Uma Proposta de Meta-Escalonamento e Execução de Workflows WS-BPEL em Ambientes Multi-Cluster

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Thiago A. Lechuga e aprovada pela Banca Examinadora.

Campinas, 21 de outubro de 2010.

Maria Beatriz Felgar de Toledo

Profª. Drª. Maria Beatriz Felgar de Toledo
(Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Lechuga, Thiago Alvarenga
L495p Uma proposta de meta-escalonamento e execução de workflows WS-BPEL em ambientes multi-cluster/Thiago Alvarenga Lechuga--Campinas, [S.P. : s.n.], 2010.

Orientador : Maria Beatriz Felgar de Toledo.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Serviços na Web. 2.Fluxo de trabalho. 3.WS-BPEL. 4.Engenharia de software. I. Toledo, Maria Beatriz Felgar de . II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: A proposal for meta-scheduling and execution of WS-BPEL workflows in multi-cluster environments

Palavras-chave em inglês (Keywords): 1. We b services. 2. Workflow. 3. WS-BPEL. 4. Software engineering.

Área de concentração: Sistemas de Informação

Titulação: Mestre em Ciência da Computação

Banca examinadora: Profa. Dra. Maria Beatriz Felgar de Toledo (IC – UNICAMP)
Prof. Dr. Mario Antonio Ribeiro Dantas (CT - UFSC)
Profa. Dra. Islene Calciolari Garcia (IC - UNICAMP)

Data da defesa: 21/10/2010

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 21 de outubro de 2010, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Mario Antonio Ribeiro Dantas
Centro Tecnológico / UFSC

Profª. Drª. Islene Calciolari Garcia
IC / UNICAMP

Profª. Drª. Maria Beatriz Felgar de Toledo
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Uma Proposta de Meta-Escalonamento e Execução de Workflows WS-BPEL em Ambientes Multi-Cluster

Thiago A. Lechuga[1]

Outubro de 2010

Banca Examinadora:

- Profª. Drª. Maria Beatriz Felgar de Toledo (Orientadora)
- Prof. Dr. Mario Antonio Ribeiro Dantas
  Centro Tecnológico / UFSC
- Prof. Dr. Edmundo Roberto Mauro Madeira
  IC / UNICAMP
- Profª. Drª. Islene Calciolari Garcia (Suplente)
  IC / UNICAMP
- Prof. Dr. Eleri Cardozo (Suplente)
  FEEC / UNICAMP

[1]Suporte financeiro de: Bolsa CNPq (processo 135020/2008-5) 2008–2009, Bolsa FAPESP (processo 2008/52867-0) 2009–2010.

Knowledge is only useful if you can share it.
Autor desconhecido.

Resumo

A utilização de sistemas distribuídos heterogêneos, tais como os *Multi-Clusters*, também chamados de *Clusters* de *Clusters* (CoC), e as grades computacionais, permite o desenvolvimento de sistemas computacionais mais complexos. No entanto, alguns desses sistemas requerem que os serviços sejam facilmente compostos e executados, mantendo as dependências entre eles; ambos, serviços e dependências, podem ser representados como *workflows*. Ainda, os serviços Web foram adotados na estrutura desses sistemas distribuídos, e WS-BPEL é o padrão para a sua composição. Portanto, o objetivo deste trabalho é permitir a execução de *workflows* escritos em WS-BPEL em *Multi-Clusters*. Para alcançar esse objetivo, uma extensão da linguagem WS-BPEL é apresentada incluindo a especificação de qualidade de serviço (QoS), juntamente com os recursos necessários para execução dos serviços no CoC. Além disso, este trabalho apresenta uma infraestrutura que permite a execução de *workflows* em CoC, e um estudo de caso utilizando a SHARCNET, a fim de avaliar a abordagem proposta. Finalmente, um *workflow* científico real para gerar mosaicos personalizados do céu é implementado usando o ambiente criado.

Abstract

The use of heterogeneous distributed systems, such as Multi-Clusters, as known as Cluster of Clusters (CoC), and computational grids, enables the development of complex computational systems. However some of these systems require services to be easily composed and executed while maintaining the dependencies among them; both the services and the dependencies can be represented as workflows. Moreover, Web Services have been adopted in distributed systems, and WS-BPEL is the standard for Web Services composition. Accordingly, the objective of this work is to enable the execution of WS-BPEL workflows in a CoC. In order to achieve this execution, an extension to the WS-BPEL language is presented that includes the specification of Quality of Services (QoS), along with the required computational resources for service execution in CoCs. Additionally, this paper presents an infrastructure that enables the execution of such workflows in CoCs, and a case study using SHARCNET is discussed in order to evaluate the proposed approach. Finally, a real scientific workflow to generate custom mosaics of the sky is implemented using the criated environment.

Agradecimentos

Apesar de o título que esta dissertação busca ser individual, de forma alguma a trajetória para chegar até aqui seria cumprida dessa forma. Gostaria, portanto, de dedicar essas linhas a agradecer a pelo menos uma parte das pessoas que tornaram mais esse objetivo possível.

Primeiramente à **Deus**, à **Vida** e ao meus pais, **Willian** e **Lourdes**, que me educaram e apoiaram, sendo assim os responsáveis por tudo o que sou hoje. Amo vocês.

À toda a minha família, à minha vó, **Maria**, e aos meus tios e tias, **Esther**, **Elísio**, **Imaculada**, **Carlos**, **Geraldo**, **Lúcia**, **Yvonne** e **Magno**. De ajudando na mudança a diminuindo minha pilha de roupas sujas, cada um emprestou um pouco de si para me apoiar nessa jornada. Eu poderia passar horas escrevendo as inúmeras maneiras que vocês me ajudaram, mas nada disso se compara a companhia, carinho, amor e afeto que recebi como se eu ainda fosse o pequeno sobrinho/neto que se mudou para longe a mais de 10 anos. Amo todos vocês. Não poderia esquecer de mandar um abraço especial para meus irmãos, primos e primas e para meu avô, **Edgard**, que, esteja onde estiver, sei que cuida muito bem de todos nós e está muito contente com essa conquista.

À **Camila** e meus grandes amigos, **Diogo**, **Henrique** e **Leonardo**, que hoje já me apoiaram em uma grande parcela da minha vida. Perto ou longe, algumas pessoas são para a vida toda.

À minha orientadora, Prof. **Maria Beatriz**. Obrigado pelo tempo, orientação e paciência.

A todas as pessoas que me receberam e me ajudaram no Canadá. À Prof. **Miriam**, ao **Américo**, ao **Mário** e ao **Lúcio**, obrigado pela orientação, ajuda, apoio e companhia de todos. Aos grandes novos amigos que tornaram minha estadia nesse país gelado muito mais divertida e prazerosa, **Elisa**, **Fabrício**, **Lorena** e **Márcia**. Obrigado a todos, esse período estará para sempre gravado em minha memória com muita saudade.

Aos amigos da UNICAMP e do mestrado, entre eles, **Cristiano**, **Daniel**, **Douglas**, **Faveri**, **Guilherme**, **Isaura**, **Marcos**, **Maria**, **Robinho**, **Tripodi**, **Willian**. Obrigado pela companhia, pelos dias divertidos, pelas risadas e por toda a *fundamental* ajuda. Sem vocês eu ainda estaria procurando tempo para estudar grafos. Podem ter certeza que a

minha gratidão é muito maior do que eu conseguiria expressar mesmo com a “Perícia de cirurgião atômico da Noruega”. ;) Vou sentir muita falta de vocês.

A todo o pessoal do laboratório e ao chefe **Tizzei**, que aguentaram pacientemente minhas bagunças e ainda estavam sempre dispostos a me ajudar com valiosas dicas.

Por ultimo, e não menos importante, todas as outras pessoas que colaboram direta ou indiretamente e que, por falta de memória e/ou espaço, não estão nomeadas aqui. Um sincero OBRIGADO.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Códigos

Capítulo 1

Introdução

Os *clusters* de alta performance (HPCCs) [1] continuam sendo uma solução para a demanda computacional constantemente crescente de diversos institutos comerciais e de pesquisa. Entretanto, enquanto um *cluster* é projetado para atender os picos de carga de uma organização, essa situação não ocorre com frequência, deixando o equipamento a maior parte do tempo ocioso. Portando, para otimizar a utilização dos seus recursos, diversas organizações tentem a compartilhar HPCCs. Assim, enquanto uma organização não utiliza seus recursos, outros parceiros podem utilizá-los, aumentando a capacidade máxima de processamento de ambas. Então, ao invés de pequenos grupos de usuários tendo acesso exclusivo a um sistema específico, temos grandes grupos compartilhando diversos *clusters*. Essa abordagem colaborativa pode maximizar a utilização dos recursos, permitir a implementação de tarefas mais complexas e reduzir o tempo de espera [42].

O uso de sistemas heterogêneos distribuídos, como os *Multi-Clusters*, ou *Clusters* de *Clusters*(CoC) [2], e as grades computacionais, permitem o estudo de diversos problemas que eram computacionalmente inviáveis em outros ambientes. Entretanto, alguns desses problemas são complexos e precisam ser facilmente compostos e executados, mantendo as dependências entre si. Embora serviços e dependências possam ser representadas através de *wokflows*, os padrões atuais para a composição de serviços em sistemas distribuídos heterogêneos ainda são inadequadas. Como os serviços Web [2] (WS)[3] foram adotados na infraestrutura desses sistemas e como o WS-BPEL [38], *Web Services Business Process Execution Language*, é o padrão para a sua composição, essa linguagem se torna uma forte candidata para a utilização nesse novo cenário. Contundo, ainda existem requisitos específicos que devem ser considerados, como por exemplo a seleção de recursos baseada em requisitos não-funcionais [19].

[1] Do inglês, *High Performance Computing Clusters*
[2] Do inglês, *Cluster of Clusters*
[3] Do inglês, *Web Services*

1.1 Objetivos

O objetivo deste trabalho é possibilitar a especificação e execução de *wokflows* especificados na linguagem WS-BPEL que utilizam recursos de *Multi-Clusters*. Especificamente, as contribuições esperadas incluem:

- Uma extensão à linguagem WS-BPEL que permita especificação e seleção de recursos baseada em requisitos de qualidade de serviço (QoS)[4] e de recursos computacionais requeridos para a execução de cada serviço;
- Uma sugestão de arquitetura para permitir a execução de *workflows* em CoCs.
- Implementação da arquitetura proposta em um ambiente de produção.
- Especificação e implementação de um problema real no ambiente criado.

1.2 Estrutura da Dissertação

Esta dissertação contém mais cinco capítulos estruturados como descrito a baixo:

- Capítulo 2: Apresenta os conceitos básicos necessários para o entendimento deste trabalho. Dividido em três sessões, fala sobre os conceitos de *clusters*, computação orientada a serviços e orquestração de serviços;
- Capítulo 3: Relaciona e compara alguns trabalhos relacionados. Divide-se em duas sessões. Na Sessão 3.1 são levantados os trabalhos com o foco na execução de *workflows* em sistemas distribuídos heterogêneos. Na Sessão 3.2 algumas pesquisas sobre escalonamento global, ou Meta-Escalonamento, em *Multi-Clusters* são apresentadas;
- Capítulo 4: Descreve com detalhes as principais contribuições desenvolvidas neste trabalho. A Sessão 4.1 explica as adaptações feitas na linguagem WS-BPEL para permitir que requisitos de QoS e recursos necessários para a execução das tarefas sejam expressos na composição, condição necessária para a execução em CoCs. Na Sessão 4.2 é proposta uma arquitetura para a execução de *workflows* descritos em WS-BPEL que utilizam recursos de um CoC, permitindo seleção de recursos baseada em requisitos funcionais e não-funcionais. Finalmente, a Sessão 4.3 demonstra uma aplicação dessa arquitetura utilizando a rede SHARCNET como *Multi-Cluster*, e ilustra detalhes da implementação, como o `Escalonador Global` (Sessão 4.3.2), o `Broker` (Sessão 4.3.2), o `Mediador` (Sessão 4.3.2), o `Monitor` (Sessão 4.3.2), e outros elementos da arquitetura;

[4] Do inglês, *Quality of Services*

- Capítulo 5: Demonstra uma aplicação de um problema real implementada utilizando as contribuições apresentadas neste trabalho;
- Capítulo 6: Conclui a dissertação destacando as contribuições e listando trabalhos futuros.

Capítulo 2

Fundamentos

Neste capítulo, de caráter conceitual, serão apresentados os fundamentos necessários para o entendimento desta dissertação. Inicialmente, serão abordados os conceitos de *Cluster* e *Multi-Clusters*. Em seguida, os fundamentos de computação orientada a serviços são abordados, falando sobre os mecanismos de descoberta, descrição e comunicação dos serviços Web. Para concluir, o texto aborda os conceitos de *workflows* e WS-BPEL, e os relaciona com os *Multi-Clusters*.

2.1 Clusters

Na tentativa de resolver problemas com a complexidade cada vez maior utilizando computadores, a tendência natural foi criar processadores cada vez mais rápidos. Gordon Moore previu em 1965 [34] um crescimento quase que exponencial para os circuitos integrados, o que pode também ser refletido, de certa forma, na capacidade de processamento dos mesmos. Entretanto, esse crescimento está atingindo os limites da própria física e decaindo cada vez mais. Assim, na tentativa de transpor essas barreiras, as indústrias de microprocessadores estão investindo em novas tecnologias. Uma possível solução é o uso dos *clusters* computacionais, também conhecidos na literatura em português por agregados computacionais [15].

O processamento de alto desempenho (PAD), conhecido em inglês por *High Performance Computing* (HPC), foi baseado durante muito tempo em computadores paralelos específicos, tais como aqueles com arquiteturas massivamente paralelas (MPP)[1] e os de memória compartilhada (SMP)[2]. Contudo, devido ao alto custo de máquinas dos tipos MPP e SMP, além da grande oferta de computadores pessoais existentes nas redes das instituições, políticas de agregar máquinas estão sendo cada vez mais utilizadas [15].

[1]Do inglês, *Massively Parallel Processor*
[2]Do inglês, *Shared Memory Processors*

2.1.1 Classificação

Atualmente não existe uma taxonomia globalmente aceita para os ambientes de *clusters*. Dantas [15] classifica os agregados usando cinco tipos de métricas:

- **Limite Geográfico:** Pequena, média e grande;
- **Utilização dos Nós:** Dedicados e não dedicados;
- **Tipo de Hardware:** NOW, COW e CLUMPS;
- **Aplicações Alvo:** Alto desempenho e alta disponibilidade;
- **Tipos de Nós:** Homogêneo, Heterogêneo.

A seguir cada uma das métricas anteriormente listada é detalhada e exemplificada.

Limite Geográfico

As configurações podem variar desde a simples agregação de computadores do tipo PC utilizando *hubs* ou *switches*, até o uso de dispositivos especiais de interligação, como *Myrinet*, *Quadrics* e *Memory Channel*. Na maioria das organizações os *clusters* tendem a surgir em salas específicas e laboratórios, ou seja, locais com certa limitação geográfica. A partir do conhecimento adquirido por uma determinada equipe, o *cluster* tende a crescer para uma configuração departamental. Finalmente, obtendo sucesso, o sistema pode se expandir para um nível organizacional com o intuito de propagar os benefícios do uso do paradigma.

Utilização dos Nós

Este aspecto pode ser considerado um dos mais importantes a ser observado na formação de um *cluster*. É fundamental que seja estabelecido a participação dedicada ou não-dedicada dos nós que irão compor o agregado. Fatores como políticas de gerenciamento, segurança, disponibilidade, escalonamento e balanceamento de carga estão relacionados diretamente à forma de utilização dos nós.

Tipo de Hardware

Os tipos de *hardware* empregados vem sendo classificados de uma forma empírica como *Nows*, *Cows* e *Clumps*.

As *Nows* (*Network of Workstations*) são caracterizadas pelo uso de estações de trabalho distribuídas em uma rede local. Configurações classificadas como *Cows* (*Cluster*

of Workstations) se diferenciam das *Nows* pelo uso de máquinas homogêneas e dedicadas à execução de aplicações específicas. Os *Clumps* (*Cluster of SMPs*) são ambientes compostos de máquinas com arquitetura SMP, anteriormente citada.

Aplicações Alvo

Com relação à necessidade das aplicações os *clusters* geralmente se dividem em dois alvos principais: alto desempenho e alta disponibilidade. Embora também seja possível observar aplicações com os dois tipos de requisitos.

No primeiro grupo as aplicações necessitam de alto desempenho para sua execução. Assim, um *cluster* com um grande número de processadores, memória e disco, é o ambiente ideal para esse tipo de aplicação.

Já para o outro grupo, no qual as aplicações precisam de alta disponibilidade, o agregado se mostra eficiente permitindo que a aplicação seja distribuída garantindo que esta não seja interrompida.

Tipos de Nós

Outro fator importante a ser observado na classificação dos *clusters* se refere à similaridade de *hardware* e *software* dos nós que compõem o ambiente. Podendo ser agrupados em nós homogêneos e heterogêneos.

2.1.2 Multi-Clusters

Os *High Performance Computing Clusters (HPCC)* continuam sendo uma solução comum para satisfazer a sempre crescente demanda computacional de diversos institutos comerciais e de pesquisa. Entretanto, um único *cluster* é normalmente planejado para suportar cargas máximas de trabalho de uma organização, que geralmente não acontecem com frequência. Para otimizar o uso dos recursos, diversas organizações tendem a compartilhar HPCCs entre si. Dessa forma, ao invés de pequenos grupos de usuário com acesso exclusivo para um *hardware* específico, temos grupos maiores que compartilham diversos *clusters*. Essa abordagem pode potencializar o uso dos recursos, permitindo a execução de tarefas maiores e diminuindo o tempo de espera [42].

Entretanto, para atingir tal nível de compartilhamento alguns desafios devem ser superados; entre eles, está o gerenciamento dos recursos desse tipo de arquitetura, comumente chamada de *Cluster* de *Clusters* (CoC)[3], ou Multi-Clusters. Por esse motivo o escalonamento de tarefas em plataformas heterogêneas tornou-se recentemente alvo de pesquisas [22].

[3]Na literatura inlgesa, *Cluster of Clusters*

Como exemplo desse tipo de estrutura pode-se citar a SHARCNET [43], uma rede multi-institucional de *clusters* de alto desempenho, espalhada por 16 instituições acadêmicas na província de Ontário, Canadá. O sistema citado não possui um escalonador global, por esse motivo a seleção de onde cada tarefa deve ser executada é feita de forma manual, obrigando o usuário a verificar quais sistemas estão ativos, e impedindo que suas tarefas sejam executadas de forma automatizada, como em um *workflow*.

2.2 Computação Orientada a Serviços

Computação Orientada a Serviços é um paradigma de modelagem que especifica a criação da infraestrutura de *software* de um determinado negócio em forma de serviços. Serviços podem ser definidos como componentes de software auto-descritivos que apoiam a composição ágil de aplicações [30].

Esse modelo pode ser aplicado como estratégia para a implantação de uma arquitetura orientada a serviços, denominada SOA[4] [1]. Como outros paradigmas de modelagem, a orientação a serviços fornece meios para desacoplar interesses e fomentar o reuso.

2.2.1 SOA e Serviços Web

Um sistema distribuído é composto por um conjunto de elementos de *hardware* e *software* que se comunicam através da troca de mensagens [11]. A comunicação entre máquina e homem é algo já bastante difundido com a proliferação da Web e das tecnologias correlatas como, por exemplo, a linguagem HTML. No entanto a possibilidade de efetuar interações entre máquina-máquina, ou melhor, aplicação-aplicação vem ganhando um espaço considerável na área da computação. Algumas tecnologias já implementam mecanismos que possibilitam a execução de procedimentos remotos como, por exemplo, Sun RPC [11] e Java RMI [21]. Essas tecnologias possibilitam que uma aplicação invoque uma outra remotamente passando parâmetros de processamento e recebendo uma resposta como resultado de tal procedimento.

Durante muitos anos, aplicações que necessitavam de algum tipo de comunicação externa interagiram através de soluções *ad hoc*, utilizando a infraestrutura básica da internet [12]. No entanto tais tecnologias são, em muitos casos, dependentes de plataformas e linguagens de programação, fato que as torna mais fechadas.

