjunto de padrões de composição previstos em inSOA;
- Execução paralela dos *web services* participantes da composição;
- Tratamento de quedas de conexão que podem ocorrer devido às características intrínsecas da computação móvel.

## 7.2 Limitações

A implementação de SmallSOA visou, nesta primeira abordagem, ser uma prova de conceito do compenente necessário ao projeto maior U-SOA. Dessa forma, o nível de validação exigido neste momento foi o de um teste de conceito, com o funcionamento básico dos seus principais requisitos e características. Devido a isso e ao reduzido tempo disponível para implementação, algumas limitações do trabalho podem ser citadas, sendo que poderão ser solucionadas numa segunda abordagem:

- Testes com telefones celulares baseados na plataforma Android;
- Implementação de comunicação real entre dispositivos móveis;
- Construção de uma arquitetura completa em ambiente real;
- Implementação de uma gama de *web services* a serem disponibilizados em servidores e dispositivos próprios, para utilização nas simulações dos trabalhos envolvidos;

- Simulações a partir de um ambiente controlado para aumentar o nível de correção das medições.

## 7.3 Trabalhos futuros

A partir da definição e implementação do motor para execução de composições de serviços da arquitetura U-SOA, validado com sucesso, é possível identificar trabalhos futuros que podem incrementar as funcionalidades do motor e da própria arquitetura como um todo. Entre eles, destacam-se:

- Em conjunto com a linguagem inSOA, pode-se aumentar a quantidade de padrões de composição previstos na arquitetura;
- Implementação de QoS interno do motor;
- Otimização para diminuição do tempo de execução das composições de serviço;
- Otimização do consumo de recursos do dispositivo móvel;
- Implementação em dispositivo móvel real, o que não é possível neste momento porque esses dispositivos ainda não estão disponíveis no Brasil;
- Implementação de características de computação ubíqua.

# 8 REFERÊNCIAS


[Amazon 2008] **Amazon Web Services**. http://aws.amazon.com/, 2008.

[Android 2008] **Android – An Open Handset Alliance Project**. http://code.google.com/android/, 2008.

[Axis 2008] The Apache Software Foundation. **Apache Axis2**. http://ws.apache.org/axis2/, 2008.

[Baker 2001] Baker, K., Greenberg, S. e Gutwin, C. **Heuristic Evaluation of Groupware Based on the Mechanics of Collaboration**, 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCl'0I) Toronto, Canada, 2001, pp. 123-140.

[Ballinger 2004], Ballinger, K., Ehnebuske, D., Gudgin, M., Nottingham, M. e Yendluri, P. (Eds) **WS-I Basic Profile Version 1.0**, http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html, Abril, 2004.

[Bih 2006] Bih, J. **Service Oriented Architecture (SOA): A New Paradigm to Implement Dynamic E-business Solutions**, Ubiquity Volume 7, Issue 30, Agosto, 2006.

[Blow 2004] Blow, M., Goland Y., Kloppmann M. et al. **BPELJ: BPEL for Java**. http://www.ibm.com/developerworks/library/specification/ws-bpelj/, Março, 2004.

[Bray 2006] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F. e Cowan, J. **Extensible Markup Language (XML) 1.1 (Second Edition).** http://www.w3.org/TR/2006/REC-xml11-20060816/, Setembro, 2006.

[Cerami 2002] Cerami, E. **Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL**. [S.l.]: O'Reilly, 2002.

[Christensen 2001] Christensen, E., Curbera, F., Meredith, G. e Weerawarana, S. **Web Services Description Language (WSDL) 1.1**, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, Março, 2001.

[Clements 2002] Clements, T. **Overview of Soap**. http://developer.java.sun.com/developer/technicalArticles/xml/webservices/, 2002.

[Cremarenco 2003] Cremarenco, C. **Mini xPath Processor for J2ME**. http://sourceforge.net/projects/minixpath, 2003.

[Ellis 1991] Ellis, C. A., Gibbs, S. J. e Rein, G. L. **Groupware: some issues and experiences**, Communications of the ACM, Volume 34, Issue 1, Janeiro, 1991, pp. 39-58.