As arquiteturas orientadas a serviços fazem uso extensivo da tecnologia de serviços Web de maneira a definir uma estrutura distribuída, independente de plataforma e linguagem de programação. Assim, os serviços Web emergiram como um *framework* sistemático e extensível para possibilitar a interação entre aplicações. Um serviço Web, portanto, é um

[4]Do inglês, *Service-Oriented Architecture*

sistema de software projetado, utilizando um conjunto de padrões abertos, para permitir a interoperabilidade entre sistemas [2]. Dessa forma, uma determinada aplicação pode ser implementada na forma de um serviço e disponibilizada para ser invocada remotamente.

Na tentativa de ilustrar a utilização dos serviços Web em uma situação real, podemos imaginar um site de vendas pela Internet, que necessita validar o crédito do comprador antes de proceder com a venda. O sistema então acessa um serviço (WS) que cuida de todos os passos necessários à verificação de crédito: verifica o histórico das compras efetuadas pelo consumidor na empresa, a situação de crédito do consumidor no sistema público, etc. O serviço Web obtém estes dados e retorna a situação de crédito deste consumidor para o site.

A estrutura SOA possui três camadas básicas: descoberta de serviços, descrição de serviços e protocolo de comunicação. A Figura 2.1 mostra os protocolos mais comuns, e aceitos como padrão, para cada uma das três camadas; UDDI, WSDL e SOAP, respectivamente. Cada uma delas será detalhada posteriormente.

A Figura 2.2 ilustra, de uma maneira abstrata, o modelo empregado através da estrutura de camadas de serviços Web.

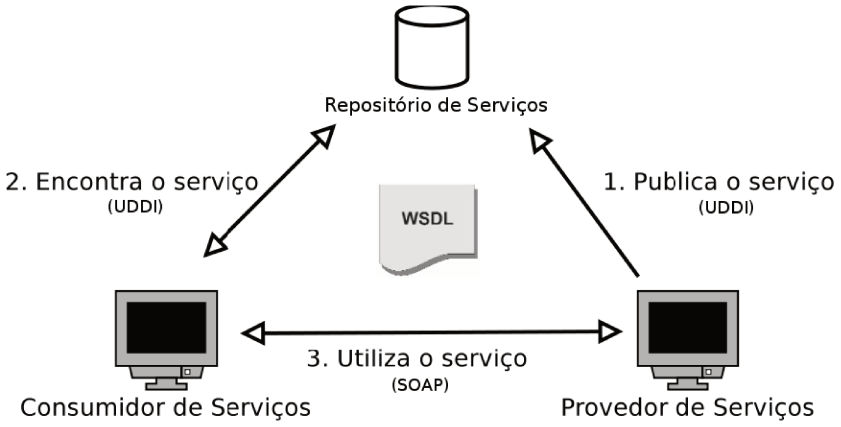

Figura 2.1: Padrões SOA estabelecidos.

Descoberta: UDDI

O padrão *Universal Description, Discovery and Integration* (UDDI) [39] oferece aos usuários de serviços Web, através de um registro centralizado equivalente a uma "lista telefônica" online, um mecanismo para localizar provedores, os serviços disponibilizados por eles e as informações necessárias para acessá-los [12].

Baseado em XML, provê uma infraestrutura para integrar informação em ambientes de serviços Web tanto para serviços disponíveis publicamente quanto para serviços expostos

Figura 2.2: Estrutura básica de um serviço Web.

internamente em organizações. O UDDI provê duas especificações básicas que definem a estrutura e operações do serviço de registro [12]:

- uma definição das informações a serem fornecidas sobre cada serviço, e como codificá-las;
- uma API de consulta e atualização do registro que descreve como a informação pode ser acessada ou atualizada.

Tanto a consulta quanto a atualização são realizadas através da troca de mensagens SOAP.

As informações no repositório estão categorizadas de acordo com o objetivo de uso de cada tipo de informação, fazendo uma analogia com um diretório de telefones [2]:

- **Páginas Brancas:** Listagens de organizações, informações de contato e serviços que elas provêem. Usando o registro como um catálogo de páginas brancas, os usuários do UDDI podem encontrar serviços Web providos por uma dada empresa;
- **Páginas Amarelas:** Classificações de organizações e WSs de acordo com taxonomias padrão ou definidas pelo usuário. Através das páginas amarelas, é possível buscar serviços pela categoria a qual eles pertencem, de acordo com um dado esquema de classificação.

- **Páginas Verdes:** Descrevem como um dado Serviço Web pode ser invocado; Essa informação é obtida através de apontadores para a documentação do serviço, geralmente armazenada fora do registro (ex: Página da Web do provedor do serviço).

Inicialmente, o padrão UDDI foi projetado para apoiar um modelo conhecido como *Universal Business Registry* (UBR) [12], representando um serviço público e central. Entretanto, a maioria dos serviços Web da atualidade são ou internos a uma única instituição, ou compartilhado por alguns parceiros de confiança. Dessa forma, o padrão UDDI evoluiu na sua versão 3.0 para também apoiar registros privados. Os seguintes tipos de repositório são permitidos [2]:

- **Repositório público:** repositório que provê acesso público e aberto aos dados de registro. Um repositório disponível como um site na Web publicado e conhecido é um exemplo de repositório público;
- **Repositório privado:** repositório interno, isolado por um *firewall* de uma rede privada. Organizações podem utilizar repositórios privados para apoiar a integração de suas próprias aplicações internas;
- **Repositório compartilhado:** repositório instalado em um ambiente controlado, sendo compartilhado com um grupo limitado de organizações. Por exemplo, uma rede de parceiros de negócio pode utilizar um repositório compartilhado para apoiar a integração de aplicações entre eles.

Descrição: WSDL

A *Web Services Description Language* (WSDL) [10] foi criada em conjunto pela IBM, Microsoft e Ariba através da união de especificações propostas individualmente por essas empresas. Seu objetivo principal é fornecer aos usuários a interface dos serviços Web, provendo um ponto de acesso para consumidores de serviços, informando quais mensagens devem ser trocadas para executá-los. Trata-se de um documento escrito em XML[5] que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis [12].

Uma descrição WSDL completa inclui dois tipos de informação. Uma descrição de serviços no nível de aplicação, chamada de interface abstrata, e a descrição detalhada e dependente de protocolo, chamada de interface concreta [12]. A Figura 2.3 mostra a estrutura de um documento WSDL contendo as interfaces concreta e abstrata.

A parte abstrata de uma descrição de serviço WSDL é definida em termos de mensagens trocadas em uma interação com o serviço, compreendendo os seguintes elementos [15]:

[5] *Extensible Markup Language*

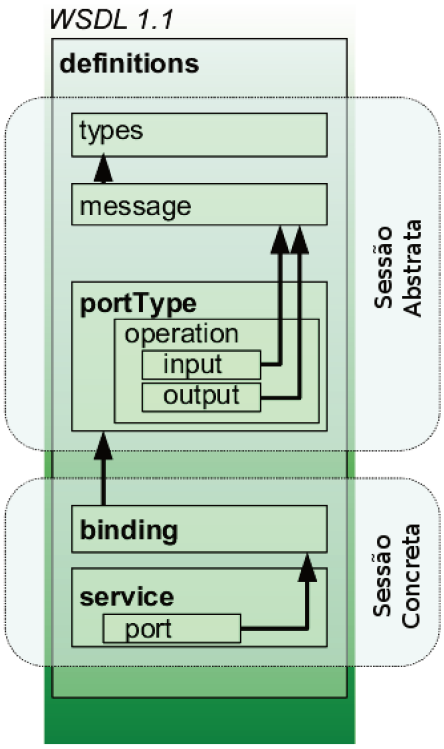

Figura 2.3: Estrutura WSDL 1.1.

- **portType:** descreve uma coleção de operações oferecidas;
- **operation:** descrição abstrata de uma ação que um determinado serviço pode oferecer. Define uma coleção de mensagens informando qual o papel de cada uma;
- **types:** informa os tipos de dados empregados pelo serviço. Geralmente utiliza-se o esquema de definição XML, também conhecido como XSD [49].

Os elementos WSDL que compõem a parte concreta de uma descrição de serviço são apresentados a seguir [2]:

- **binding:** especifica o estilo de interação e o protocolo de comunicação para um elemento `portType`;
- **port:** combina informações de ligação com um ponto de acesso, especificado por um URI através do qual uma implementação de um `portType` determinado pode ser acessada;
- **service:** é uma coleção de elementos `port`.

O Código 2.1 ilustra a utilização desses elementos em uma descrição de serviço Web. No exemplo de documento WSDL, um elemento `portType` (`Implementação`) define uma operação chamada `executar` (linhas 7-11). Essa operação inclui uma mensagem de entrada nomeada `executarMsg` (linha 9). Os componentes da mensagem são definidos utilizando o elemento `message` (linhas 3-5). Na parte concreta da descrição de serviço, um elemento `binding` é especificado para o `portType` definido na parte abstrata (linhas 14-21). O elemento binding `ImplementaçãoPortBinding` determina, por exemplo, a utilização de HTTP como protocolo de comunicações (linha 15). Finalmente, um serviço (`ImplementaçãoService`) é criado, definindo um elemento `port`, chamado `ImplementaçãoPort`, que combina o elemento *binding* com o ponto de acesso de uma implementação do elemento `portType` definido anteriormente (linhas 23-27).

```
1  <definitions name="ImplementacaoService" targetNamespace="http://ewe.org/">
2
3      <message name="executarMsg">
4          <part name="parameters" element="executarParam" />
5      </message>
6
7      <portType name="Implementacao">
8          <operation name="executar">
9              <input wsaw:Action=".../executar" message="executarMsg" />
10         </operation>
11     </portType>
12
13     <!-- Binding concreto usando SOAP-->
14     <binding name="ImplementacaoPortBinding" type="Implementacao">
15         <soap:binding transport=".../soap/http" style="document" />
16         <operation name="executar">
17             <input>
18                 <soap:body parts="parameters"/>
19             </input>
20         </operation>
21     </binding>
22
23     <service name="ImplementacaoService">
24         <port name="ImplementacaoPort" binding="ImplementacaoPortBinding">
25             <soap:address location=".../Implementacao" />
26         </port>
27     </service>
28
29 </definitions>
```

Código 2.1: Exemplo WSDL 1.1 contendo apenas as estruturas principais. Detalhes foram omitidos.

Comunicação: SOAP

O *Simple Object Access Protocol* (SOAP) é um protocolo baseado em XML que possibilita a comunicação entre serviços Web. Ele define como deve ser o formato de uma mensagem trocada entre duas aplicações e utiliza protocolos já existentes para o envio das mensagens como, por exemplo, HTTP, SMTP [12].

A estrutura básica de uma mensagem SOAP é bem simples: um elemento XML principal, chamado de envelope, contendo uma área de cabeçalho e uma área para o corpo da mensagem. Essa estrutura pode ser vista na Figura 2.4.

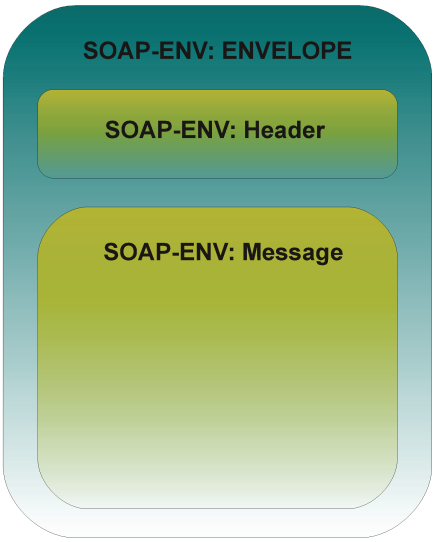

Figura 2.4: Estrutura de uma mensagem SOAP.

O elemento `Header` é opcional e pode ser utilizado para funções específicas da aplicação no transporte da mensagem. Os intermediários através dos quais a mensagem passa entre a origem e o destino final podem processar esses cabeçalhos de maneira a implementar diversas funcionalidades como, por exemplo, autenticação, controle de transação, entre outros [31].

Já o elemento `Body` contém a mensagem propriamente dita que está sendo trocada entre as aplicações. Esse corpo da mensagem pode implementar dois tipos de interação entre as aplicações [2]:

- **Document-Style:** Nesse tipo de interação o elemento `Body` transporta uma mensagem XML que diz respeito a um documento que está sendo trocado entre as aplicações como, por exemplo, um pedido de compra contendo todos os itens, quan-

tidade e preço. Dessa forma, o serviço destino recebe essa mensagem e processa esse documento;

- **RPC-Style:** Nesse tipo de interação o elemento `Body` transporta um documento que carrega informações sobre uma chamada de procedimento remoto, contendo qual procedimento deve ser invocado e o valor de cada um dos seus parâmetros.

O Código 2.2 retrata um exemplo de mensagem SOAP. Primeiramente o `Envelope` é criado (linhas 1-17). Em seguida, o *Header* é definido (linhas 3-8) com os parâmetros *MessageID* (linha 4) e *ReplyTo* (linhas 5-7). Finalmente o corpo da mensagem é criado (linhas 10-15) para invocar a operação `executar` (linhas 11-14), e informando os parâmetros `comando` (linha 12) e `arqOutput` (linha 13). Essa mensagem SOAP foi criada para invocar o serviço definido no WSDL demonstrado anteriormente no Código 2.1.

```
1  <soapenv:Envelope>
2
3     <soapenv:Header>
4        <add:MessageID>ID-Unico</add:MessageID>
5        <add:ReplyTo>
6           <add:Address>http://127.0.0.1/</add:Address>
7        </add:ReplyTo>
8     </soapenv:Header>
9
10    <soapenv:Body>
11       <ewe:executar>
12          <comando>parametro1</comando>
13          <arqOutput>parametro2</arqOutput>
14       </ewe:execSharcnet>
15    </soapenv:Body>
16
17 </soapenv:Envelope>
```

Código 2.2: Exemplo de uma mensagem SOAP contendo apenas as estruturas principais. Detalhes foram omitidos.

2.2.2 Outros Padrões Relacionados

Existem diversos outros padrões associados à arquitetura orientada a serviços e os WSs. Nessa subseção serão apresentados os mais relevantes a este trabalho.

WS-Addressing

WS-Addressing [24] é uma especificação para permitir que serviços Web comuniquem informações de endereçamento. É uma maneira padronizada de incluir informações de

roteamento da mensagem no cabeçalho de mensagens SOAP [2].

Essa especificação é comumente utilizada para a comunicação de serviços Web em modo assíncrono. Através da configuração da *tag* `ReplyTo` no cabeçalho WS-Addressing, o provedor do serviço pode responder à mensagem através de uma nova conexão. Dessa forma, o tempo de vida da chamada/resposta SOAP fica desacoplado do tempo de vida do protocolo HTTP, permitindo a execução de processos muito demorados [2].

Os principais elementos de um cabeçalho WS-Addressing são:

- **MessageID:** Código usado para identificar unicamente a mensagem;
- **ReplyTo:** Endereço para qual o serviço deve enviar a resposta;
- **RelatesTo:** Utilizado nas mensagens de resposta, contém o mesmo valor do *MessageID* da mensagem original. É através desse valor que o cliente pode associar as requisições às respostas.

Ainda no Código 2.2 é possível verificar a utilização do WS-Addressing no cabeçalho (linhas 3-8). A mensagem de resposta deverá ser destinada ao endereço contido na *tag* `ReplyTo`, e deverá conter a *tag* `RelatesTo` com o mesmo valor do *MessageID* de entrada, de forma similar ao Código 2.3.

```
1 <soapenv:Header>
2     <add:RelatesTo>ID-Unico</add:RelatesTo>
3 </soapenv:Header>
```

Código 2.3: Trecho do cabeçalho de uma requisição SOAP de resposta utilizando WS-Addressing.

WS-Policy

Embora WSDL seja o padrão para a descrição de funcionalidade de serviços Web, esforços de pesquisas recentes estão focados em linguagens que possam aprimorar tal descrição. Esse aprimoramento inclui a consideração de aspectos que não são diretamente relacionados com o que um serviço faz ou como invocar um serviço Web. O *Web Services Policy Framework* (WS-Policy) [51] representa um desses esforços e, devido à sua flexibilidade e capacidade de extensão, é um candidato para tornar-se um padrão para a especificação de políticas para serviços Web [5].

O WS-Policy é apoiado por um consórcio formado por diversas empresas, como, por exemplo, BEA, IBM, Microsoft, SAP e VeriSign. Esse consórcio foi responsável pela submissão de especificações nas áreas de transações, confiabilidade de mensagens, dentre outras, para entidades de padronização de serviços Web.

O padrão compreende um modelo para definir propriedades de serviços Web e uma sintaxe correspondente para expressar essas definições como políticas. Referências para políticas podem ser incluídas em documentos XML [50]. Políticas podem ser aplicadas em diferentes níveis, tais como, mensagem, operação e serviço.

Em WS-Policy, uma política é uma coleção de alternativas. Cada alternativa de política é uma coleção de asserções. Tipicamente, asserções de política especificam características que são importantes para a seleção e utilização apropriadas de serviços Web, por exemplo, características não-funcionais [51].

O padrão WS-Policy define um formato normalizado para especificar políticas com o objetivo de facilitar sua manipulação. A forma normal de política WS-Policy é uma disjunção das alternativas de uma política e uma conjunção das asserções em cada uma das alternativas da política.

Alguns exemplos de atributos de QoS são descritos a seguir:

- **Tempo de resposta:** O intervalo de tempo demandado para que um serviço complete sua tarefa;
- **Capacidade:** O número de requisições concorrentes permitido por um serviço;
- **Disponibilidade:** A porcentagem de tempo em que um serviço está operante;
- **Segurança:** Define se um serviço oferece mecanismos de confidencialidade, integridade, autenticação e autorização.

O Código 2.4 apresenta um exemplo de um WS-Policy. A política define que tempo de resposta (`ResponseTime`) para a operação `get` deve ser menor que `45` (linha 8). A *tag* `wsp:All` indica que todas as políticas devem ser atendidas (linha 3).

```
1  <wsp:Policy>
2      <wsp:ExactlyOne>
3          <wsp:All>
4              <qosp:ResponseTime
5                  xmlns:qosp="...schema/qospolicy"
6                  operation="get"
7                  specification="uddi:qos:attribute:
8                  responsetime">45</qosp:ResponseTime>
9          </wsp:All>
10     </wsp:ExactlyOne>
11 </wsp:Policy>
```

Código 2.4: Exemplo de uma política WS-Policy contendo apenas as estruturas principais. Detalhes foram omitidos.

2.3 Orquestração de serviços

A computação distribuída tem rumado para uma arquitetura orientada a serviços em que funções computacionais são representadas por serviços Web. Essa abordagem abre a possibilidade de se criar novos serviços combinando outros serviços existentes. Dessa forma, a reusabilidade de recursos existentes entra em cena e novas funcionalidades podem ser implementadas de maneira mais rápida.

A orquestração é criada através da definição de um "processo de negócio" que capta a lógica de como os serviços individuais devem ser combinados. Essa abordagem atrai um grande interesse tanto da indústria quando da academia [45].

Serviços Web podem ser combinados de diferentes maneiras para formar uma composição de serviços. Os principais modos são: sequencial, paralela e com escolha. Na composição sequencial um segundo serviço só será invocado depois que o primeiro terminar a sua execução. Na composição com escolha apenas um serviço é escolhido para ser executado dependendo de um critério previamente especificado. Por fim, na composição paralela, diversos serviços são executados de maneira simultânea.

Uma orquestração pode ser representada através de um *workflow*, que pode conter os três tipos citados anteriormente. No contexto dos serviços Web, um grande avanço veio do desenvolvimento do WS-BPEL (*Web Services Business Process Execution Language*), que agora está estabelecida como a maneira padrão de os compor [45].

2.3.1 Workflow

A história dos *workflows* começou ainda no fim dos anos 70 quando estes despertaram atração pela habilidade de descrever um processo de uma maneira simples de entender, modificar, executar e monitorar. A *Workflow Management Coalition* (WfMC) [52] define um *workflow* como a automação de um processo de negócio, ou parte dele, cujos documentos, informações ou tarefas são passadas de um participante para outro para serem processados, de acordo com um grupo de regras [44].

Os sistemas baseados nessa ideia podem ser classificados em uma escala com três níveis [19]: Simples e linear, grafos acíclicos e grafos cíclicos. O primeiro tipo é caracterizado por uma sequência de tarefas em uma ordem linear, em que a primeira tarefa transforma um objeto de dados iniciais em um novo objeto que irá servir de entrada para a próxima tarefa e assim sucessivamente. O segundo nível é formado por um grafo acíclico, cujos nós representam as tarefas a serem executadas e as arestas a dependência entre tarefas. O último tipo é similar ao anterior, porém possui ciclos que representam, de forma implícita ou explícita, uma repetição ou um mecanismo de controle de iterações.