[Erl 2005] Erl T. **Service-oriented Architecture: Concepts, Technology, and Design**. Prentice Hall, 2005, 760 p.

[Erl 2007] Erl T. **SOA: Principles of Service Design**. Prentice Hall, 2007, 573 p.

[Fuks 2003] Fuks, H., Raposo, A. B. e Gerosa, M. A. **Do Modelo de Colaboração 3C à Engenharia de Groupware**, Simpósio Brasileiro de Sistemas Multimídia e Web – Webmidia 2003, Trilha especial de Trabalho Cooperativo Assistido por Computador, Novembro, 2003, Salvador-BA.

[Gehlen 2005] Gehlen, G. e Pham, L. **Mobile Web Services for Peer-to-Peer Applications**. In Proceedings of the Consumer Communications and Networking Conference 2005, p. 7, Las Vegas, USA, 01/2005, ISBN: 0-7803-8784-8.

[Gerlach 2006] Gerlach, M. e Schrörs, C. **JME SOAP Server**. http://sourceforge.net/projects/jmesoapserver/, 2006.

[Gerosa 2006] Gerosa, M. A., Pimentel, M., Fuks H. e Lucena, C. J. P. de. **Development of Groupware Based on the 3C Collaboration Model and Component Technology**, Y.A. Dimitriadis et al. (Eds.): CRIWG 2006, LNCS 4154, 2006, pp. 302 – 309.

[Graham 2001] Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura Y. e Neyama, R. **Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI**. [S.l.]: Sams Publishing, 2001.

[Hackmann 2006] 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. (published in Proceedings of 4th International Conference on Service Oriented Computing (ICSOC 2006)) .

[Hackmann 2007] Hackmann, G., Gill, C., e Roman, G.-C., **Extending BPEL for Interoperable Pervasive Computing**, Washington University, Department of Computer Science and Engineering, St. Louis, Missouri. (published in Proceedings of IEEE International Conference on Pervasive Services 2007 (ICPS 2007)).

[Haustein 2005] Haustein, S. **kXML 2**. http://www.kxml.org/, 2005.

[Haustein 2006] Haustein, S. e Seigel, J. **kSOAP 2**. http://www.ksoap.org/, 2006.

[He 2003] He, H. **What Is Service-Oriented Architecture**, http://www.xml.com/pub /a/ws/2003/09/30/soa.html, Setembro, 2003.

[Hendricks 2002] Hendricks, M., Galbraith, B., Irani, R. et al. **Professional Java Web Services**. Rio de Janeiro: Altabooks, 2002.

[High Jr. 2005] High Jr, R., Kinder, S. e Graham, S. **IBM's SOA Foundation: An Architectural Introduction and Overview**. http://download.boulder.ibm.com/ibmdl/pub/ software/dw/webservices/ws-soawhitepaper.pdf, Novembro, 2005.

[Hightower 2001] Hightower, J. e Borriello, G. **Location Systems for Ubiquitous Computing**, Computer, vol. 34, no. 8, pp. 57-66, IEEE Computer Society Press, Agosto, 2001.

[Jacobson 1999] Jacobson, I., Booch, G. e Rumbaugh, J. **The Unified Software Development Process**, Reading, MA.: Addison-Wesley, 1999.

[JME 2008] **Java Platform, Micro Edition (Java ME)**. http://java.sun.com/javame/index.jsp, 2008.

[Jorgensen 2002] Jorgensen, D. **Developing .NET Web Services with XML**. Rockland: Syngress, 2002.

[Larman 2004] Larman, C. **Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Processo Unificado**, 2.ed., Porto Alegre: Bookman, 2004.

[Laso-Ballesteros 2006] Laso-Ballesteros, I. **Research perspectives on Collaborative Infrastructures for Collaborative Work Environments**, Proceedings of the 15th IEEE International Workshops on Enabling Technologies:Infrastructure for Collaborative Enterprises (WETICE'06), 2006, pp. 3-10.

[MSDN 2008] **.NET Compact Framework**. http://msdn2.microsoft.com/en-us/library/ms950380.aspx, 2008.