Os principais componentes de um *workflow* podem ser definidos da seguinte maneira:

- **Processo:** Um processo é um conjunto de um ou mais procedimentos interdependentes que, coletivamente, cumprem um objetivo. A divisão entre processos de negócio e processos científicos é comum, diferenciados pelo contexto em que estão inseridos. Respectivamente: envolvendo trocas comerciais e/ou monetárias; e executando experimentos científicos;

- **Atividade:** Uma descrição de uma parte do trabalho a ser realizado em um processo. Uma atividade pode ser básica (ou atômica), ou pode ser um subfluxo, composto de outras atividades. Atividades são os elementos básicos da construção de *workfows*;

- **Sistema Gerenciador de Workflows (SGWf):** Um sistema automatizado para definir e instanciar *workfows*, e gerenciar sua execução. Essa execução pode ocorrer em uma ou mais máquinas de execução, que são capazes de interpretar as definições de um *workfow* e interagir com os seus participantes.

Para sistemas grandes, ou para *workflows* de estruturas mais complexas e dinamicamente estruturadas, é necessário algo que se assemelhe a uma linguagem de programação completa, e WS-BPEL é uma das candidatas para esse propósito [19].

2.3.2 WS-BPEL

A linguagem WS-BPEL [38], *Web Services Business Process Execution Language*, é um padrão criado pela OASIS (*Organization for the Advancement of Structured Information Standards*) para a execução de processos de negócio, descrevendo como ocorre o relacionamento entre os diversos serviços Web participantes da composição. Trata-se de uma linguagem baseada em XML que descreve um *workflow* de serviços [33].

BPEL herdou as características de linguagens anteriores como WSFL da IBM e XLANG da Microsoft e está disponível na versão 2.0. A estrutura principal de um processo de negócio especificado em WS-BPEL é formada por três seções: `PartnerLinks`, `Variables` e `Activities`. Cada uma dessas sessões será discutida posteriormente.

Um processo BPEL executa um *workflow* primariamente acessando um serviço após o outro, conforme a ordem definida na composição. Cada um desses serviços é chamado de `partner service`.

Cada atividade (`activity`) é equivalente a uma chamada de função em uma linguagem de programação. A atividade do tipo `receive` aguarda uma mensagem de entrada, e a atividade do tipo `invoke` transmite uma mensagem, geralmente executando um serviço.

O Código 2.5 retrata um exemplo de *workflow* descrito na linguagem WS-BPEL. Podemos identificar as seguintes etapas:

1. Criação dos `partner links`, que fornecem detalhes sobre a relação entre os processo BPEL e cada `partner service` (linhas 2-4);
2. Atribuição de variáveis, que serão utilizadas durante a execução do processo (linhas 6-9);
3. A criação de uma atividade estruturada do tipo `sequence`, contendo outras atividades (linhas 11-22);
4. O recebimento de uma chamada a função `OperationA` através da uma operação simples do tipo `receive` (linha 12);
5. Atribuição de valores as variáveis criadas anteriormente através a operação `assign` (linhas 14-19);
6. Chamada da operação externa `execSharcnet` através da operação `invoke` (linha 21).

```
1  <process name="Sharcnet">
2      <partnerLinks>
3          <partnerLink name="PLSharcnet" partnerLinkType="AsyncSharcnet"
               partnerRole="R1" />
4      </partnerLinks>
5
6      <variables>
7          <variable name="RunOut" messageType="runResponse" />
8          <variable name="RunIn" messageType="run" />
9      </variables>
10
11     <sequence>
12         <receive name="ReceiveInicio" partnerLink="PLinkClient" operation="
               operationA" portType="MyPTClient" variable="OperationAIn" />
13
14         <assign name="Assign0">
15             <copy>
16                 <from>'VALOR1'</from>
17                 <to>$RunIn.parameters/arg0</to>
18             </copy>
19         </assign>
20
21         <invoke name="InvokeSharcnet" partnerLink="PLSharcnet" operation="
               execSharcnet" inputVariable="ExecSharcnetIn" />
22     </sequence>
23 </process>
```

Código 2.5: Exemplo de um *workflow* descrito na linguagem BPEL. Detalhes foram omitidos.

PartnerLinks

Seção que define os diferentes parceiros (partes envolvidas) que interagem com o processo de negócio durante toda sua execução. Eles são usados para identificar a funcionalidade que deve ser oferecida por cada serviço parceiro. As ligações de parceiros (`partner link`) devem estar associadas a um tipo de ligação entre parceiros (`partner link type`) definido na especificação de serviços Web em WSDL.

Variables

Seção que define as variáveis de dados usadas pelo processo de negócio. As definições são feitas em termos de tipos de mensagem WSDL, elementos ou tipos simples de esquemas XML. Elas são usadas para manter os dados de estado e o histórico do processo com base nas mensagens trocadas. As variáveis devem estar associadas a tipos de mensagens (`messages`) definidos na especificação WSDL.

Activities

Seção que contém a descrição do comportamento normal para a execução do processo de negócio. Existem basicamente dois tipos de atividades, as atividades básicas (`Basic Activities`) e as atividades estruturadas (`Structured Activities`).

Uma `Basic Activity` é um tipo de atividade usado para executar alguma operação. Algumas dessas atividades básicas envolvem a interação com algum parceiro, como: `invoke`, `receive` e `reply`. Outras dessas atividades são executas sem a interação com qualquer parceiro, como: `wait`, `terminate`, `assign`, `empty`, `throw` e `compensate`.

Uma `Structured Activity` é um tipo de atividade usado para agrupar atividades básicas dentro de algumas estruturas de fluxo. Tais atividades são: `while`, `pick`, `flow`, `sequence`, `switch` e `scope`.

Execução de serviços assíncronos

A chamada de um serviço pode ser tanto de forma síncrona quando de forma assíncrona, sendo os dois métodos definidos pela estrutura *invoke* da linguagem BPEL. Chamadas que incluem tanto variáveis de entrada quanto de saída são executadas de forma síncrona e o *workflow* BPEL é bloqueado enquanto o serviço executa. Para execuções assíncronas, o BPEL fornece a notação de grupos de correlação (`Correlation Sets`) que são utilizados para associar `invokes`, contendo somente variáveis de saída, com seus respectivos `replies`, contendo as variáveis de entrada [16].

O solicitante de um serviço assíncrono inicia a interação invocando uma operação de direção única no provedor. Quando o provedor termina a execução, ele retorna o resul-

tado invocando outra operação de direção única no solicitante, conhecida como *callback*. O endereço do de resposta do solicitante e os valores utilizados para criar a correlação são enviados do cabeçalho das mensagens utilizando o WS-addressing, explicado anteriormente na Sessão 2.2.2. O provedor utiliza, então, o grupo de correlação para especificar qual instância de processo a mensagem de *callback* se refere [16].

A Figura 2.5 retrata um diagrama de interação demonstrando um exemplo de um processo WS-BPEL executando um serviço Web de forma assíncrona. As seguintes ações podem ser vistas:

1. No processo BPEL, uma atividade do tipo `invoke` inicia um serviço montando uma mensagem SOAP;

2. Um cabeçalho WS-addressing é adicionado à mensagem SOAP, informando o endereço do serviço que aguarda a mensagem de *callback*, e o identificador de correlação;

3. Usando a descrição WSDL do serviço, uma porta é iniciada e os dados para o serviço assíncrono são enviados;

4. O serviço assíncrono processa a requisição e envia uma resposta de volta ao processo BPEL através de uma porta de *calback*. O cabeçalho WS-addressing é montado com o identificador de correlação apropriado (o mesmo da mensagem `invoke`);

5. A máquina de composição BPEL verifica o identificador de correlação na mensagem de *callback* e identifica a instância que a aguardava.

2.3.3 WS-BPEL e os Multi-Clusters

Pesquisas recentes sugerem a integração de duas tecnologias existentes, *workflows* e sistemas distribuídos, como uma solução para problemas que são ao mesmo tempo complexos e necessitam de um grande poder computacional.

Os *workflows* têm sido revistos para aplicações, científicas e comerciais, de alta intensidade que executam em sistemas distribuídos, como *Multi-Clusters* ou grades computacionais. Apesar de diversos serviços, incluindo autenticação, gerenciamento, descoberta e acesso a recursos remotos, estarem disponíveis aos desenvolvedores de aplicações distribuídas, esses serviços não são suficientes para resolver problemas complexos. Nesse caso, os serviços precisam ser compostos e executados de forma simples, levando em consideração suas dependências. A necessidade de se representar *workflows* científicos e comerciais em sistemas distribuídos já foi destacada por diversos pesquisadores [3, 53], e

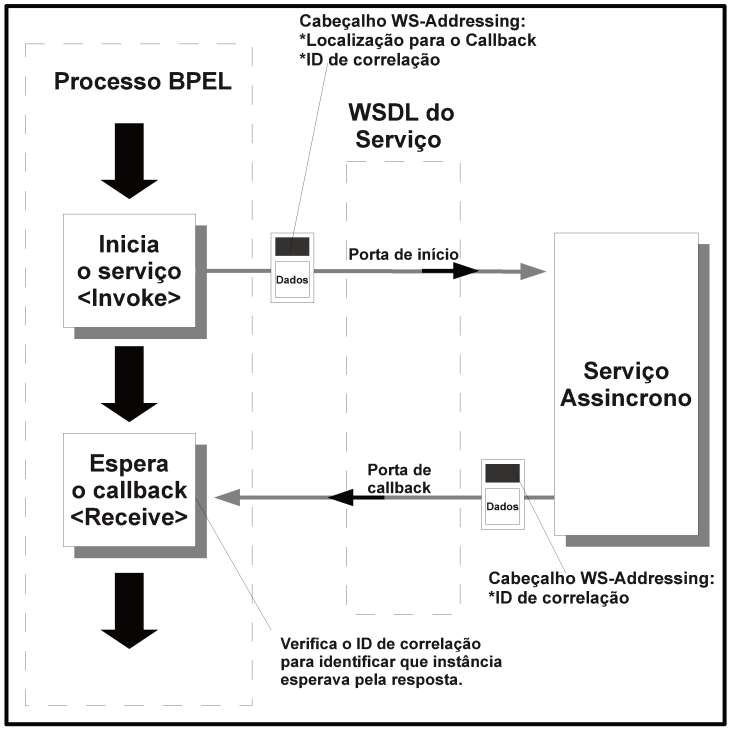

Figura 2.5: Diagrama de interação de um processo WS-BPEL executando um serviço Web de forma assíncrona.

a inadequações dos conceitos e padrões existentes foi reconhecida pelo *Global Grid Forum* [19].

Visto que os serviços Web foram adotados na infraestrutura desses sistemas heterogêneos, WS-BPEL, o padrão para composição de serviços, é um forte candidato para a especificação de *workflows* complexos [25]. O WS-BPEL provê um processo abstrato para a descrição de interações entre parceiros de negócio, além de tratamento de falhas e exceções, com uma linguagem para a definição da lógica de um processo executável em termos de Serviços Web e dados em XML [19].

Além do mais, WS-BPEL é vantajoso por existirem máquinas de execução, comerciais e de código aberto, disponíveis para a interpretação e execução dos *workflows*, facilitando a interoperabilidade. Leymann [29] sugere que o WS-BPEL deve ser estendido aos ambientes distribuídos ao invés de criar-se um novo padrão. Entretanto, o fluxo de atividade dessas arquiteturas possuem requisitos específicos, como gerenciamento de ciclo de vida e a seleção de recursos baseada em requisitos não-funcionais [26].

Capítulo 3

Trabalhos Relacionados

Neste capítulo serão discutidos os principais trabalhos relacionados à presente dissertação. Primeiramente são levantados os trabalhos com o foco na execução de *workflows* em sistemas distribuídos heterogêneos. Em seguida algumas pesquisas sobre escalonamento global em *Multi-Clusters* são apresentadas.

Como o intuito manter este trabalho sucinto, na Sessão 3.1 serão relacionados apenas os trabalhos que utilizam a linguagem WS-BPEL para expressar os *workflows*. Contudo, também existem pesquisas com objetivos similares mas direcionadas a criação de novas linguagens [13,26,47]. Outro ponto importante é o fato de a grande maioria dos trabalhos utilizarem como infraestrutura grades computacionais. Até a data de escrita deste trabalho não foram encontradas pesquisas que tivessem um *Multi-Clusters* como infraestrutura alvo.

Apesar de não ser objetivo primário deste trabalho, criar um escalonador global funcional teve grande importância. Na Sessão 3.2 são listados os trabalhos com esse intuito.

3.1 Execução de Workflows em Sistemas Distribuídos Heterogêneos

Como serviços Web já foram adotados nas infraestruturas de sistemas distribuídos e como BPEL oferece os requisitos de interoperabilidade, e permite o uso de máquinas de *workflow* disponíveis, open-source ou industriais, ela é candidata a ser adotada também na comunidade de grades [25]. Leymann [29] sugere que BPEL seja estendida para esses sistemas ao invés de se criar um novo padrão. Contudo, fluxos de atividades para aplicações desse tipo de infraestruturas resultam em requisitos específicos [29].

WS-BPEL é utilizado em diversos projetos [16, 18, 29, 32, 44]. De fato, a maioria dos autores concorda que a WS-BPEL, com algumas modificações, pode e deve ser adotada

para a composição de serviços de grade. Até mesmo autores que não a utilizam, como Cybok [13], reconhecem sua importância a consideram como opção em pesquisas futuras.

3.1.1 Leymann

Leymann [29] discute como os requisitos específicos de grades podem ser cumpridos pela linguagem BPEL. Alguns dos requisitos discutidos são: tempo de vida e propriedades dos recursos, monitoramento e tratamento de transações.

O autor discute extensivamente as vantagens e desafios para a utilização do BPEL em sistemas distribuídos heterogêneos e conclui que a comunidade de grades deve considerar o BPEL como base para composições ao invés de criar novas linguagens. Segundo o autor, hoje a linguagem BPEL está perto de atender os requisitos para a execução de *workflows* em grades, precisando apenas de novas funcionalidades como, por exemplo, monitoramento.

3.1.2 Slomiski

Slomiski [44] discute os benefícios e desafios no uso de BPEL em ambientes de grade. O autor descreve suas experiências na tentativa de verificar se essa linguagem pode ser utilizada para facilitar a execução de *workflows* grades OGSI, *Open Grid Services Infrastructure*, e WSRF, *Web Services Resource Framework*, (Tipos de arquitetura mais utilizadas em grades computacionais) e as compara. Vale ressaltar que esse trabalho se limitou a usar uma linguagem existente, no caso o BPEL, e trabalhar somente com o que ela já disponibilizava, sem acrescentar nenhum tipo de funcionalidade.

Inicialmente o trabalho discute como a linguagem BPEL pode ser integrada com uma grade baseada em OGSI. Levanta divergências entre as especificações e propõe como solução para alguns dos problemas a criação de um *proxy*.

Em seguida, o trabalho discute a integração da linguagem BPEL com uma grade baseada em WSRF. Por ser uma especificação mais recente e mais direcionada aos serviços Web, a integração se torna muito mais simples. Entretanto, algumas dificuldades ainda foram encontradas. Entre elas, o autor destaca a dificuldade de se criar mecanismos de notificação e monitoramento.

Ao fim do trabalho, o autor argumenta que BPEL pode cumprir os requisitos para a execução de *workflows* em grades com algumas adaptações para OGSI e WSRF.

3.1.3 Emmerich et al.

Em seu trabalho Emmerich et al. [16] descrevem suas experiências na orquestração de *workflows* científicos usando BPEL, obtidas durante um estudo de caso compondo serviços

de grade. Seus objetivos eram mostrar em que medida BPEL pode ser usado para a orquestração e descobrir se o mesmo é expressivo o suficiente para *workflows* científicos.

Primeiramente foi escolhido um problema representativo para o estudo de caso. Foi escolhido um *workflow* científico da área de teoria química, mais especificamente predição computacional de estruturas de cristais orgânicos a partir de um diagrama químico. Em particular, o estudo replicou um experimento que foi codificado manualmente e executado em um servidor comum, codificando-o em BPEL e o executando em uma grade computacional.

Em seguida, foi utilizada a ferramenta ActiveBPEL para definir e executar o *worflow* e foram medidos o desempenho, escalabilidade e confiabilidade da solução.

Nesse trabalho também foi discutido como derivar os requisitos para uma composição de serviços de grade e como BPEL trata cada um deles. Os principais desses requisitos são:

- **Definição:** Invocar serviços de grades, controlar fluxo de controle e de dados, permitir composição hierárquica e tratamento de falhas;
- **Implantação:** Possibilitar testes e implantação dos *workflows*;
- **Execução:** Tratar de concorrência, escalabilidade e monitoramento.

Concluiu-se então que, em resumo, BPEL satisfaz a maioria desses requisitos. Já que foi possível expressar a aplicação do estudo de caso por completo. Contudo, os resultados não foram tão bons quanto poderiam. Foi verificado que a linguagem poderia ser mais expressiva, pois acabou se tornando muito difícil para ser interpretada e modificada por humanos, e com um excesso de informação redundante. Além de ser uma linguagem muito complexa, prejudicando a usabilidade e tornando uma ferramenta gráfica acessório necessário.

3.1.4 Ezenwoye et al.

Em sua pesquisa, Ezenwoye et al. [18] demonstram experiências utilizando BPEL para integrar, criar e gerenciar recursos de grades baseadas em WSRF e que implementam um padrão de fábrica de objetos. O autor apresenta um trabalho similar ao de Emmerich, entretanto utiliza um tipo diferente de aplicação, nesse caso uma aplicação de bioinformática para mapeamento de sequências de proteínas.

O autor destaca problemas com a transferência de dados e demonstra os principais pontos de ajustes. Como conclusão obteve-se sucesso na orquestração sendo o primeiro trabalho a incluir a descoberta de serviços.

3.1.5 Ma et al.

Em Ma et al. [32] os autores têm como objetivo projetar e implementar um sistema de gerenciamento de *workflows* baseados em BPEL em um sistema de grade. Sua ideia principal é integrar a linguagem BPEL com um sistema WSRF através da implementação de um protótipo.

Esse trabalho define uma sequência de atividades para invocação de um serviço WSRF a partir de um processo BPEL. Em seguida propõe uma extensão à máquina de *workflow* ActiveBPEL para conseguir a integração sem modificar a especificação da linguagem BPEL e implementa um protótipo.

Concluindo, o trabalho consegue executar os *workflows* com sucesso, integrando WSRF com BPEL sem modificar a linguagem, mas alterando a máquina de execução.

3.2 Escalonamento Global em Multi-Clusters

Criar um escalonador global, ou Meta-Escalonador, para um *Multi-Clusters* não é um objetivo primário deste trabalho. Entretanto, como o ambiente utilizado para a implementação (descrito na Sessão 4.3.1) não possui esse tipo de estrutura, criá-la se tornou fundamental.

Por se tratar de um ambiente real, alguns aspectos particulares devem ser levados em consideração. Entre eles a impossibilidade de grande alterações na estrutura e a necessidade de se manter a compatibilidade com as aplicações que atualmente a utilizam. Mais detalhes sobre a situação problema e o escalonador são tratados na Sessão 4.3.2.

3.2.1 Chau e Fu

No trabalho de Chau e Fu [9] os autores apresentam uma proposta de método de balanceamento de carga em *clusters* heterogêneos e distantes entre si. A pesquisa assume que os *clusters* estão conectados em forma de hipercubo e demonstra apenas matematicamente que, após aplicar seu algoritmo, a diferença de carga entre quaisquer dois *clusters* é no máximo uma unidade maior que a ótima.

Entretanto, esse método exige que tarefas sejam realocadas, o que requer modificações no funcionamento dos *clusters* e, portanto, não mantém compatibilidade com os processos já existentes.

3.2.2 Takpé e Suter

Takpé e Suter [37] propõem modificações em trabalhos anteriores aplicando-os no escalonamento de aplicações paralelas em plataformas heterogêneas. Esse trabalho é mais

abrangente e se aproxima mais das necessidades da presente dissertação. Os autores não assumem um tipo de conexão em forma de hipercubo e não permitem a a realocação de tarefas.