[Newcomer, 2002] Newcomer, E. **Understanding Web Services: XML, WSDL, SOAP and UDDI**. [S.l.]: Addison-Wesley, 2002.

[Nokia 2007] Forum Nokia. **Mobile Web Server: Technology White Paper**, http://sw.nokia.com/id/d34f8746-c121-4fce-aa97-8561804c566b/Mobile_Web_Server_ Technology_White_Paper_v1_0_en.pdf, Maio, 2007.

[O'Reilly 2005] O'Reilly, T. **What Is Web 2.0 – Design Patterns and Business Models for the Next Generation of Software**", www.oreillynet.com/pub/a/oreilly/tim/news/2005 /09/30/what-is-web-20.html, 2005.

[Pham 2005] Pham, L. e Gehlen, G. **Realization and Performance Analysis of a SOAP Server for Mobile Devices**. In Proceedings of the 11th European Wireless Conference 2005, Vol.2, p.p. 791-797, Nicosia, Cyprus, 04/2005, ISBN: 3-8007-2886-9.

[Pijanowski 2007] Pijanowski, K. **Visibility and Control in a Service-Oriented Architecture**, http://msdn2. microsoft.com/en-us/architecture/bb507204.aspx, Maio, 2007.

[Satyanarayanan 1996] Satyanarayanan, M. **Fundamental Challenges in Mobile Computing**, Proc. 15th ACM Symp. Principles of Dist. Comp., Philadelphia, PA, May, 1996.

[Satyanarayanan 2001] Satyanarayanan, M. **Pervasive Computing: Vision and Challenges**, IEEE Personal Communications, Agosto, 2001.

[Snell 2001] Snell, J., Tidwell e D., Kulchenko, P. **Programming Web Services with SOAP**. [S.l.]: O'Reilly, 2001.

[Soliman 2005] Soliman, R., Braun, R. e Simoff, S. **The essential ingredients of collaboration**, Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems, 2005, pp. 366-373.

[SQLite 2008] SQLite. http://www.sqlite.org/, 2008.

[Srirama 2006] Srirama, S., Jarke, M. e Prinz, W. **Mobile Host: A feasibility analysis of mobile Web Service provisioning**, Proceedings of the CAISE'06 Workshop on Ubiquitous Mobile Information and Collaboration Systems UMICS'06, 2006.

[Sun 2008] Sun Microsystems. **The Java Tutorials**. http://java.sun.com/docs/books/tutorial/, 2008.

[Tergujeff 2007] Tergujeff, R., Haajanen, J., Leppänen J. e Toivonen, S. **Mobile SOA: Service Orientation on Lightweight Mobile Devices**, IEEE International Conference on Web Services (ICWS 2007), 2007.

[UDDI 2000] **UDDI Technical White Paper**. http://www.uddi.org/pubs/Iru_UDDI_ Technical_White_Paper.pdf, Setembro, 2000.

[Van der Aalst 2003] Van der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B. e Barros, A. P. **Workflow patterns**, Distributed and Parallel Databases, vol. 14, no. 1, pp. 5–51, 2003.

[Weerawarana 2002] Weerawarana, S. e Curbera, F. **Business Process with BPEL4WS: Understanding BPEL4WS**. http://www.ibm.com/developerworks/ webservices/library/ws-bpelcol1/, Agosto, 2002.

[Weiser 1991] Weiser, M. **The Computer for the Twenty-First Century**, Scientific American, Setembro, 1991.

[Wikman 2006] Wikman J. e Dosa, F. **Providing HTTP Access to Web Servers Running on Mobile Phones**. Nokia Research Center, NRC-TR-2006-005, Maio, 2006.

[Zanuz 2008] Zanuz, L., Filippetto, A., Barcelos, G. e Pinto, S. C. C. S. **SOA Engine – Services Compositions Execution in Ubiquitous Environments**, XIV Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), Vila Velha, 2008. Artigos Resumidos e Workshops. Porto Alegre : Sociedade Brasileira de Computação - SBC, 2008. v. 2. p. 25-28.