Entretanto, os dois algoritmos apresentados assumem que o escalonador é central e tem conhecimento de todas as novas tarefas. Portanto, necessitaria de alterações no modo no qual os *clusters* são utilizados e não poderiam ser aplicados à SHARCNET. Os resultados foram demonstrados apenas com simulações, não sendo aplicados em um ambiente real.

3.2.3 Qin e Bauer

Qin e Bauer [41] propõem um método para a coalocação de tarefas em um *cluster* de *clusters*. A ideia principal é de, ao invés de aguardar que um *cluster* esteja plenamente disponível para executar uma dada tarefa, essa seja dividida em diversas subtarefas e cada uma delas seja alocada em um *cluster* diferente, com quantidades menores de processadores disponíveis. Os resultados são demonstrados apenas através de simulações.

Essa proposta também necessitaria de modificações no ambiente, pois permite coalocação. Além de assumir que o escalonador tem conhecimento de todas as tarefas.

3.2.4 Hunold, Rauber e Suter

No trabalho de Hunold, Rauber e Suter [23] é proposto um método de escalonamento dinâmico de *workflows* em um *cluster* de *clusters*. Os resultados são demonstrados através de simulações.

Esse algoritmo, no entanto, precisa saber do tempo estimado de execução de uma tarefa, o que nem sempre é possível. Outro contratempo é a técnica de adiamento utilizada, pois, quando o escalonador não possui controle total sobre as tarefas, este método pode gerar um período de espera indeterminado aguardando a disponibilidade de um *cluster*. Portanto, podendo ocorrer *starvation*.

3.2.5 Beltrán e Guzmán

Beltrán e Guzmán [7] demonstram uma proposta para o balanceamento de carga em *clusters* heterogêneos.

O trabalho demonstra um algoritmo é simples e escalável. Além de apresentar uma implementação e testes em um ambiente real. Contudo, a proposta necessita de ferramentas para que uma tarefa seja retirada de um *cluster* e realocada para outro. O que, mais uma vez, não permite a compatibilidade com as aplicações já existentes no *cluster*, e necessita de modificações no ambiente.

3.3 Discussão

Vários estudos [18,29,32,44,47] tratam da execução de *workflows* em grades OGSI (Open Grid Services Infrastructure) [48] e WSRF (WS-Resource Framework) [14]. Contudo, até a data de escrita deste trabalho não foram encontradas pesquisas que tivessem um CoC como infraestrutura alvo. Mas, diversas ideias anteriormente apresentadas puderam ser aproveitadas. Outro fator de destaque para a presente dissertação é o fato de ser a única proposta a contemplar a seleção de serviços baseada em atributos de QoS e necessidade de recursos, como poder computacional e quantidade de memória.

A Tabela 3.1 compara os trabalhos anteriores de execução de *workflows* em sistemas distribuídos heterogêneos com o proposto nesta dissertação. A primeira coluna lista os trabalhos a serem comparados. As três colunas seguintes definem qual tipo de infraestrutura é compatível com cada trabalho, WSRF, OGSI ou CoC. Em seguida são definidas outras propriedades relevantes como o fato de a proposta utilizar BPEL como linguagem de composição, se ela contempla a descoberta de recursos, e se contempla a seleção de serviços baseada em políticas de recursos e de QoS.

Tabela 3.1: Tabela comparativa de trabalhos correlatos: Execução de *workflows* em sistemas distribuídos heterogêneos.

Autor	**Arquitetura Alvo**			**Outras Propriedades**		
	CoC	**WSRF**	**OGSI**	**BPEL**	**Descoberta**	**Políticas**
Leymann	Não	Não	Não	Sim	Não	Não
Slomiski	Não	Sim	Sim	Sim	Não	Não
Emmerich	Não	Não	Sim	Sim	Não	Não
Ezenwoye	Não	Sim	Não	Sim	Sim	Não
Ma	Não	Sim	Não	Sim	Não	Não
Lechuga	**Sim**	**Não**	**Não**	**Sim**	**Sim**	**Sim**

Diversos pesquisadores [7, 9, 23, 37, 41] demonstram métodos de escalonamento para *Multi-Clusters*. Até então, a maioria das propostas nesse sentido se limitou a provas formais e/ou simulações, sem implementação em um ambiente real, o que implica na superação de diversos problemas, como por exemplo a necessidade de se manter compatibilidade com os métodos de seleção já existentes. Essa necessidade invalida algumas propostas, especialmente aquelas que assumem que todas as requisições são direcionadas através do novo escalonador global [7, 37, 41].

Hunold et al. [23] propõe um método de escalonamento utilizando uma técnica de adiamento. No entanto, quando o escalonador não possui controle total sobre as tarefas, este método pode gerar um período de espera indeterminado aguardando a disponibilidade de um *cluster*, podendo ocorrer *starvation*.

A Tabela 3.2 compara os trabalhos anteriores de escalonamento de tarefas entre HPCCs com o proposto nesta dissertação. A primeira coluna lista os trabalhos a serem comparados. A segunda define se o escalonador precisa ser centralizado e conhecer todas as requisições ao CoC. Em seguida os trabalhos são diferenciados quanto a necessidade de alterações à infraestrutura do CoC. A quarta coluna especifica quais trabalhos apresentam uma implementação em ambiente real. Finalmente, a última coluna define quais propostas mantém compatibilidade com as aplicações já existente no sistema.

Tabela 3.2: Tabela comparativa de trabalhos correlatos: Escalonamento global em *Multi-Clusters*.

Autor	**Problemas**		**Vantagens**	
	Centralizado	**Alterações**	**Implementação**	**Compatibilidade**
Chau	Não	Sim	Não	Não
Takpé	Sim	Não	Não	Não
Qin	Sim	Sim	Não	Não
Hunold	Não	Não	Não	Não
Beltrán	Não	Sim	Sim	Não
Lechuga	**Não**	**Não**	**Sim**	**Sim**

Capítulo 4

Proposta de Infraestrutura para a Execução de Workflows em Multi-Clusters

Este capítulo tem como objetivo descrever com detalhes as principais contribuições desenvolvidas neste trabalho.

A Sessão 4.1 explica as adaptações feitas na linguagem WS-BPEL para permitir que requisitos de QoS e recursos necessários para a execução das tarefas sejam expressos em cada serviço e para toda a composição de serviços, condição necessária para a execução em CoCs. Na Sessão 4.2 esta dissertação propõe uma arquitetura para a execução de *workflows* descritos em WS-BPEL que utilizam recursos de um CoC, permitindo a seleção de recursos baseada em requisitos funcionais e não-funcionais. Finalmente, a Sessão 4.3 demonstra uma aplicação dessa arquitetura utilizando a rede SHARCNET como *Multi-Cluster*, e ilustra detalhes da implementação, como o `Escalonador Global` (Sessão 4.3.2), o `Broker` (Sessão 4.3.2), o `Mediador` (Sessão 4.3.2), o `Monitor` (Sessão 4.3.2), e outros elementos da arquitetura.

4.1 Estendendo o WS-BPEL para especificação de QoS

Para permitir a especificação e execução de serviços em sistemas distribuídos heterogêneos, diversos aspectos devem ser considerados. Entre eles, está a especificação dos recursos necessários para a execução, como memória, disco e poder de processamento, e atributos de QoS; permitindo a seleção de serviços baseada nessas especificações. De acordo com isso, este trabalho propõe uma extensão ao WS-BPEL compreendendo os seguintes aspectos:

- Inclusão de informações sobre recursos necessários para a execução; Serviços podem ter necessidades relacionadas a aspectos como poder de processamento, capacidade de memória e disco.
- Incorporação de requisitos de QoS dos usuários dos serviços e a capacidade de QoS dos provedores;
- Seleção de serviços baseada em propriedades funcionais, atributos de QoS e recursos requeridos.

Como visto anteriormente na Sessão 2.3.2, a linguagem WS-BPEL utiliza somente o padrão WSDL como descrição para os WSs. Na arquitetura atual, esse padrão é baseado somente nas funcionalidades dos serviços sem considerar outros aspectos. Para suprir essa necessidade, permitindo a seleção de serviços baseada em requisitos não-funcionais e satisfazer a necessidade dos consumidores, utilizamos principalmente o WS-Policy, especificando esses requisitos através de políticas.

O Código 4.1 retrata um exemplo de política utilizada para especificar uma necessidade de um grau de processamento elevado (linha 8).

```
1  <wsp:Policy>
2      <wsp:ExactlyOne>
3        <wsp:All>
4          <resp:CPU
5            xmlns:qosp="../schema/respolicy"
6            operation="InvokeSharcnet"
7            specification="uddi:qos:attribute:
8            CPUPower">high</resp:CPU>
9        </wsp:All>
10     </wsp:ExactlyOne>
11 </wsp:Policy>
```

Código 4.1: Exemplo de política para necessidade de recursos.

Dessa forma, provedores podem descrever seus serviços usando WSDL para funcionalidade e WS-Policy para QoS e recursos necessários, referenciando suas políticas no WSDL. Nessa abordagem, os aspectos de QoS, associados às interfaces, e as descrições em WSDL são armazenados em arquivos distintos, permitindo que especificações de requisitos não-funcionais sejam alteradas sem mudar os arquivos WSDL, que são geralmente mais estáveis.

Do outro lado, os consumidores de serviços descrevem composições usando WS-BPEL. Nesta proposta, a linguagem é estendida com referências para políticas em WS-Policy. Quando uma política é especificada para um serviço, a seleção do mesmo é realizada usando os atributos funcionais, de recursos e de QoS.

As políticas são incluídas na composição de acordo com um mecanismo definido pelo padrão WS-PolicyAttachment [50]. O WS-PolicyAttachment define o atributo `PolicyURIs` que liga uma política em WS-Policy a um elemento XML. Nessa proposta, os atributos `PolicyURIs` permitem que políticas sejam ligadas a elementos WS-BPEL que representam composições de serviços ou serviços incluídos em composições.

O atributo `PolicyURIs` inclui uma lista de URIs. Cada URI identifica uma política em WS-Policy. Se houver mais de uma política no atributo `PolicyURIs`, as políticas devem ser unidas para formar uma única política. A política resultante é então associada com o elemento que contém o atributo `PolicyURIs`. A operação `merge` do padrão WS-Policy combina políticas em uma única, incluindo o conteúdo das políticas originais.

No Código 4.2 vemos um fragmento de uma composição. O serviço tem duas políticas associadas (Linhas 07-08). Essa política especifica requisitos não funcionais aplicados à operação `Purchase` (Linha 04) do serviço `Purchasing` (Linha 03). O serviço é fornecido pelo participante `Seller` (Linha 02), indicado na atividade de invocação (Linha 01).

```
1 <b:invoke
2   partnerLink="Seller"
3   portType="a:Purchasing"
4   operation="Purchase"
5   inputVariable="SendOrder"
6   outputVariable="getResponse"
7   p:PolicyURIs="...policies#RM
8   ...policies#SEC">
```

Código 4.2: Exemplo de política em uma composição de serviços.

Em tempo de execução, a seleção de serviços é baseada em descrições funcionais e em políticas para recursos e QoS. Primeiramente os serviços são selecionados pela sua funcionalidade, em seguida, as políticas de consumidores e fornecedores são comparadas e combinadas. Esse processo será descrito com mais detalhes na Sessão 4.2.

4.2 Proposta de Arquitetura

A arquitetura proposta, retratada na Figura 4.1, apoia a execução de *workflows* descritos em WS-BPEL que podem utilizar recursos de um CoC. O termo *workflow* é utilizado para uma composição de serviços. Serviços são descritos em termos de funcionalidade, utilizando WSDL, e em termos de requisitos de recursos e atributos de QoS, utilizando WS-Policy. Os principais elementos da arquitetura incluem: (a) A `Máquina de Composição`, que interpreta e executa os *workflows* em WS-BPEL; (b) Os `Provedores Comuns`, provedores que não participam de um CoC; (c) Os `Provedores de CoC`, que permitem que a

`Máquina de Composição` invoque serviços no CoC; (d) O `Mediador` faz a interface entre a `Máquina de Composição` e os `Provedores de CoC`, responsável por garantir que os requisitos de recursos sejam atendidos; (e) O `Broker`, parte principal do UDDI estendido proposto por Garcia e Toledo [20], que seleciona implementações de serviço baseando-se na compatibilidade de requisitos de QoS entre consumidor e fornecedor; e (f) O `Repositório UDDI`, que armazena as descrições dos serviços.

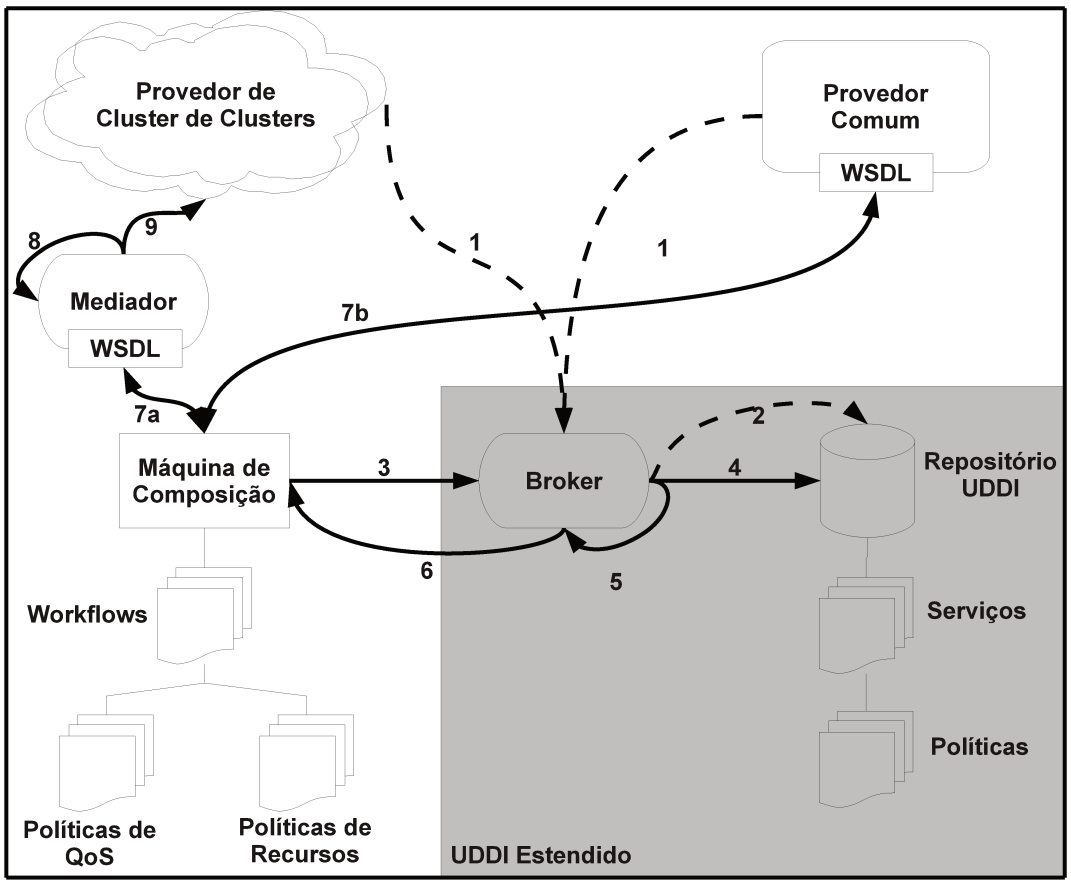

Figura 4.1: Arquitetura proposta para execução de *workflows* descritos em WS-BPEL que podem utilizar recursos de um CoC. Componentes em branco foram desenvolvidos nesta dissertação e componentes em cinza foram propostos por Garcia e Toledo [20].

O `Broker` é responsável por publicar as descrições dos serviços no `Repositório UDDI`. Assim, os provedores de serviço precisam enviar suas políticas de QoS de seus serviços para o `Broker`, junto com a informação funcional tipicamente armazenada em um registro UDDI. Durante a execução, a `Máquina de Composição` é configurada com uma

composição em WS-BPEL. Em sequida, quando o *workflow* é iniciado e um serviço precisa ser executado, a `Máquina de Composição` envia ao `Broker` um pedido de seleção de implementação. A requisição inclui requisitos funcionais, que indicam o tipo de implementação necessária, e politicas de QoS e de recursos associadas ao serviço, indicando os requisitos do usuário, como, por exemplo, poder de processamento necessário (política de recurso) e tempo limite de resposta (política de QoS).

Portanto, para cada serviço no *workflow*, o `Broker` seleciona uma implementação entre os serviços publicados no `Repositório UDDI`. Esta seleção compreende três fases; na primeira, o `Broker` seleciona serviços com as funcionalidades necessárias no UDDI, utilizando o mecanismo de busca do mesmo. Um conjunto de serviços é retornado ao `Broker`. Em seguida, as políticas dos consumidores e fornecedores são combinadas, permitindo ao `Broker` selecionar serviços adequados. Finalmente, o `Broker` envia os endereços de serviços para a `Máquina de Composição`, que invoca a tarefa. Se o serviço deve ser executado no CoC, a tarefa é enviada ao `Mediador`, que se encarrega de verificar os recursos necessários, submeter a tarefa, e a monitorar. Caso contrário, um `Provedor Comum` é invocado.

Em tempo de especificação, provedor e consumidor devem:

1. O provedor faz uma requisição para a publicação de um serviço, especificando a descrição funcional (WSDL) e as políticas (WS-Policy) (Passo 1 na Figura 4.1), essa informação é passada para o `Repositório UDDI`;

2. O consumidor especifica sua composição suas políticas para cada serviço.

Em tempo de execução:

1. Quando uma chamada de serviço é alcançada, a `Máquina de Composição` requisita uma seleção de implementação provendo requisitos funcionais e políticas ao `Broker` (Passo 3 na Figura 4.1);

2. O `Broker` busca serviços no `Repositório UDDI` de acordo com as funcionalidades necessárias (Passo 4) e faz o casamento das políticas dos consumidores com a dos fornecedores (Passo 5);

3. O `Broker` informa os serviços selecionados à `Máquina de Composição` (Passo 6);

4. Para serviços que devem ser executados em um `Provedor de CoC`, a `Máquina de Composição` o executa usando o `Mediador` (Passo 7a);

5. O `Mediador` funciona como um *proxy*, seleciona os recursos do CoC de acordo com os requisitos definidos nas políticas (Passo 8), envia o serviço a ser executado (Passo 9), o monitora e envia a resposta de volta para a `Máquina de Composição`;

6. Para serviços comuns, a `Máquina de Composição` invoca a operação no serviço selecionado através do ponto de acesso do `Provedor Comum` retornado (Passo 7b).

A abordagem sugerida permite que novas aplicações, desenvolvidas como composições WS-BPEL, executem em conjunto a aplicações antigas, sem requerer nenhum tipo de modificação. Um estudo de caso com exemplo de de aplicação dessa arquitetura será mostrado em seguida.

4.3 Estudo de caso: SHARCNET

Para avaliar a arquitetura proposta era necessário implementá-la em um *Multi-Clusters* real, para isso foi escolhida a rede SHARCNET.

4.3.1 SHARCNET

Estruturada em forma de um *Cluster* de *Clusters*, a SHARCNET é uma rede multi-institucional de *clusters* de alta performance distribuídos em dezesseis instituições acadêmicas na província de Ontário, no Canadá [41]. Atualmente, esse sistema abriga pesquisas tanto das diversas áreas que tradicionalmente necessitam de computação de alta performance, como química, física, ciência dos materiais e engenharia, quanto estudos de outras áreas, incluindo negócios, economia e biologia.

Essa rede tem três *clusters* primários que já foram listados no *Top 500* [46] de super-computadores. Respectivamente, esses três *clusters* contêm 267 nós com 1068 processadores, 768 nós com 1536 processadores e 768 nós com 3072 processadores; cada um deles provê 70TB de armazenamento de alta velocidade. Além disso, também existem outros 17 *clusters* secundários, variando de 32 a 64 nós e totalizando mais de 8.000 processadores e 200TB de armazenamento. A interconexão entre os nós em cada um inclui G2 Myrinet, Quadrics (Elan, Elan 4 e 3) e Gigabit Ethernet; a comunicação entre eles é realizada através de uma rede dedicada de alta velocidade e de 10 Gibabit Ethernet. Em conjunto, todos esses recursos criam uma "grade computacional", de alta performance [6].

Os sistemas operacionais são variantes do Linux. O escalonamento de tarefas é realizado pelo Load Sharing Facility (LSF) e a computação paralela pode ser implementada utilizando MPI ou OpenMPI. As contas são gerenciadas pelo Lightweight Directory Access Protocol (LDAP), que permite que os pesquisadores utilizem apenas uma conta para acessar qualquer um dos *clusters* [6].

Os *clusters* da rede estão situados a uma distância considerável uns dos outros e são conectados por uma rede dedicada de fibra ótica. Cada um deles tem um nó central que atua como escalonador local; entretanto, não existe um escalonador global, capaz de escalonar tarefas entre *clusters* distintos. Portanto, cada usuário deve manualmente

verificar a carga atual e a capacidade de cada um, e escolher o que atende melhor os seus requisitos [8].

Assim, o balanceamento e a seleção de recursos não é transparente, como deveria ser em uma grade computacional. Assim, a SHARCNET não é capaz de executar tarefas de maneira automática, como no caso de um sistema de gerência de workflows. Consequentemente, os usuários acabam focando nos mesmo recursos, sobrecarregado um único *cluster*, mesmo que existam outros com uma carga menor. Além do mais, os usuários são obrigados a modificar manualmente a submissão de tarefas quando o *cluster* que está sendo utilizado tem algum tipo de problema.

Por esse motivo, este trabalho propõe um escalonador global para a SHARCNET, permitindo a submissão automática de tarefas e a execução de *workflows*.

Como discutido anteriormente, a seleção de *clusters* será baseada em requisitos de QoS e de recursos especificados na composição descrita pelo WS-BPEL estendido, também proposto neste trabalho. Essa abordagem não requer modificações na estrutura do sistema, portanto mantém a compatibilidade com a seleção manual de recursos utilizada até então e permite que aplicações existentes se mantenham inalteradas.

O escalonador global é discutido na Sessão 4.3.2 e a arquitetura implementada para a execução de *workflows* na SHARCNET é apresentada na Sessão 4.3.2.

4.3.2 Arquitetura Aplicada

A Figura 4.2 retrata a arquitetura aplicada para a execução de *workflows* que podem utilizar recursos da SHARCNET. Alguns novos elementos tiveram que ser incluídos para atender requisitos específicos desse ambiente. Primeiramente, a `Máquina de Composição` foi alterada para invocar as tarefas e receber a resposta correspondente através de uma `Interface de Callback`. Segundo, o `Escalonador Global`, explicado na Sessão 4.3.2, foi adicionado ao `Mediador`. Finalmente, o `Monitor` foi adicionado, com a responsabilidade de monitorar as tarefas em execução na SHARCNET e, quando estas estiverem completas, enviar os resultados para a `Máquina de Composição`.

O usuário inicia uma execução de *workflow*. A `Máquina de Composição` invoca o `Broker` buscando serviços que atendam as especificações funcionais e de QoS. A `Máquina de Composição` invoca o `Mediador` como um serviço Web assíncrono, informando os requisitos de recursos utilizando o WS-PolicyAttachment em conjunto com o WS-Policy e a `Interface de Callback` utilizando o WS-Addressing. O `Mediador` atua como um *proxy*, seleciona o *cluster* mais adequado utilizando o `Escalonador Global`, e submete a tarefa. Em seguida, o serviço `Monitor` é criado para monitorar a tarefa até que ela esteja completa. Então, o resultado é enviado ao endereço de *callback* especificado com o WS-Addressing, a tarefa é considerada completa e o *workflow* continua a execução.

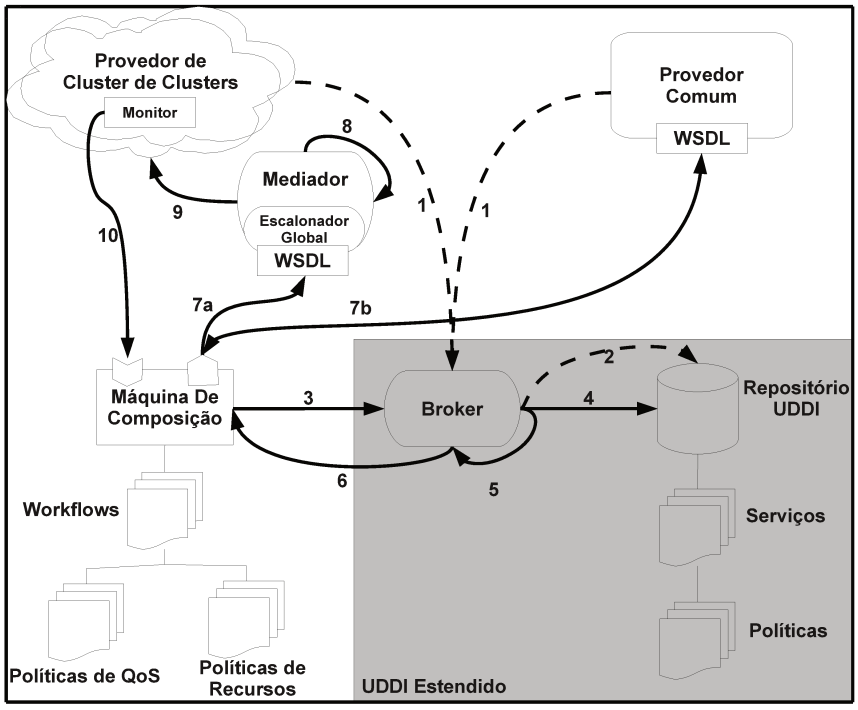

Figura 4.2: Arquitetura implementada.

A arquitetura implementada permite a execução de *workflows* utilizando recursos da SHARCNET de maneira automática e transparente. Os detalhes de cada componente da arquitetura são explicados em seguida.

Máquina de Composição

O componente mais importante para a execução de um processo BPEL é a `Máquina de Composição`. A utilizada neste trabalho foi a *BPEL Service Engine*, um *plugin* para a execução de *wokflows* no pacote Glasfish Open ESB do Netbeans [40]. Toda a arquitetura foi adaptada para utilizar as funcionalidades providas por padrão por esse componente, portanto não foi necessário modificá-lo.

Broker

Parte principal do UDDI estendido proposto por Garcia e Toledo [20]. Este componente funciona como um WS fazendo uma interface para o repositório UDDI. Ao invés do usuário (`Máquina de Execução`) fazer uma busca por serviços utilizando o UDDI diretamente, ele

utiliza o `Broker`, que faz as buscas baseadas em requisitos funcionais e de QoS. Como retorno, o `Broker` envia o endereço do serviço mais indicado.

Essa etapa de busca deve ser adicionada explicitamente pelo usuário na composição BPEL. Em seguida a `Máquina de Composição` deve ser configurada para, em tempo de execução, utilizar o endereço recebido como resposta do `Broker` para o serviço a ser executado em seguida. Essa técnica é conhecida como *Binding* Dinâmico.

A Figura 4.3 retrata esta funcionalidade no Netbeans. O parâmetro `return`, da variável que recebe a saída do `Broker`, é atribuído ao parâmetro URL, das propriedades SOAP do serviço `ExecSharcnetIn`. O Código 4.3 mostra o trecho de código BPEL associado a esta operação.

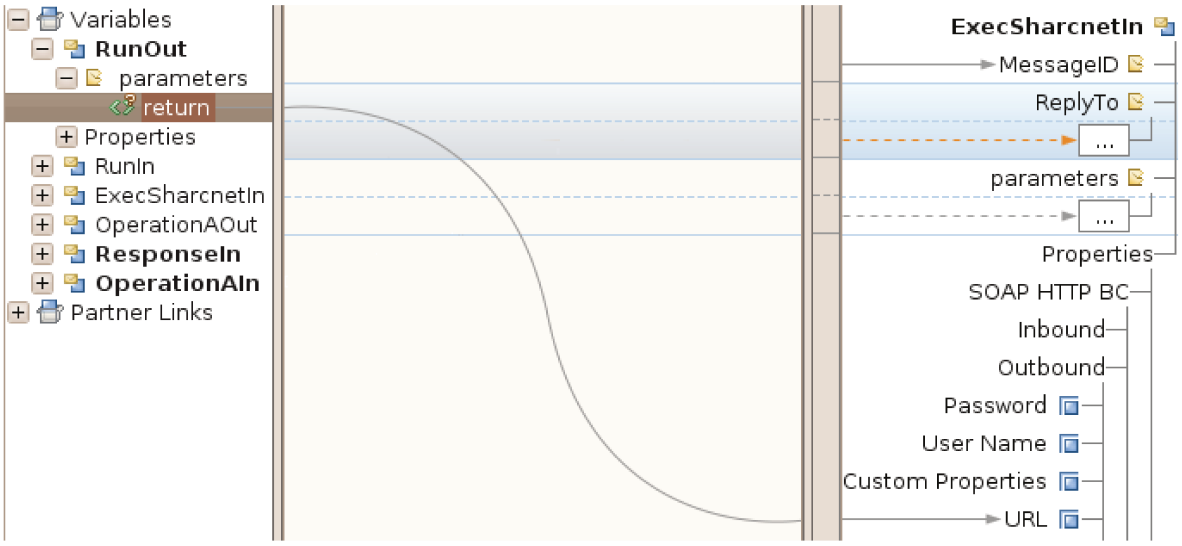

Figura 4.3: Configurando um *binding* dinâmico no Netbeans.

```
1 <assign name="Assign1">
2     <copy>
3         <from>$RunOut.parameters/return</from>
4         <to variable="ExecSharcnetIn" sxnmp:nmProperty="org.glassfish.
              openesb.outbound.address.url"/>
5     </copy>
6 </assign>
```

Código 4.3: Trecho de código demonstrando um *Binding* Dinâmico.

UDDI

O repositório UDDI utilizado pelo `Broker` é o JUDDI [4]. O funcionamento detalhado desse componente da arquitetura é descrito na Sessão 2.2.1.

Aqui, seu papel é armazenar os serviços publicados e auxiliar o `Broker` nas buscas por serviços.

Mediador

O `Mediador` é um elemento importante da arquitetura. Se trata de um WS desenvolvido em Java que atua como um *proxy*, fazendo a interface entre a `Máquina de Composição` e a SHARCNET. Todas as requisições direcionadas a ela devem ser enviadas ao `Mediador`. Ele é responsável por gerenciar a execução das tarefas. Verifica os requisitos descritos pelo usuário em suas políticas e, utilizando outros componentes da arquitetura, encontra o melhor *cluster* para executar a tarefa e monitora a sua execução.

Esse componente, como a maior parte da estrutura, não é centralizado. Portanto, cada cliente pode ter um `Mediador` rodando localmente, mantendo a característica distribuída do sistema.

Inicialmente o serviço deve ser configurado com os *clusters* da SHARCNET. Essa configuração é feita através do arquivo *config.ini* retratado no Código 4.4.

```
1  [servers]
2  bala=bala.sharcnet.ca
3  bruce=bruce.sharcnet.ca
4  bull=bull.sharcnet.ca
5  #dolphin=dolphin.sharcnet.ca
6  megaladon=megaladon.sharcnet.ca
7  narwhal=narwhal.sharcnet.ca
8  requin=requin.sharcnet.ca
9  tiger=tiger.sharcnet.ca
10 whale=whale.sharcnet.ca
11 zebra=zebra.sharcnet.ca
12 saw=saw.sharcnet.ca
```

Código 4.4: Configurações de servidores no arquivo `config.ini`.

Para executar esse serviço, o usuário deve invocá-lo na composição BPEL de forma assíncrona. Essa operação é composta de três etapas. Primeiramente o usuário deve invocar o `Mediador` através da função `INVOKE` do BPEL usando a interface WSDL demonstrada, de forma simplificada e apenas com seus componentes mais importantes, no Código 4.5. Nas linhas de 3 a 7 vemos a construção da mensagem dividida em três partes: O `MessageID`, parte da descrição WS-Addressing que define um código único para a mensagem; o `ReplyTo`, parte da descrição WS-Addressing que define o endereço da interface

de *callback* que deve receber a resposta e; o `parameters`, outros parâmetros necessários para executar o serviço. As linhas de 13 a 17 informam que o `MessageID` e o `ReplyTo` fazem parte co cabeçalho e o `parameters` faz parte do corpo da mensagem.

```
1  <definitions name="AsyncSharcnetImplService">
2
3    <message name="execSharcnet">
4      <part name="MessageID" element="wsa:MessageID" />
5      <part name="ReplyTo" element="wsa:ReplyTo" />
6      <part name="parameters" element="tns:execSharcnet"></part>
7    </message>
8
9    <binding name="AsyncSharcnetImplPortBinding" type="tns:AsyncSharcnetImpl"
       >
10     <wsaw:UsingAddressing></wsaw:UsingAddressing>
11     <operation name="execSharcnet">
12       <soap:operation soapAction=""></soap:operation>
13       <input>
14         <soap:header message="tns:execSharcnet" part="ReplyTo" use="
             literal"/>
15         <soap:header message="tns:execSharcnet" part="MessageID" use="
             literal"/>
16         <soap:body use="literal" parts="parameters"/>
17       </input>
18     </operation>
19   </binding>
20
21 </definitions>
```

Código 4.5: Componentes principais da interface WSDL para invocar o `Mediador`.

O Código 4.6 retrata o esquema XML que detalha os tipos de dados do elemento `execSharcnet`, tipo do atributo `parameters`. Os seguintes elementos são identificados:

- **server**: Parâmetro que define o *cluster* que deve executar a tarefa. Geralmente é utilizado o valor `AUTO`, que informa que o `Escalonador Global` deve automaticamente escolher o melhor *cluster*. Caso outro valor seja informado o `Escalonador Global` não é executado e o *cluster* especificado é executado diretamente;
- **login**: Para executar qualquer função na SHARCNET é necessário possuir uma conta. Esse parâmetro é usado para informar o nome usuário do cliente;
- **senha**: Senha do cliente de acesso à SHARCNET;
- **servico**: Informa qual serviço deve ser executado na SHARCNET.

```
1  <xs:schema>
2
3    <xs:element name="execSharcnet" type="tns:execSharcnet"></xs:element>
4
5    <xs:complexType name="execSharcnet">
6      <xs:sequence>
7        <xs:element name="server" type="string"></xs:element>
8        <xs:element name="login" type="string"></xs:element>
9        <xs:element name="senha" type="string"></xs:element>
10       <xs:element name="servico" type="string"></xs:element>
11     </xs:sequence>
12   </xs:complexType>
13
14 </xs:schema>
```

Código 4.6: Componentes principais do esquema XML do elemento `execSharcnet`.

A segunda etapa para a execução do serviço de forma assíncrona consiste em "implementar" a interface de *callback* para receber a resposta do serviço. Isso é feito através da função `RECEIVE` do BPEL. Essa interface é retratada no Código 4.7. Nas linhas de 3 a 6 vemos a construção da mensagem dividida em duas partes: O `RelatesTo`, parte da descrição WS-Addressing que especifica qual mensagem originou a resposta e; o `parameters`, outros parâmetros necessários para executar o serviço. As linhas de 12 a 15 informam que o `RelatesTo` faz parte co cabeçalho e o `parameters` faz parte do corpo da mensagem.

O Código 4.8 retrata o esquema XML que detalha os tipos de dados do elemento `response`, tipo do atributo `parameters`. Podemos identificar o elemento `result`, que é preenchido com a resposta do serviço executado na SHARCNET.

Na terceira e última etapa, o usuário deve criar a correlação da mensagem de `INVOKE` com a de `RECEIVE`. Na `Máquina de Execução` podem existir diversos processos executando ao mesmo tempo. Por isso, ela precisa saber a que instância deve enviar uma resposta recebida na interface de *callback*. Portanto, o usuário deve criar uma `Correlation` ligando o parâmetro `MessageID` da mensagem de `INVOKE` com o parâmetro `RelatesTo` da mensagem de `RECEIVE`. Essa ação está demonstrada na Figura 4.4.

```
<definitions name="AsyncSharcnetImplService">

    <message name="response">
        <part name="RelatesTo" element="wsa:RelatesTo" />
        <part name="parameters" element="tns:response"/>
    </message>

    <binding name="AsyncSharcnetResponseImplPortBinding" type="
        tns:AsyncSharcnetResponseImpl">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
            style="document"/>
        <operation name="response">
            <soap:operation soapAction=""/>
            <input>
                <soap:header message="tns:response" part="RelatesTo" use="
                    literal"/>
                <soap:body use="literal" parts="parameters" />
            </input>
        </operation>
    </binding>

</definitions>
```

Código 4.7: Componentes principais da interface WSDL de *callback* do `Mediador`.

```
<xs:schema>
  <xs:element name="response" type="tns:response"/>
  <xs:complexType name="response">
    <xs:sequence>
      <xs:element name="result" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Código 4.8: Componentes principais do esquema XML do elemento `response`.

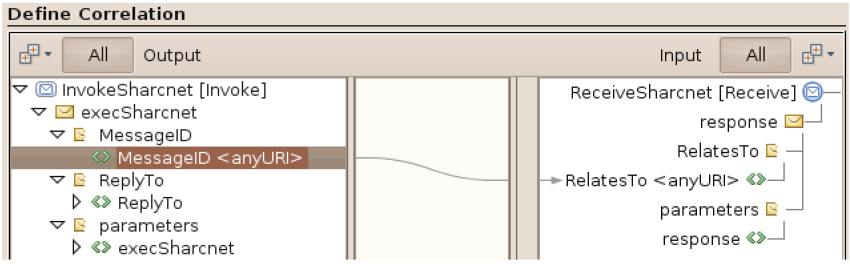

Figura 4.4: Estrutura de `Correlation` sendo criada.

Escalonador Global

O `Escalonador Global`, ou Meta-Escalonador, é a parte da estrutura responsável por verificar o estado dos *clusters* da SHARCNET e escolher o mais apropriado para enviar a tarefa. Como ele foi planejado para ser implantado em um sistema com aplicações em execução, manter a compatibilidade com os métodos antigos de seleção de recursos se torna fundamental. Entretanto, quando uma tarefa é submetida diretamente a algum *cluster*, o novo escalonador não tem conhecimento.

Esse cenário se assemelha ao funcionamento de pagamentos de contas em um banco, ilustrado na Figura 4.5. Por exemplo, um cliente pode saber em que agência (representando um *cluster*) ele pode pagar a conta, quantas pessoas estão em cada fila (programas esperando para serem executados) e quantos funcionários estão atendendo (número de processadores disponíveis). No entanto, ele não pode saber quanto tempo cada pessoa vai gastar no atendimento ou quantas pessoas vão estar na fila em algum momento no futuro.

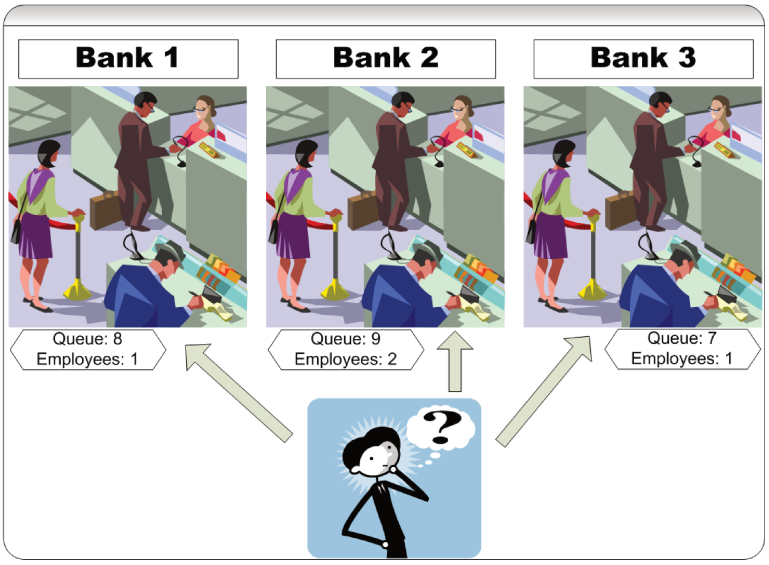

Figura 4.5: Cenário de pagamento de contas em um banco.

O usuário envia os requisitos de sua tarefa em termos de memória, espaço em disco e quantidade de processadores. Em seguida, o escalonador global seleciona um *cluster* para executar a tarefa; o *cluster* escolhido é o que tem a menor fila de espera entre os que atendem os requisitos do usuário. Para comparar a carga de cada HPCC propomos a seguinte métrica, definindo o tamanho da fila pela seguinte Fórmula 4.1.

$$\frac{Quantidade\ de\ processadores\ requisitados\ pela\ fila}{Quantidade\ total\ de\ processadores * Potencia\ dos\ processadores} \tag{4.1}$$

Nessa fórmula, a quantidade total de processadores disponíveis no *cluster* é multiplicada pelo seu poder de processamento, representando a capacidade total de um *cluster*. Então, o total de processadores requisitados pelos processos na fila é dividido por esse valor. O número resultante representa a carga atual do *cluster*.

Esse conceito foi implementado em uma aplicação desenvolvida na linguagem Python que se conecta a cada um dos *clusters* cadastrados e busca o estado corrente do escalonador LSF local. Utilizando os dados de cada um deles, o sistema seleciona o com a menor carga que atende aos requisitos do usuário e envia a tarefa, que agora passa a ser responsabilidade do escalonador local. Para evitar sobrecarregar a rede com buscas desnecessárias, o software só verifica o estado dos *clusters* quando existem tarefas para serem enviadas, além da possibilidade de configurar um tempo para armazenar as informações em *cache*.

O Código 4.9 mostra um trecho do arquivo de configuração `servers.conf`, utilizado para armazenar as informações dos *clusters* e configurar o tempo de *cache*. A marcação `DEFAULT` guarda a informação de quando foi a última a atualização e o tempo que programa deve aguardar para se atualizar novamente. A marcação `requin` guarda as informações obtidas sobre o *cluster* denominado `requin`, como endereço do servidor, quantidade de processadores, processos na fila, e outros.