[Zanuz 2008b] Zanuz, L., Barcelos, G., Filippetto, A., Pinto, S. C. C. S. **U-SOA - Towards a Ubiquitous Platform Based on Service-Oriented Architecture**, XIV Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), Vila Velha, 2008. Artigos Resumidos e Workshops. Porto Alegre : Sociedade Brasileira de Computação - SBC, 2008. v. 2. p. 105-108.

# APÊNDICE A

**WSDL do *web service* SmallSOA**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="http://localhost/SmallSOA/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SmallSOA"
      targetNamespace="http://localhost/SmallSOA/">
      <wsdl:types>
            <xsd:schema targetNamespace="http://localhost/SmallSOA/">
                  <xsd:element name="Publish" type="tns:PublishType">
                  </xsd:element>
                  <xsd:element name="PublishResponse"
                        type="tns:PublishResponseType">
                  </xsd:element>
                  <xsd:complexType name="PublishType">
                        <xsd:sequence>
                              <xsd:element name="IdComposition"
type="xsd:string"></xsd:element>
                              <xsd:element name="IdUser"
type="xsd:string"></xsd:element>
                              <xsd:element name="Password"
type="xsd:string"></xsd:element>
                              <xsd:element name="Rule"
type="xsd:string"></xsd:element>
                              <xsd:element name="InSOA"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="PublishResponseType">
                        <xsd:sequence>
                              <xsd:element name="Result"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:element name="Get" type="tns:GetType"></xsd:element>
                  <xsd:element name="GetResponse"
                        type="tns:GetResponseType">
                  </xsd:element>
                  <xsd:element name="Execute" type="tns:ExecuteType">
                  </xsd:element>
                  <xsd:element name="ExecuteResponse"
                        type="tns:ExecuteResponseType">
                  </xsd:element>
                  <xsd:complexType name="GetType">
                        <xsd:sequence>
                              <xsd:element name="IdComposition"
                                    type="xsd:string">
                              </xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="GetResponseType">
                        <xsd:sequence>
```

```
                              <xsd:element name="InSOA"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="ExecuteResponseType">
                        <xsd:sequence>
                              <xsd:element name="Result"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="ExecuteType">
                        <xsd:sequence>
                              <xsd:element name="IdComposition"
type="xsd:string"></xsd:element>
                              <xsd:element name="IdUser"
type="xsd:string"></xsd:element>
                              <xsd:element name="Password"
type="xsd:string"></xsd:element>
                              <xsd:element name="Timeout"
type="xsd:int"></xsd:element>
                              <xsd:element name="NumberOfTries"
type="xsd:int"></xsd:element>
                              <xsd:element name="Parameters"
type="tns:ParametersType"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="ParametersType">
                        <xsd:sequence>
                              <xsd:element name="Parameter"
type="tns:ParameterType" maxOccurs="unbounded" minOccurs="0"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="ParameterType">
                        <xsd:sequence>
                              <xsd:element name="Name"
type="xsd:string"></xsd:element>
                              <xsd:element name="Value"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                  </xsd:complexType>
            </xsd:schema>
      </wsdl:types>
      <wsdl:message name="PublishRequest">
            <wsdl:part element="tns:Publish" name="parameters" />
      </wsdl:message>
      <wsdl:message name="PublishResponse">
            <wsdl:part element="tns:PublishResponse" name="parameters" />
      </wsdl:message>
      <wsdl:message name="GetRequest">
            <wsdl:part name="parameters" element="tns:Get"></wsdl:part>
      </wsdl:message>
      <wsdl:message name="GetResponse">
            <wsdl:part name="parameters"
element="tns:GetResponse"></wsdl:part>
      </wsdl:message>
      <wsdl:message name="ExecuteRequest">
            <wsdl:part name="parameters" element="tns:Execute"></wsdl:part>
      </wsdl:message>
      <wsdl:message name="ExecuteResponse">
            <wsdl:part name="parameters"
element="tns:ExecuteResponse"></wsdl:part>
```