```
1  [DEFAULT]
2  last_upate = 1267481866.13
3  update_time = 1800
4
5  [requin]
6  status = busy
7  cpu_total = 1368
8  name = requin
9  cpu_idle = 237
10 cpu_queued = 267
11 jobs_queued = 15
12 jobs_runing = 38
13 host = requin.sharcnet.ca
14 cpu_busy = 1131
15 features = pD
```

Código 4.9: Trecho do arquivo de configuração `servers.conf`.

Esse método se mostra vantajoso de diversas maneiras. Primeiramente, é simples e distribuído, permitindo que cada usuário execute o escalonador localmente, vendo a situação dos *clusters*, e envie as tarefas ao mais apropriado de forma transparente. Em segundo lugar, a abordagem não afeta a execução aplicações já existentes. Finalmente, a única carga adicional é calcular a formula mostrada anteriormente. A estrutura geral do `Escalonador Global` é descrita na Figura 4.6.

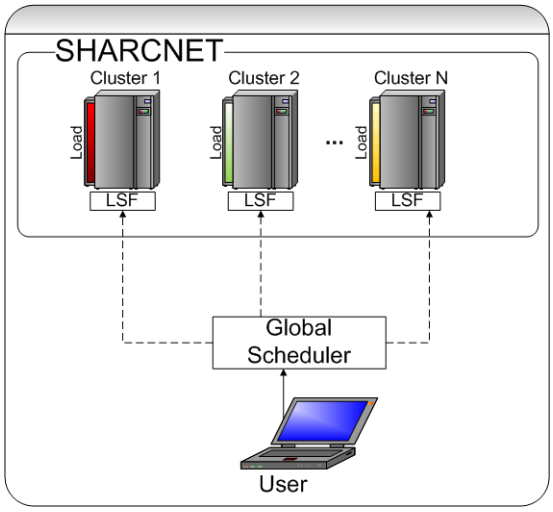

Figura 4.6: Arquitetura do Escalonador Global.

Monitor

Aplicação desenvolvida na linguagem Python responsável por monitorar o estado dos serviços sendo executados na SHARCNET. Para cada tarefa que o `Mediador` envia para a SHARCNET, ele cria uma instância de `Monitor` para monitorá-la. O `Monitor` guarda o número de identificação da tarefa e utiliza funções do escalonador LSF local para verificar seu estado e aguardar a sua finalização. Quando a tarefa conclui sua execução, seu resultado é lido e enviado para o endereço de *callback* especificado no WS-Addressing.

Provedor de CoC

O `Provedor de Cluster de Clusters` nada mais é do que a rede SHARCNET explicada na Sessão 4.3.1. Responsável por executar os serviços que necessitam de grande quantidade de poder computacional.

Provedor Comum

O `Provedor Comum` representa todos os provedores de serviços Web convencionais.

Para demosntrar o funcionamento geral do sistema, as Figuras 4.7 e 4.8 exibem, respectivamente, diagrama BPEL necessário para a execução e a sequencia de ações tomadas para executar serviços que utilizam a SHARCNET.

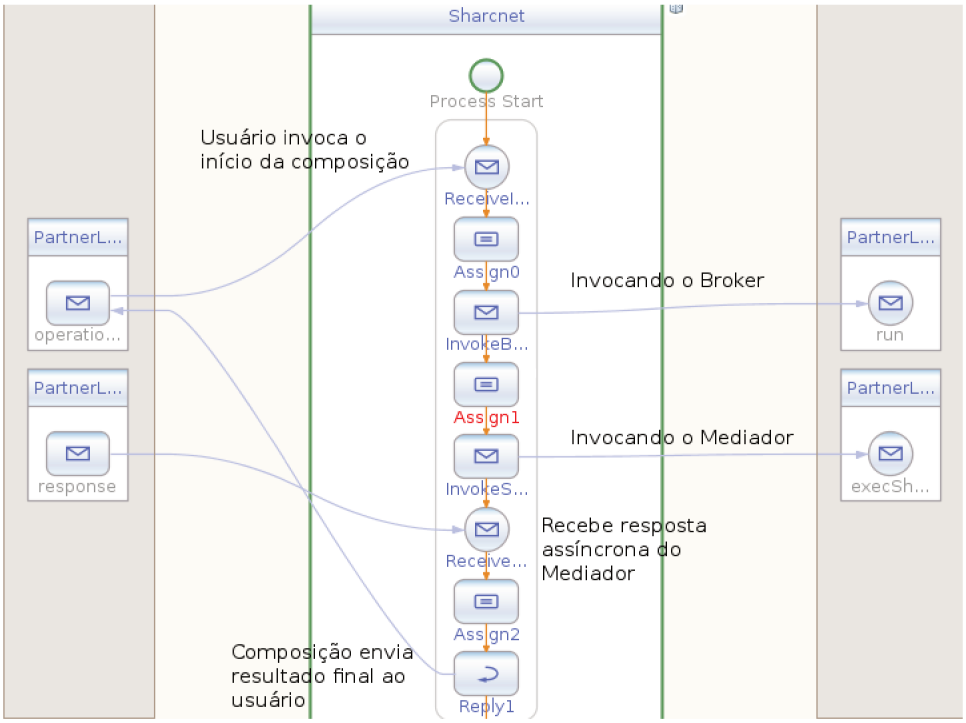

Figura 4.7: Diagrama BPEL mínimo para executar um serviço na SHARCNET.

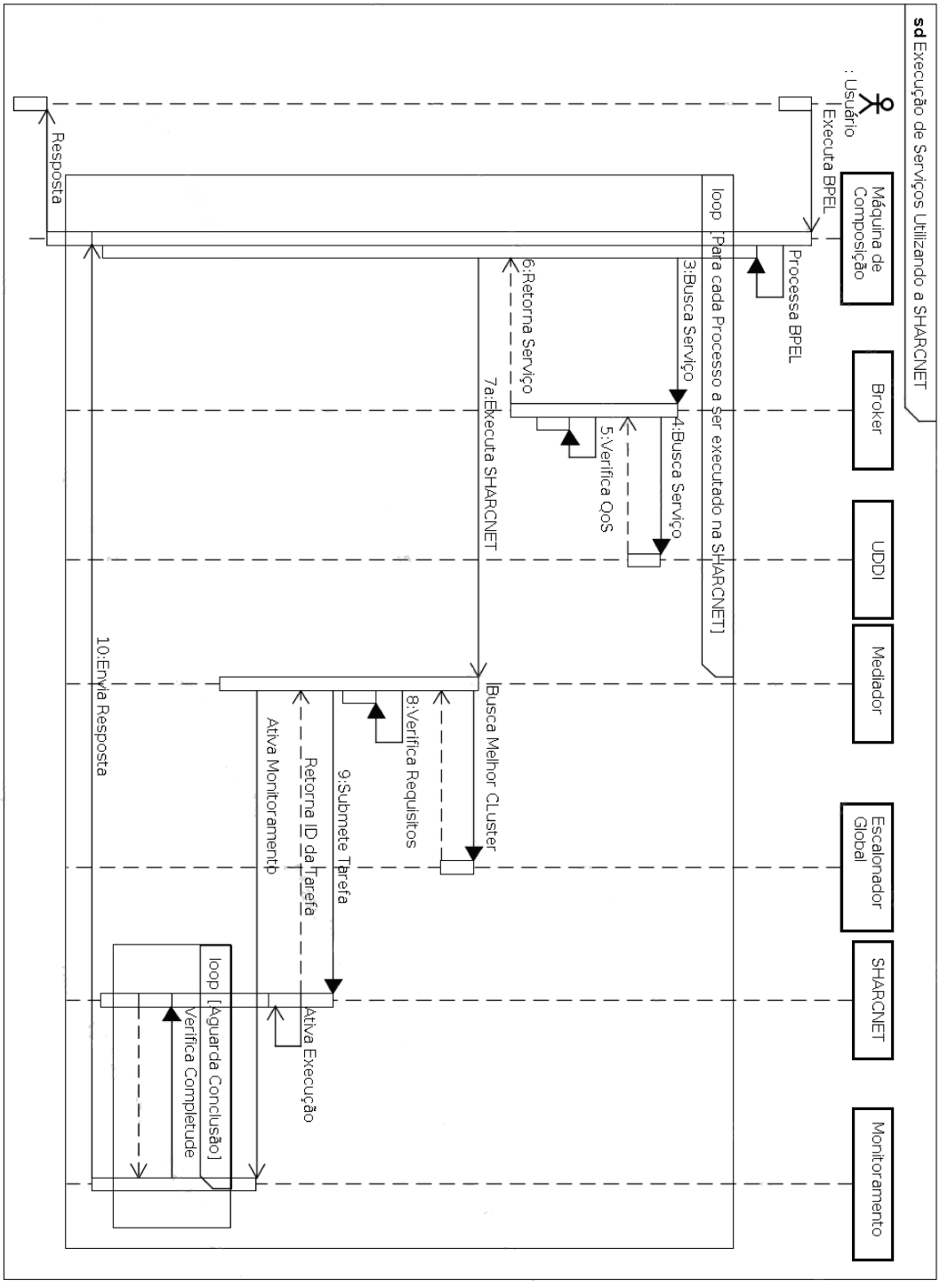

Figura 4.8: Diagrama de sequência de uma execução de serviços utilizando a SHARCNET. Os números na imagem são referentes aos números na Figura 4.2.

Capítulo 5

Aplicação

Para validar este trabalho e testar seu comportamento em um ambiente de produção, o mesmo foi aplicado em um problema real. Foi utilizado o *workflow* de criação de mosaicos da aplicação Montage [35], que junta diversas imagens para criar mosaicos personalizados do céu. Foram criados serviços que executam cada uma das etapas do algoritmo. As interações desses serviços foram especificadas em WS-BPEL e as partes mais custosas do algoritmo foram configuradas para serem executadas na rede SHARCNET.

A Sessão 5.1 explica com detalhes a aplicação Montage e seu *workflow* para a criação de mosaicos. A Sessão 5.2 demonstra os detalhes da implementação, como o processo BPEL, serviços e políticas. Por fim, a Sessão 5.3 discute as principais características, vantagens e desvantagens encontradas.

5.1 Montage

O Montage [35] foi criado pela NASA como um conjunto de ferramentas de código aberto que pode ser usado para gerar mosaicos personalizados do céu através de imagens de entrada no formato FITS (Sistema de Transporte Imagem Flexível) [1]. O algoritmo pode ser representado por um *workflow* e é algumas vezes executado em ambientes de grade, como o TeraGrid [36]. Durante a produção do mosaico, a geometria do produto final é calculada a partir da geometria das imagens de entrada. As entradas são então reprojetadas para ficarem na mesma escala espacial e rotação. As emissões de fundo das imagens são corrigidas para ficarem do mesmo nível em todas as imagens. As imagens reprojetadas e corrigidas são então unidas para formarem o mosaico final.

Neste trabalho foi utilizado um exemplo de criação de um mosaico a partir de 10 imagens do Atlas 2MASS da galáxia Pinwheel, ou M101. O *workflow* é dividido em cinco

[1] Do inglês, *Flexible Image Transport System*

etapas sequenciais e bem definidas: Reprojetar imagens, gerar mosaico inicial, modelar planos de fundo, combinar planos de fundo e gerar mosaico corrigido.

A primeira etapa é a mais custosa de todo o *workflow* e utiliza as ferramentas `mImgtbl` e `mProjExec` para, respectivamente, criar uma tabela de metadados descrevendo as imagens e reprojetá-las. Na segunda etapa as ferramentas `mAdd` e `mJPEG` são utilizadas para gerar o mosaico inicial, visto na Figura 5.1. Em seguida, são utilizadas as ferramentas `mOverlaps`, `mDiffExec` e `mFitExec` para suavizar os níveis de fundo entre as imagens sobrepostas. Após, os ajustes de plano de fundo são combinados e aplicados na imagem original reprojetada usando as ferramentas `mBgModel` e `mBgExec`. Por fim, o mosaico corrigido é gerado. Para isso as ferramentas `mAdd` e `mJPEG` são utilizadas novamente. A Figura 5.2 mostra o mosaico final corrigido.

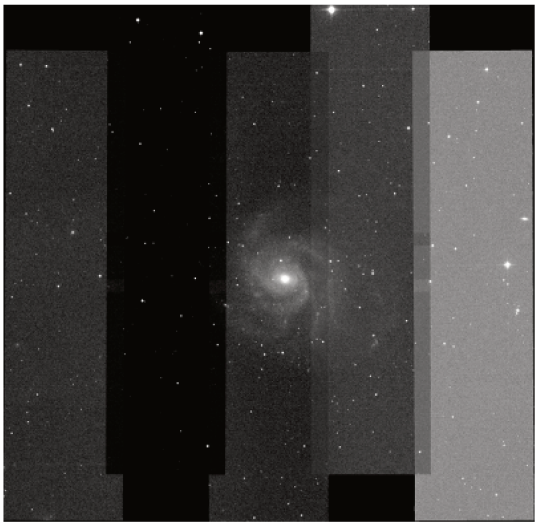

Figura 5.1: Mosaico M101 não corrigido.

Figura 5.2: Mosaico M101 corrigido.

5.2 Implementação

Para implementar e executar o *workflow* Montage foram criados cinco serviços Web, cada um executando uma das etapas do algoritmo mencionadas anteriormente, e um serviço extra responsável por enviar e receber dados da SHARCNET. Uma composição BPEL foi desenvolvida conectando os serviços de forma sequencial e configurando a primeira das etapas, a mais custosa, para ser executada na SHARCNET.

A Figura 5.3 demonstra o *workflow* final. Seu código BPEL e resposta final gerada são listados no Apêndice B. A SHARCNET é configurada inicialmente com as 10 imagens do Atlas 2MASS da galáxia Pinwheel. Em seguida, cada uma das seguintes etapas é executada:

1. O serviço que contém o *workflow* recebe a chamada de inicio e começa a execução.
2. Os parâmetros para a chamada ao UDDI estendido são configurados.
3. O UDDI estendido é consultado e, nesse caso, deverá retornar o URI para a SHARCNET.
4. Utilizando os parâmetros retornados pelo UDDI estendido, a chamada para a execução na SHARCNET é configurada.
5. A primeira etapa do algoritmo Montage é executada na SHARCNET.
6. A SHARCNET envia a resposta de forma assíncrona.
7. Os parâmetros para o restante do WF são configurados.
8. O serviço para executar a transferência de arquivos, obtendo os dados gerados na SHARCNET, é invocado.
9. O serviço com a segunda etapa do Montage é invocado.
10. O serviço com a terceira etapa do Montage é invocado.
11. O serviço com a quarta etapa do Montage é invocado.
12. O serviço com a quinta etapa do Montage é invocado.
13. Os parâmetros de retorno são preparados.
14. Resposta final é retornada.

Para invocar o BPEL foi utilizado o SOAPUI [17], uma ferramenta de código aberto para a execução de testes em serviços Web. Os Códigos B.4, B.5 e B.6, no Apêndice B, mostram *logs* de execução deste exemplo em horários distintos, com o objetivo de testar a escolha de *clusters* com cargas variadas.

5.3 Discussão

Alguns fatores importantes devem ser destacados. Primeiramente, foi possível resolver o problema através de um *wokflow* escrito em BPEL e executando parte do mesmo na SHARCNET. Dessa forma, é possível gerenciar e modificar o processo de uma forma simples, aproveitando as vantagens dos *workflows* e, ao mesmo tempo, utilizar o grande poder

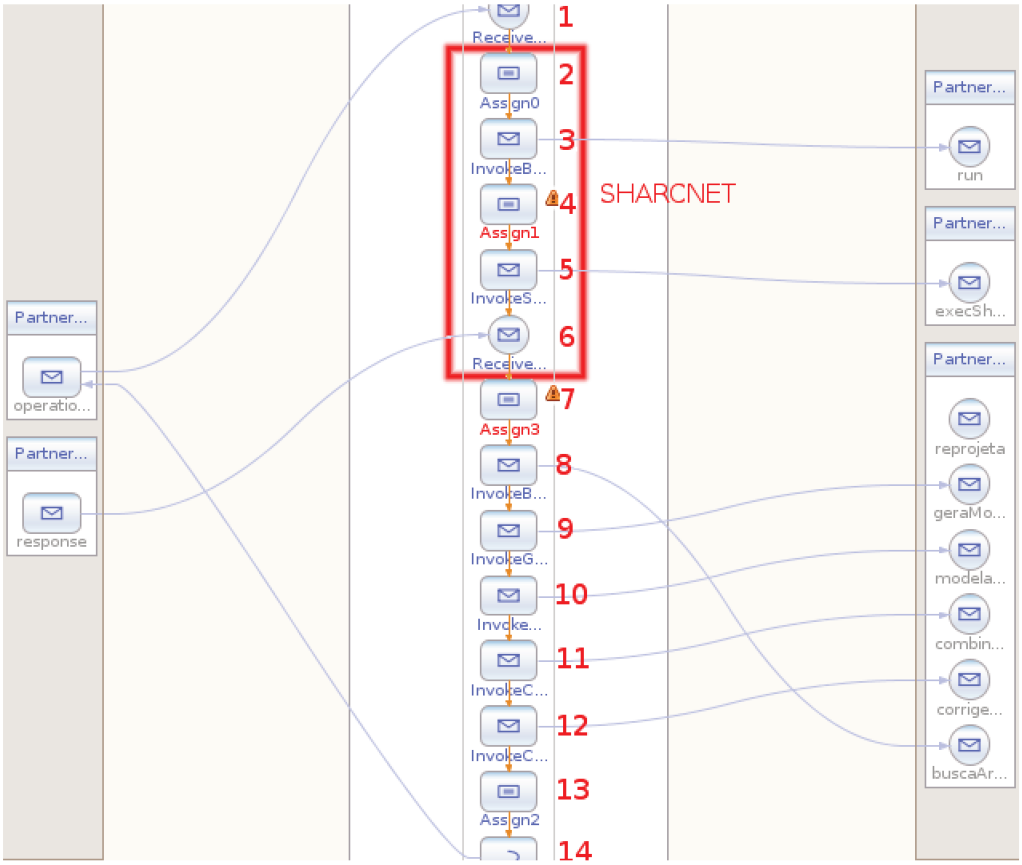

Figura 5.3: Imagem do BPEL para o *workflow* do Montage.

computacional de um *Multi-Cluster*. Também foi possível notar que utilizar o processamento da SHARCNET foi simples e a alocação de *clusters* foi feita de forma transparente, sempre atendendo aos requisitos do usuário e optando pelos mais disponíveis. Entretanto, a movimentação de dados ainda foi feita de forma manual.

Outro fator importante é a dificuldade de se monitorar a execução. Como o tipo de *workflow* utilizado possui serviços demorados, que podem levar de minutos a dias em execução, fica difícil saber qual estágio está sendo executado em um determinado momento. Este problema ainda é maximizado quando existem diversas instâncias de *workflow* sendo executadas ao mesmo tempo. Neste trabalho o monitoramento foi todo feito baseado nas saídas impressas pelos serviços. Entretanto, esse processo poderia ser facilitado com uma ferramenta específica para esse objetivo.

Capítulo 6

Conclusões

Esta dissertação descreve uma extensão à linguagem WS-BPEL para atender os requisitos de um *Multi-Cluster*. Essa extensão permite a especificação de atributos de QoS e de recursos necessários utilizando WS-Policy. Além do mais, este trabalho também propõe uma infraestrutura para a execução de *workflows* em *Multi-Clusters* e sua implementação correspondente na SHARCNET, uma rede multi-institucional de *clusters* de alta performance distribuídos em dezesseis instituições acadêmicas na província de Ontário, no Canadá. Em particular, o principal elemento dessa arquitetura é o *Mediador*, que atua como um *proxy* e contém um `Meta-Escalonador`, que seleciona de forma transparente os *clusters* de acordo com os requisitos dos usuários.

Diversos trabalhos abordam o problema de especificação e implementação de *workflows* em sistemas distribuídos mas alguns deles implementam novas linguagens para criar a composição, enquanto a tendência é a utilização de padrões já estabelecidos como WS-BPEL. Além do mais, não foram encontrados trabalhos que foquem em uma estrutura de *Multi-Cluster*, sendo a maioria deles baseados em grades computacionais.

A implementação integra uma máquina de execução WS-BPEL para executar *workflows* na rede SHARCNET através do uso de serviços Web assíncronos e WS-Addressing em conjunto com o UDDI estendido proposto por Garcia e Toledo [20]. Para cada tarefa de CoC a `Máquina de Execução` requisita ao `Escalonador Global` uma seleção de *cluster*. Em seguida, a tarefa é submetida e executada no *cluster* escolhido.

A solução se mostrou viável. Contudo, trabalhar com BPEL é um processo muitas vezes lento e utilizar funcionalidades que não são comuns é complexo e dependente da máquina de execução que está sendo utilizada. Foi possível executar processos utilizando recursos de um CoC, mas como esse tipo de processo pode demorar de minutos a dias em execução, fica difícil saber qual estágio está sendo executado em um determinado momento.

6.1 Contribuições

As principais contribuições incluem:

- Uma extensão à linguagem WS-BPEL para a especificação de *workflows* que podem utilizar recursos de um ambiente Multi-Cluster usando padrões consolidados, como o WS-Policy, permitindo a seleção de serviços baseada em requisitos não-funcionais;
- Uma proposta de arquitetura para a execução de *workflows* em um *Multi-Cluster*;
- Um estudo de caso aplicando essa arquitetura na rede SHARCNET;
- Uma implementação de escalonador global para a SHARCNET (Meta-Escalonador);
- Especificação e implementação de um *workflow* científico real, para gerar mosaicos personalizados do céu, utilizando o ambiente criado.
- Poster e artigo resumido com o título: "*A global scheduler for SHARCNET*" no *SHARCNET Research Day 2009*; Waterloo, Ontário, Canadá [27].
- Artigo completo com o título: "*An Infrastructure for Executing WS-BPEL Workflows in a Cluster of Clusters*" no *IEEE symposium on Computers and Communications (ISCC 2010)*; Riccione, Itália [28].

6.2 Trabalhos Futuros

Trabalhos futuros incluem o gerenciamento de dados, como armazenamento distribuido e considerações custo para a transferência dos mesmos entre nós; Estudos contemplando tolerância a falhas; Considerações de reserva, co-alocação e realocação de recursos; Consideração de ambientes móveis para a solicitação de serviços; E a criação de uma ferramenta para auxiliar no monitoramento do estado de cada *workflow* sendo executado.

Apêndice A

Interfaces para a utilização do Mediador

A.1 Interface WSDL para acesso ao Mediador e Callback

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <definitions name="AsyncSharcnetImplService" targetNamespace="http://ewe.
       org/"
4  xmlns:tns="http://ewe.org/" xmlns="http://schemas.xmlsoap.org/wsdl/"
5  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" xmlns:wsdl="
       http://schemas.xmlsoap.org/wsdl/"
6  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:xsd=
       "http://www.w3.org/2001/XMLSchema"
7  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
8  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
       wssecurity-utility-1.0.xsd"
9  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl" xmlns:plnk1="http://
       docs.oasis-open.org/wsbpel/2.0/plnktype">
10
11     <ns1:Policy xmlns:ns1="http://www.w3.org/ns/ws-policy" wsu:Id="
           AsyncSharcnetImplPortBinding_Addressing_Policy">
12         <ns1:ExactlyOne>
13             <ns1:All>
14                 <wsaw:UsingAddressing ns1:Optional="true"></
                       wsaw:UsingAddressing>
15             </ns1:All>
16         </ns1:ExactlyOne>
17     </ns1:Policy>
```

```
18
19      <ns2:Policy xmlns:ns2="http://www.w3.org/ns/ws-policy" wsu:Id="
            AsyncSharcnetImplPortBinding_execSharcnet_WSAT_Policy">
20          <ns2:ExactlyOne>
21              <ns2:All>
22                  <ns3:ATAlwaysCapability xmlns:ns3="http://schemas.xmlsoap.
                        org/ws/2004/10/wsat"></ns3:ATAlwaysCapability>
23                  <ns4:ATAssertion xmlns:ns5="http://schemas.xmlsoap.org/ws
                        /2002/12/policy" xmlns:ns4="http://schemas.xmlsoap.org/
                        ws/2004/10/wsat" ns2:Optional="true" ns5:Optional="true"
                        ></ns4:ATAssertion>
24              </ns2:All>
25          </ns2:ExactlyOne>
26      </ns2:Policy>
27
28      <types>
29          <xsd:schema>
30              <xsd:import namespace="http://schemas.xmlsoap.org/ws/2004/08/
                    addressing" schemaLocation="wsaddressing.xsd" />
31          </xsd:schema>
32
33          <xsd:schema>
34              <xsd:import namespace="http://ewe.org/" schemaLocation="
                    AsyncSharcnetJunto.xsd"/>
35          </xsd:schema>
36      </types>
37
38      <message name="execSharcnet">
39          <part name="MessageID" element="wsa:MessageID" />
40          <part name="ReplyTo" element="wsa:ReplyTo" />
41          <part name="parameters" element="tns:execSharcnet"></part>
42      </message>
43
44      <message name="response">
45          <part name="RelatesTo" element="wsa:RelatesTo" />
46          <part name="parameters" element="tns:response"/>
47      </message>
48
49      <portType name="AsyncSharcnetImpl">
50          <operation name="execSharcnet">
51              <input wsaw:Action="http://ewe.org/AsyncSharcnetImpl/
                    execSharcnet" message="tns:execSharcnet"></input>
52          </operation>
53      </portType>
54
55      <portType name="AsyncSharcnetResponseImpl">
```

```
56          <operation name="response">
57              <input message="tns:response"/>
58          </operation>
59      </portType>
60
61      <binding name="AsyncSharcnetImplPortBinding" type="
            tns:AsyncSharcnetImpl">
62          <wsaw:UsingAddressing></wsaw:UsingAddressing>
63          <ns6:PolicyReference xmlns:ns6="http://www.w3.org/ns/ws-policy" URI
                ="#AsyncSharcnetImplPortBinding_Addressing_Policy"></
                ns6:PolicyReference>
64          <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                style="document"></soap:binding>
65
66          <operation name="execSharcnet">
67              <ns7:PolicyReference xmlns:ns7="http://www.w3.org/ns/ws-policy"
                    URI="#AsyncSharcnetImplPortBinding_execSharcnet_WSAT_Policy
                    "></ns7:PolicyReference>
68              <soap:operation soapAction=""></soap:operation>
69              <input>
70                  <soap:header message="tns:execSharcnet" part="ReplyTo" use=
                        "literal" encodingStyle="" />
71                  <soap:header message="tns:execSharcnet" part="MessageID"
                        use="literal" encodingStyle="" />
72                  <soap:body use="literal" parts="parameters"/>
73              </input>
74          </operation>
75      </binding>
76
77      <binding name="AsyncSharcnetResponseImplPortBinding" type="
            tns:AsyncSharcnetResponseImpl">
78          <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                style="document"/>
79          <operation name="response">
80              <soap:operation soapAction=""/>
81              <input>
82                  <soap:header message="tns:response" part="RelatesTo" use="
                        literal" encodingStyle="" />
83                  <soap:body use="literal" parts="parameters" />
84              </input>
85          </operation>
86      </binding>
87
88      <service name="AsyncSharcnetImplService">
89          <port name="AsyncSharcnetImplPort" binding="
                tns:AsyncSharcnetImplPortBinding">
```

```
90          <!--<soap:address location="http://127.0.0.1:8080/
                AsyncSharcnetImplService/AsyncSharcnetImpl"></soap:address>-->
91              <soap:address location=""></soap:address>
92          </port>
93      </service>
94
95      <service name="AsyncSharcnetResponseImplService">
96          <port name="AsyncSharcnetResponseImplPort" binding="
                tns:AsyncSharcnetResponseImplPortBinding">
97              <soap:address location="http://localhost:18181/
                    AsynchronousClient/response" />
98          </port>
99      </service>
100
101     <plnk1:partnerLinkType name="AsyncSharcnetImpl">
102         <plnk1:role name="role1" portType="tns:AsyncSharcnetImpl" />
103     </plnk1:partnerLinkType>
104
105     <plnk1:partnerLinkType name="AsyncSharcnetResponseImpl">
106         <plnk1:role name="role1" portType="tns:AsyncSharcnetResponseImpl" />
107     </plnk1:partnerLinkType>
108
109 </definitions>
```