```
	</wsdl:message>
	<wsdl:portType name="SmallSOA">
		<wsdl:operation name="Publish">
			<wsdl:input message="tns:PublishRequest" />
			<wsdl:output message="tns:PublishResponse" />
		</wsdl:operation>
		<wsdl:operation name="Get">
			<wsdl:input message="tns:GetRequest"></wsdl:input>
			<wsdl:output message="tns:GetResponse"></wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="Execute">
			<wsdl:input message="tns:ExecuteRequest"></wsdl:input>
			<wsdl:output message="tns:ExecuteResponse"></wsdl:output>
		</wsdl:operation>
	</wsdl:portType>
	<wsdl:binding name="SmallSOASOAP" type="tns:SmallSOA">
		<soap:binding style="document"
			transport="http://schemas.xmlsoap.org/soap/http" />
		<wsdl:operation name="Publish">
			<soap:operation
				soapAction="http://localhost/SmallSOA/Publish" />
			<wsdl:input>
				<soap:body use="literal" />
			</wsdl:input>
			<wsdl:output>
				<soap:body use="literal" />
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="Get">
			<soap:operation
soapAction="http://localhost/SmallSOA/Get" />
			<wsdl:input>
				<soap:body use="literal" />
			</wsdl:input>
			<wsdl:output>
				<soap:body use="literal" />
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="Execute">
			<soap:operation
				soapAction="http://localhost/SmallSOA/Execute" />
			<wsdl:input>
				<soap:body use="literal" />
			</wsdl:input>
			<wsdl:output>
				<soap:body use="literal" />
			</wsdl:output>
		</wsdl:operation>
	</wsdl:binding>
	<wsdl:service name="SmallSOA">
		<wsdl:port binding="tns:SmallSOASOAP" name="SmallSOASOAP">
			<soap:address location="http://localhost/" />
		</wsdl:port>
	</wsdl:service>
</wsdl:definitions>
```

# APÊNDICE B

**_Log_ de execução da composição com um único _web service_**

```
01-15 00:38:30.394: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:32.763: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:38.073: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:39.782: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:41.363: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:43.152: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:44.663: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:46.352: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:47.683: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:49.502: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:50.823: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:52.352: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:53.582: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:55.353: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:56.672: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:38:58.483: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:38:59.703: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:39:01.762: DEBUG/SmallSOA(3038): ------ FIM ------
01-15 00:39:03.012: DEBUG/SmallSOA(3038): ------ INICIO ------
01-15 00:39:07.533: DEBUG/SmallSOA(3038): ------ FIM ------
```

# APÊNDICE C

**_Log_ de execução da composição quatro _web services_ sem paralelismo**

```
01-15 00:42:49.302: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:42:49.359: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:42:51.673: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:42:51.685: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:42:53.392: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:42:53.413: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:42:55.892: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:42:55.902: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:42:57.772: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:42:57.772: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:43:03.263: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:43:03.275: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:43:04.962: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:43:04.992: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:43:07.023: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:43:07.033: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:43:08.762: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:43:08.772: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:43:10.453: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:43:10.453: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:43:13.053: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:43:13.062: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:43:14.873: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:43:14.883: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:43:15.932: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:43:15.942: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:43:15.972: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:43:17.852: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:43:17.862: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:43:19.463: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:43:19.478: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:43:21.433: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:43:21.443: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:43:23.172: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:43:23.213: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:43:24.833: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:43:24.843: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:43:25.962: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:43:25.962: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:43:25.982: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:43:27.623: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:43:27.633: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:43:29.923: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:43:29.933: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:43:31.753: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:43:31.766: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:43:34.263: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:43:34.285: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:43:35.893: DEBUG/SmallSOA(3202): RUN FIM: c
```