Código A.1: Interface WSDL para acesso ao `Mediador`.

A.2 Interface XSD para acesso ao Mediador e Callback

```
1  <?xml version="1.0" encoding="UTF-8"?><!-- Published by JAX-WS RI at http:
       //jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.1-hudson-749-
       SNAPSHOT. -->
2
3  <xs:schema xmlns:tns="http://ewe.org/" xmlns:xs="http://www.w3.org/2001/
       XMLSchema" version="1.0" targetNamespace="http://ewe.org/">
4
5  <xs:element name="execSharcnet" type="tns:execSharcnet"></xs:element>
6
7  <xs:complexType name="execSharcnet">
8  <xs:sequence>
9  <xs:element name="server" type="xs:string" minOccurs="0"></xs:element>
10 <xs:element name="login" type="xs:string"></xs:element>
11 <xs:element name="senha" type="xs:string"></xs:element>
12 <xs:element name="comando" type="xs:string"></xs:element>
13 <xs:element name="arqOutput" type="xs:string"></xs:element>
14 <xs:element name="requisitos" type="xs:string" minOccurs="0"></xs:element>
15 </xs:sequence>
16 </xs:complexType>
17
18   <xs:element name="response" type="tns:response"/>
19
20   <xs:complexType name="response">
21     <xs:sequence>
22       <xs:element name="result" type="xs:string"/>
23     </xs:sequence>
24   </xs:complexType>
25
26 </xs:schema>
```

Código A.2: Interface XSD para acesso ao `Mediador`.

Apêndice B

Fontes Completos do Exemplo Montage

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <process
3      name="Sharcnet"
4      targetNamespace="http://enterprise.netbeans.org/bpel/
           ExemploAsynchronous/Sharcnet"
5      xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
6      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7      xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/
           SUNExtension/Trace"
8      xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/
           SUNExtension/Editor"
9      xmlns:sxat="http://www.sun.com/wsbpel/2.0/process/executable/
           SUNExtension/Attachment"
10     xmlns:sxeh="http://www.sun.com/wsbpel/2.0/process/executable/
           SUNExtension/ErrorHandling"
11     xmlns:tns="http://enterprise.netbeans.org/bpel/ExemploAsynchronous/
           Sharcnet" xmlns:sxed2="http://www.sun.com/wsbpel/2.0/process/
           executable/SUNExtension/Editor2" xmlns:ns0="http://enterprise.
           netbeans.org/bpel/WizardCorrelationProperties" xmlns:sxnmp="http://
           www.sun.com/wsbpel/2.0/process/executable/SUNExtension/NMProperty"
           xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/08/addressing"
           xmlns:ns2="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
            xmlns:ns3="http://www.w3.org/2005/08/addressing" xmlns:sxxf="http:
           //www.sun.com/wsbpel/2.0/process/executable/SUNExtension/
           XPathFunctions">
12
13     <import namespace="http://enterprise.netbeans.org/bpel/
           AsynchronousClient" location="AsynchronousClient.wsdl" importType="
           http://schemas.xmlsoap.org/wsdl/" />
```

```
14      <import namespace="http://ewe.org/" location="127.0.0.1_8080/
            AsyncSharcnetJunto.wsdl" importType="http://schemas.xmlsoap.org/wsdl
            /" />
15      <import namespace="http://enterprise.netbeans.org/bpel/
            WizardCorrelationProperties" location="WizardCorrelationProperties.
            wsdl" importType="http://schemas.xmlsoap.org/wsdl/" />
16      <import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
            location="127.0.0.1_8080/wsaddressing.xsd" importType="http://www.w3
            .org/2001/XMLSchema" />
17      <import namespace="http://fbd.org/" location="127.0.0.1_8080/
            FakeBrokerService/FakeBroker.wsdl" importType="http://schemas.
            xmlsoap.org/wsdl/" />
18      <import namespace="http://amt.org/" location="127.0.0.1_8080/
            AplicacaoMontageService/AplicacaoMontage.wsdl" importType="http://
            schemas.xmlsoap.org/wsdl/" />
19
20      <partnerLinks>
21          <partnerLink name="PartnerLinkBroker" xmlns:tns="http://fbd.org/"
                partnerLinkType="tns:FakeBroker" partnerRole="role1" />
22          <partnerLink name="PartnerLinkSharcnetExec" xmlns:tns="http://ewe.
                org/" partnerLinkType="tns:AsyncSharcnetImpl" partnerRole="role1
                " />
23          <partnerLink name="PartnerLinkMontage" xmlns:tns="http://amt.org/"
                partnerLinkType="tns:AplicacaoMontage" partnerRole="role1" />
24          <partnerLink name="PartnerLinkClient" xmlns:tns="http://enterprise.
                netbeans.org/bpel/AsynchronousClient" partnerLinkType="
                tns:AsynchronousClientPartnerLinkType" myRole="
                AsynchronousClientProvider" />
25          <partnerLink name="PartnerLinkSharcnetResponse" xmlns:tns="http://
                ewe.org/" partnerLinkType="tns:AsyncSharcnetResponseImpl" myRole
                ="role1" />
26      </partnerLinks>
27
28      <variables>
29          <variable name="CorrigeMosaicoIn" xmlns:tns="http://amt.org/"
                messageType="tns:corrigeMosaico" />
30          <variable name="CorrigeMosaicoOut" xmlns:tns="http://amt.org/"
                messageType="tns:corrigeMosaicoResponse" />
31          <variable name="CombinaFundoOut" xmlns:tns="http://amt.org/"
                messageType="tns:combinaFundoResponse" />
32          <variable name="CombinaFundoIn" xmlns:tns="http://amt.org/"
                messageType="tns:combinaFundo" />
33          <variable name="ModelaFundoOut" xmlns:tns="http://amt.org/"
                messageType="tns:modelaFundoResponse" />
34          <variable name="ModelaFundoIn" xmlns:tns="http://amt.org/"
                messageType="tns:modelaFundo" />
```

```
35          <variable name="GeraMosaicoOut" xmlns:tns="http://amt.org/"
                messageType="tns:geraMosaicoResponse"/>
36          <variable name="GeraMosaicoIn" xmlns:tns="http://amt.org/"
                messageType="tns:geraMosaico"/>
37          <variable name="BuscaArquivosOut" xmlns:tns="http://amt.org/"
                messageType="tns:buscaArquivosResponse"/>
38          <variable name="BuscaArquivosIn" xmlns:tns="http://amt.org/"
                messageType="tns:buscaArquivos"/>
39          <variable name="RunOut" xmlns:tns="http://fbd.org/" messageType="
                tns:runResponse"/>
40          <variable name="RunIn" xmlns:tns="http://fbd.org/" messageType="
                tns:run"/>
41          <variable name="ExecSharcnetIn" xmlns:tns="http://ewe.org/"
                messageType="tns:execSharcnet">
42              <sxed2:editor></sxed2:editor>
43          </variable>
44          <variable name="OperationAOut" xmlns:tns="http://enterprise.
                netbeans.org/bpel/AsynchronousClient" messageType="
                tns:responseMessageClient"/>
45          <variable name="ResponseIn" xmlns:tns="http://ewe.org/" messageType
                ="tns:response">
46              <sxed2:editor></sxed2:editor>
47          </variable>
48          <variable name="OperationAIn" xmlns:tns="http://enterprise.netbeans
                .org/bpel/AsynchronousClient" messageType="
                tns:requestMessageClient"/>
49      </variables>
50
51      <correlationSets>
52          <correlationSet name="wzrd_set_InvokeSharcnet_ReceiveSharcnet"
                properties="ns0:wzrd_prop_MessageID_RelatesTo"/>
53      </correlationSets>
54
55      <sequence>
56          <receive name="ReceiveInicio" createInstance="yes" partnerLink="
                PartnerLinkClient" operation="operationA" xmlns:tns="http://
                enterprise.netbeans.org/bpel/AsynchronousClient" portType="
                tns:MyPortTypeClient" variable="OperationAIn"></receive>
57
58  <assign name="Assign0">
59              <copy>
60                  <from>'uuid:00000'</from>
61                  <to>$RunIn.parameters/arg0</to>
62              </copy>
63              <copy>
64                  <from>'WSDL'</from>
```

```
65              <to>$RunIn.parameters/arg1</to>
66          </copy>
67      </assign>
68
69      <invoke name="InvokeBroker" partnerLink="PartnerLinkBroker"
            operation="run" xmlns:tns="http://fbd.org/" portType="
            tns:FakeBroker" inputVariable="RunIn" outputVariable="RunOut"/>
70
71      <assign name="Assign1">
72          <copy>
73              <from>'http://localhost:18181/AsynchronousClient/response'<
                    /from>
74              <to>$ExecSharcnetIn.ReplyTo/ns1:Address</to>
75          </copy>
76          <copy>
77              <from>$OperationAIn.inputType/id</from>
78              <to variable="ExecSharcnetIn" part="MessageID"/>
79          </copy>
80          <copy>
81              <from>'auto'</from>
82              <to>$ExecSharcnetIn.parameters/server</to>
83          </copy>
84          <copy>
85              <from>'Usuario'</from>
86              <to>$ExecSharcnetIn.parameters/login</to>
87          </copy>
88          <copy>
89              <from>'SeNha'</from>
90              <to>$ExecSharcnetIn.parameters/senha</to>
91          </copy>
92          <copy>
93              <from>$OperationAIn.inputType/paramA</from>
94              <to>$ExecSharcnetIn.parameters/comando</to>
95          </copy>
96          <copy>
97              <from>'/home/tlechuga/out.log'</from>
98              <to>$ExecSharcnetIn.parameters/arqOutput</to>
99          </copy>
100         <copy>
101             <from>'N'</from>
102             <to>$ExecSharcnetIn.parameters/requisitos</to>
103         </copy>
104         <copy>
105             <from>$RunOut.parameters/return</from>
106             <to variable="ExecSharcnetIn" sxnmp:nmProperty="org.
                    glassfish.openesb.outbound.address.url"/>
```

```
107             </copy>
108         </assign>
109
110         <invoke name="InvokeSharcnet" partnerLink="PartnerLinkSharcnetExec"
                 operation="execSharcnet" xmlns:tns="http://ewe.org/" portType="
                tns:AsyncSharcnetImpl" inputVariable="ExecSharcnetIn" PolicyURIs
                ="http://127.0.0.1:8080/AsyncSharcnetImplService/politica.xml">
111             <correlations>
112                 <correlation set="wzrd_set_InvokeSharcnet_ReceiveSharcnet"
                        initiate="yes"/>
113             </correlations>
114         </invoke>
115
116         <receive name="ReceiveSharcnet" createInstance="no" partnerLink="
                PartnerLinkSharcnetResponse" operation="response" xmlns:tns="
                http://ewe.org/" portType="tns:AsyncSharcnetResponseImpl"
                variable="ResponseIn">
117             <correlations>
118                 <correlation set="wzrd_set_InvokeSharcnet_ReceiveSharcnet"
                        initiate="no"/>
119             </correlations>
120         </receive>
121
122         <assign name="Assign3">
123             <copy>
124                 <from>''</from>
125                 <to variable="CorrigeMosaicoIn" part="parameters"/>
126             </copy>
127             <copy>
128                 <from>''</from>
129                 <to variable="CombinaFundoIn" part="parameters"/>
130             </copy>
131             <copy>
132                 <from>''</from>
133                 <to variable="ModelaFundoIn" part="parameters"/>
134             </copy>
135             <copy>
136                 <from>''</from>
137                 <to variable="GeraMosaicoIn" part="parameters"/>
138             </copy>
139             <copy>
140                 <from>''</from>
141                 <to variable="BuscaArquivosIn" part="parameters"/>
142             </copy>
143         </assign>
144
```