```
01-15 00:43:35.902: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:43:37.753: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:43:37.763: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:43:41.362: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:43:41.372: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:43:43.043: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:43:43.053: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:43:44.703: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:43:44.712: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:43:46.443: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:43:46.453: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:43:48.113: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:43:48.123: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:43:52.123: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:43:52.144: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:43:53.743: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:43:53.767: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:43:54.893: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:43:54.893: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:43:54.903: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:43:56.533: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:43:56.543: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:43:58.173: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:43:58.183: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:44:00.403: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:44:00.432: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:44:02.243: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:44:02.259: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:44:04.063: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:44:04.073: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:44:05.772: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:44:05.782: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:44:06.813: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:44:06.813: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:44:06.858: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:44:09.013: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:44:09.023: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:44:10.633: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:44:10.643: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:44:12.463: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:44:12.473: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:44:14.033: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:44:14.043: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:44:18.903: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:44:18.912: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:44:21.383: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:44:21.397: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:44:23.052: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:44:23.064: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:44:25.953: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:44:25.963: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:44:30.683: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:44:30.703: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:44:33.372: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:44:33.383: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:44:33.393: DEBUG/SmallSOA(3202): ------ FIM ------
```

# APÊNDICE D

**_Log_ de execução da composição com quatro _web services_ com paralelismo**

```
01-15 00:45:13.353: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:13.363: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:13.382: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:15.823: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:45:15.873: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:45:15.894: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:45:16.052: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:45:18.512: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:45:18.523: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:45:18.533: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:45:21.813: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:21.833: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:21.842: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:24.573: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:45:24.583: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:45:24.602: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:45:24.623: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:45:26.382: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:45:28.703: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:28.703: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:28.713: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:45:30.603: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:30.613: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:30.643: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:33.383: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:45:33.512: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:45:33.537: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:45:33.562: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:45:34.893: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:34.903: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:35.543: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:45:35.543: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:45:37.393: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:37.423: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:37.473: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:38.753: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:38.753: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:39.363: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:45:39.383: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:45:39.393: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:45:42.183: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:45:42.223: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:45:42.233: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:45:44.293: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:44.303: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:44.333: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:45.603: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
```

```
01-15 00:45:45.603: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:46.293: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:45:46.303: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:45:46.332: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:45:47.573: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:47.593: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:48.292: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:45:48.302: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:45:50.303: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:50.346: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:50.353: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:51.702: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:51.722: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:52.443: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:45:52.473: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:45:52.482: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:45:54.302: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:45:56.573: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:56.588: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:45:56.593: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:45:58.813: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:45:58.826: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:45:58.843: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:45:59.813: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:45:59.813: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:46:02.122: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:46:02.142: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:46:02.173: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:46:03.193: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:46:03.193: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:46:04.793: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:46:04.803: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:46:06.343: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:46:06.353: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:46:06.393: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:46:09.012: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:46:09.023: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:46:09.033: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:46:09.067: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:46:11.812: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:46:11.812: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:46:11.822: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:46:14.363: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:46:14.373: DEBUG/SmallSOA(3202): RUN INI: a
01-15 00:46:14.423: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:46:17.402: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:46:17.432: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:46:17.442: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:46:17.472: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:46:18.903: DEBUG/SmallSOA(3202): InterruptedIOException (Timeout):
http://soap.amazon.com/onca/soap?service=awsecommerceservice
01-15 00:46:18.923: DEBUG/SmallSOA(3202): ERROR: null
01-15 00:46:19.602: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:46:19.612: DEBUG/SmallSOA(3202): ------ FIM ------
01-15 00:46:21.733: DEBUG/SmallSOA(3202): ------ INICIO ------
01-15 00:46:21.753: DEBUG/SmallSOA(3202): RUN INI: a
```

```
01-15 00:46:21.772: DEBUG/SmallSOA(3202): RUN INI: b
01-15 00:46:24.342: DEBUG/SmallSOA(3202): RUN FIM: a
01-15 00:46:24.384: DEBUG/SmallSOA(3202): RUN FIM: b
01-15 00:46:24.393: DEBUG/SmallSOA(3202): RUN INI: c
01-15 00:46:24.423: DEBUG/SmallSOA(3202): RUN INI: d
01-15 00:46:27.212: DEBUG/SmallSOA(3202): RUN FIM: d
01-15 00:46:27.222: DEBUG/SmallSOA(3202): RUN FIM: c
01-15 00:46:27.222: DEBUG/SmallSOA(3202): ------ FIM ------
```