```
145         <invoke name="InvokeBuscaArquivos" partnerLink="PartnerLinkMontage"
                operation="buscaArquivos" xmlns:tns="http://amt.org/" portType=
                "tns:AplicacaoMontage" inputVariable="BuscaArquivosIn"
                outputVariable="BuscaArquivosOut"/>
146         <invoke name="InvokeGeraMosaico" partnerLink="PartnerLinkMontage"
                operation="geraMosaico" xmlns:tns="http://amt.org/" portType="
                tns:AplicacaoMontage" inputVariable="GeraMosaicoIn"
                outputVariable="GeraMosaicoOut"/>
147         <invoke name="InvokeModelaFundo" partnerLink="PartnerLinkMontage"
                operation="modelaFundo" xmlns:tns="http://amt.org/" portType="
                tns:AplicacaoMontage" inputVariable="ModelaFundoIn"
                outputVariable="ModelaFundoOut"/>
148         <invoke name="InvokeCombinaFundo" partnerLink="PartnerLinkMontage"
                operation="combinaFundo" xmlns:tns="http://amt.org/" portType="
                tns:AplicacaoMontage" inputVariable="CombinaFundoIn"
                outputVariable="CombinaFundoOut"/>
149         <invoke name="InvokeCorrigeMosaico" partnerLink="PartnerLinkMontage
                " operation="corrigeMosaico" xmlns:tns="http://amt.org/"
                portType="tns:AplicacaoMontage" outputVariable="
                CorrigeMosaicoOut" inputVariable="CorrigeMosaicoIn"/>
150         <assign name="Assign2">
151             <copy>
152                 <from>string($ResponseIn.RelatesTo)</from>
153                 <to>$OperationAOut.resultType/id</to>
154             </copy>
155             <copy>
156                 <from>concat('&#13;&#13;===RESULTADO CLUSTER====&#13;&#13;'
                        , $ResponseIn.parameters/result, '&#13;&#13;===
                        RESULTADOS NAO CLUSTER====&#13;&#13;', $BuscaArquivosOut
                        .parameters/return, $GeraMosaicoOut.parameters/return, $
                        ModelaFundoOut.parameters/return, $CombinaFundoOut.
                        parameters/return, $CorrigeMosaicoOut.parameters/return)
                        </from>
157                 <to>$OperationAOut.resultType/paramA</to>
158             </copy>
159         </assign>
160
161         <reply name="Reply1" partnerLink="PartnerLinkClient" operation="
                operationA" xmlns:tns="http://enterprise.netbeans.org/bpel/
                AsynchronousClient" portType="tns:MyPortTypeClient" variable="
                OperationAOut"/>
162     </sequence>
163
164 </process>
```

Código B.1: Código do processo BPEL para a execução do *workflow* Montage.

```
<wsp:Policy>
    <wsp:ExactlyOne>
      <wsp:All>
        <resp:CPU
          xmlns:qosp="...schema/respolicy"
          operation="InvokeSharcnet"
          specification="uddi:qos:attribute:
          CPUPower">high</resp:CPU>
      </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```

Código B.2: Política de recursos utilizada na aplicação Montage.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
    /">
   <SOAP-ENV:Body>

      <typeA xmlns:msgns="http://enterprise.netbeans.org/bpel/
          AsynchronousClient" xmlns="http://enterprise.netbeans.org/bpel/
          AsynchronousSchemaNamespace">

<paramA xmlns="">
<![CDATA[

===RESULTADO CLUSTER===

====Adicionando binarios no PATH====
====Iniciando execucao====

==Step Two: Reproject images==
[struct stat="OK", count=10, badfits=0]
[struct stat="OK", count=10, failed=0, nooverlap=0]
[struct stat="OK", count=10, badfits=0]
====Copiando arquivos para Home====

------------------------------------------------------------
Sender: LSF System <lsfadmin@lsfhost.localdomain>
Subject: Job 239172: <srun sh /home/tlechuga/Montage/executar1c.sh> Done

Job <srun sh /home/tlechuga/Montage/executar1c.sh> was submitted from host
   <saw377> by user <tlechuga>.
Job was executed on host(s) <lsfhost.localdomain>, in queue <serial>, as
   user <tlechuga>.
</home/tlechuga> was used as the home directory.
</home/tlechuga> was used as the working directory.
```

```
28 Started at Mon May 24 15:56:26 2010
29 Results reported at Mon May 24 15:57:40 2010
30
31 Your job looked like:
32
33 ------------------------------------------------------------
34 # LSBATCH: User input
35 srun sh /home/tlechuga/Montage/executar1c.sh
36 ------------------------------------------------------------
37
38 Successfully completed.
39
40 Resource usage summary:
41
42     CPU time   :      1.51 sec.
43     Max Memory :         2 MB
44     Max Swap   :       125 MB
45
46
47 The output (if any) is above this job summary.
48
49
50
51
52 ===RESULTADOS NAO CLUSTER===
53
54 [Transferindo Arquivos]:OK
55 [struct stat="OK", time=6]
56 [struct stat="OK", min=80.7457, minpercent=20.00, max=180.992, maxpercent
       =99.98]
57 [struct stat="OK", count=17]
58 [struct stat="OK", count=17, failed=0]
59 [struct stat="OK", count=17, failed=0, warning=0, missing=0]
60 [struct stat="OK"]
61 [struct stat="OK", count=10, nocorrection=0, failed=0]
62 [struct stat="OK", time=3]
63 [struct stat="OK", min=82.3305, minpercent=20.00, max=178.268, maxpercent
       =99.98]]]>
64 </paramA>
65
66           <id xmlns="">54321</id>
67       </typeA>
68    </SOAP-ENV:Body>
69 </SOAP-ENV:Envelope>
```

Código B.3: Resposta final retornada pelo *workflow* da aplicação Montage.

```
1  Execucao Iniciada:  14/06/2010 - 11:53:02
2
3  ====Broker Invocado.====
4  Retornando URI
5  URI: http://127.0.0.1:8080/AsyncSharcnetImplService/AsyncSharcnetImpl
6
7  ====Iniciando SHARCNET====
8  Executando comando: auto tlechuga senha N sqsub -o out.log -r 3m sh /home/
       tlechuga/Montage/executar1c.sh
9  Status NAO Atualizado
10 Atualizando informacoes dos Servidores
11 requin--->1468 CPUs total, 259 idle, 1209 busy; 58 jobs running; 18
       suspended, 907 queued.
12 requin requin.sharcnet.ca busy
13 bruce--->128 CPUs total, 38 idle, 90 busy; 53 jobs running; 0 suspended, 1
       queued.
14 bruce bruce.sharcnet.ca busy
15 megaladon--->124 CPUs total, 24 idle, 100 busy; 19 jobs running; 0
       suspended, 0 queued.
16 megaladon megaladon.sharcnet.ca idle
17 bala--->128 CPUs total, 96 idle, 32 busy; 32 jobs running; 0 suspended, 1
       queued.
18 bala bala.sharcnet.ca busy
19 tiger-->Erro ao contactar tiger
20 bull--->384 CPUs total, 22 idle, 362 busy; 165 jobs running; 0 suspended,
       39 queued.
21 bull bull.sharcnet.ca busy
22 whale--->2620 CPUs total, 594 idle, 2026 busy; 1715 jobs running; 0
       suspended, 35615 queued
23 whale whale.sharcnet.ca busy
24 saw--->2680 CPUs total, 445 idle, 2235 busy; 213 jobs running; 0 suspended,
        121 queued.
25 saw saw.sharcnet.ca busy
26 narwhal--->1064 CPUs total, 50 idle, 1014 busy; 101 jobs running; 0
       suspended, 187 queued.
27 narwhal narwhal.sharcnet.ca busy
28 *Requisitos preenchidos
29 *Executando comando em-> megaladon megaladon.sharcnet.ca idle
30 ====Preparando Resposta====
31 submitted as jobid 26729
32 megaladon.sharcnet.ca
33 ========
34 Verificando Callback
35 Achei WSAddress
36 MessageID: 54321
```

```
37 WS_Addressing: http://localhost:18181/AsynchronousClient/response
38 ========
39 Monitorando Job
40 Monitorando: bala.sharcnet.ca tlechuga 26729
41 Monitorando Job: 26729 Em: megaladon.sharcnet.ca
42 Job Concluido
43 ========
44 Lendo resultado
45 Preparando resposta
46 Resposta enviada
47 ====Sharcnet Concluiu====
48
49 ====Executando outros servicos====
50 Baixando Resultados
51 Gerando Mosaico
52 Modelando Background
53 Combinando Fundo
54 Corrigindo Mosaico
55 ====BPEL Concluiu====
56
57 Execucao Finalizada: 14/06/2010 - 11:59:13
```

Código B.4: *Log* de execução da aplicação Montage (1).

```
1  Execucao Iniciada: 14/06/2010 - 15:40:25
2
3  ====Broker Invocado.====
4  Retornando URI
5  URI: http://127.0.0.1:8080/AsyncSharcnetImplService/AsyncSharcnetImpl
6
7  ====Iniciando SHARCNET====
8  Executando comando: auto tlechuga senha N sqsub -o out.log -r 3m sh /home/
       tlechuga/Montage/executar1c.sh
9  Status NAO Atualizado
10 Atualizando informacoes dos Servidores
11 requin--->1462 CPUs total, 88 idle, 1374 busy; 66 jobs running; 4 suspended
       , 923 queued.
12 requin requin.sharcnet.ca busy
13 bruce--->128 CPUs total, 44 idle, 84 busy; 54 jobs running; 0 suspended, 1
       queued.
14 bruce bruce.sharcnet.ca busy
15 megaladon--->124 CPUs total, 90 idle, 34 busy; 3 jobs running; 0 suspended,
        0 queued.
16 megaladon megaladon.sharcnet.ca idle
17 bala--->128 CPUs total, 19 idle, 109 busy; 33 jobs running; 0 suspended, 0
       queued.
```

```
18 bala bala.sharcnet.ca idle
19 tiger——>Erro ao contactar tiger
20 bull———>384 CPUs total, 33 idle, 351 busy; 157 jobs running; 0 suspended,
       41 queued.
21 bull bull.sharcnet.ca busy
22 whale——>Erro ao contactar whale
23 saw———>2672 CPUs total, 767 idle, 1905 busy; 314 jobs running; 9 suspended,
        121 queued.
24 saw saw.sharcnet.ca busy
25 narwhal———>1064 CPUs total, 52 idle, 1012 busy; 101 jobs running; 0
       suspended, 219 queued.
26 narwhal narwhal.sharcnet.ca busy
27 *Requisitos preenchidos
28 *Executando comando em-> bala bala.sharcnet.ca idle
29 ====Preparando Resposta====
30 submitted as jobid 102272
31 bala.sharcnet.ca
32 ========
33 Verificando Callback
34 Achei WSAddress
35 MessageID: 54322
36 WS-Addressing: http://localhost:18181/AsynchronousClient/response
37 ========
38 Monitorando Job
39 Monitorando: bala.sharcnet.ca tlechuga 102272
40 Monitorando Job:  102272  Em:  bala.sharcnet.ca
41 Job Concluido
42 ========
43 Lendo resultado
44 Preparando resposta
45 Resposta enviada
46 ====Sharcnet Concluiu====
47
48 ====Executando outros servicos====
49 Baixando Resultados
50 Gerando Mosaico
51 Modelando Background
52 Combinando Fundo
53 Corrigindo Mosaico
54 ====BPEL Concluiu====
55
56 Execucao Finalizada:  14/06/2010 - 15:44:42
```

Código B.5: *Log* de execução da aplicação Montage (2).

```
1  Execucao Iniciada:  14/06/2010 - 22:57:02
2
3  ====Broker Invocado.====
4  Retornando URI
5  URI: http://127.0.0.1:8080/AsyncSharcnetImplService/AsyncSharcnetImpl
6
7  ====Iniciando SHARCNET====
8  Executando comando: auto tlechuga senha N sqsub -o out.log -r 3m sh /home/
       tlechuga/Montage/executar1c.sh
9  Status NAO Atualizado
10 Atualizando informacoes dos Servidores
11 requin--->1466 CPUs total, 89 idle, 1377 busy; 136 jobs running; 4
       suspended, 925 queued.
12 requin requin.sharcnet.ca busy
13 bruce--->128 CPUs total, 28 idle, 100 busy; 58 jobs running; 0 suspended, 1
        queued.
14 bruce bruce.sharcnet.ca busy
15 megaladon--->124 CPUs total, 24 idle, 100 busy; 16 jobs running; 0
       suspended, 0 queued.
16 megaladon megaladon.sharcnet.ca idle
17 bala--->128 CPUs total, 13 idle, 115 busy; 42 jobs running; 0 suspended, 0
       queued.
18 bala bala.sharcnet.ca idle
19 tiger-->Erro ao contactar tiger
20 bull--->384 CPUs total, 10 idle, 374 busy; 180 jobs running; 0 suspended,
       53 queued.
21 bull bull.sharcnet.ca busy
22 whale-->Erro ao contactar whale
23 saw--->2672 CPUs total, 387 idle, 2285 busy; 250 jobs running; 0 suspended,
        152 queued.
24 saw saw.sharcnet.ca busy
25 narwhal--->1064 CPUs total, 28 idle, 1036 busy; 115 jobs running; 0
       suspended, 195 queued.
26 narwhal narwhal.sharcnet.ca busy
27 *Requisitos preenchidos
28 *Executando comando em-> bala bala.sharcnet.ca idle
29 ====Preparando Resposta====
30 submitted as jobid 102314
31 bala.sharcnet.ca
32 ========
33 Verificando Callback
34 Achei WSAddress
35 MessageID: 54323
36 WS_Addressing: http://localhost:18181/AsynchronousClient/response
37 ========
```

```
38 Monitorando Job
39 Monitorando: bala.sharcnet.ca tlechuga 102314
40 Monitorando Job: 102314 Em: bala.sharcnet.ca
41 Job Concluido
42 ========
43 Lendo resultado
44 Preparando resposta
45 Resposta enviada
46 ====Sharcnet Concluiu====
47
48 ====Executando outros servicos====
49 Baixando Resultados
50 Gerando Mosaico
51 Modelando Background
52 Combinando Fundo
53 Corrigindo Mosaico
54 ====BPEL Concluiu====
55
56 Execucao Finalizada: 14/06/2010 - 23:01:42
```

Código B.6: *Log* de execução da aplicação Montage (3).

Referências Bibliográficas

[1] Service-oriented computing. *Commun. ACM*, 46(10), 2003.

[2] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, Berlin, Germany, October 2004.

[3] Kaizar Amin, Gregor von Laszewski, Mihael Hategan, Nestor J. Zaluzec, Shawn Hampton, and Albert Rossi. Gridant: A client-controllable grid workflow system. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[4] Apache. JUDDI. `[http://ws.apache.org/juddi/]`. Acessado em março de 2010.

[5] L. Baresi, S. Guinea, and P. Plebani. Ws-policy for service monitoring. In *6th VLDB Intl. Workshop on Technologies for E-Services, volume 3811 of Lect. Notes in Computer Science*, pages 72–83. Springer, 2006.

[6] Michael A. Bauer. High performance computing: the software challenges. In *PASCO '07: Proceedings of the 2007 international workshop on Parallel symbolic computation*, pages 11–12, New York, NY, USA, 2007. ACM.

[7] Marta Beltrán and Antonio Guzmán. How to balance the load on heterogeneous clusters. *Int. J. High Perform. Comput. Appl.*, 23(1):99–118, 2009.

[8] Siu-Cheung Chau and Ada Wai-Chee Fu. Load balancing between computing clusters. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003*, pages 548–551, Aug. 2003.

[9] Siu-Cheung Chau and Ada Wai-Chee Fu. Load balancing between heterogeneous computing clusters. In *Proceedings of the Second International Workshop on Grid and Cooperative Computing. GCC2003*, pages 75–82, Aug. 2003.

[10] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (WSDL) version 2.0 part 1: Core language, 2006.

[11] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2000.

[12] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6:86–93, 2002.

[13] Dieter Cybok. A grid workflow infrastructure. *Concurr. Comput. : Pract. Exper.*, 18(10):1243–1254, 2006.

[14] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework version 1.0, 2004.

[15] Mario Dantas. *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*. Axcel, 2005.

[16] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S.L. Price. Grid service orchestration using the business process execution language(BPEL). *Journal of Grid Computing*, 3(3-4):283–304, 2005.

[17] Eviware. The web services testing tool, 2007. `[http://www.soapui.org/]`. Acessado em março de 2010.

[18] Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Cary, and Michael Robinson. Grid service composition in BPEL for scientific applications. *GADA*, 2007.

[19] Geoffrey C. Fox and Dennis Gannon. Workflow in grid systems. *Concurr. Comput. : Pract. Exper.*, 18(10):1009–1019, 2006.

[20] Diego Zuquim Guimarães Garcia and Maria Beatriz Felgar de Toledo. Achieving autonomic web service integration: A quality of service policy-based approach. *International Transactions on Systems Science and Applications*, 3:45–63, 2007.

[21] I. Horton. *Begining Java*. Wrox, 1997.

[22] S. Hunold, T. Rauber, and G. Runger. Dynamic scheduling of multi-processor tasks on clusters of clusters. In *2007 IEEE International Conference on Cluster Computing*, pages 507–514, Sept. 2007.

[23] S. Hunold, T. Rauber, and F. Suter. Scheduling dynamic workflows onto clusters of clusters using postponing. In *8th IEEE International Symposium on Cluster Computing and the Grid. CCGRID '08*, pages 669–674, May 2008.

[24] IBM. Ws-addressing, 2007. [http://www-106.ibm.com/developerworks/library/specification/ws-add]. Acessado em março de 2010.

[25] R. Khalaf, A. Keller, and F. Leymann. Business processes for web services: principles and applications. *IBM Syst. J.*, 45(2):425–446, 2006.

[26] Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL: A workflow framework for grid services. Technical report, ARGONNE NATIONAL LABORATORY, 2002.

[27] Thiago A. Lechuga, Maria Beatriz Felgar de Toledo, and Miriam A. M. Capretz. A global scheduler for sharcnet. *SHARCNET Research Day*, 2009.

[28] Thiago A. Lechuga, Maria Beatriz Felgar de Toledo, and Miriam A. M. Capretz. An infrastructure for executing ws-bpel workflows in a cluster of clusters. *IEEE symposium on Computers and Communications (ISCC 2010)*, 2010.

[29] Frank Leymann. Choreography for the grid: towards fitting BPEL to the resource framework. *Concurr. Comput. : Pract. Exper.*, 18(10):1201–1217, 2006.

[30] Frank Leymann, Dieter Roller, and Marc-Thomas Schmidt. Web services and business process management. *IBM Systems Journal: New Developments in Web Services and Electronic Commerce*, 41:198–211, 2002.

[31] Heiko Ludwig. Web services QoS: External SLAs and internal policies - or: How do we deliver what we promise? In *WISE 03: Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops*, pages 115 – 120. Springer, 2003.

[32] Ru-Yue Ma, Yong-Wei Wu, Xiang-Xu Meng, Shi-Jun Liu, and Li Pan. Grid-enabled workflow management system based on BPEL. *Int. J. High Perform. Comput. Appl.*, 22(3):238–249, 2008.

[33] Ben Margolis. *SOA for the Business Developer: Concepts, BPEL, and SCA (Business Developers series).* Mc Press, 2007.

[34] Gordon Moore. Moores law, 1965. [http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf]. Acessado em março de 2010.

[35] NASA/IPAC. Montage. [http://irsa.ipac.caltech.edu/Montage/]. Acessado em março de 2010.

[36] NSF. Teragrid. [https://www.teragrid.org/]. Acessado em março de 2010.

[37] T. N'Takpe, F. Suter, and H. Casanova. A comparison of scheduling approaches for mixed-parallel applications on heterogeneous platforms. In *Sixth International Symposium on Parallel and Distributed Computing. ISPDC '07*, pages 35–35, July 2007.

[38] OASIS. Business process execution language for web services version 2.0, 2007. [http://docs/oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html]. Acessado em março de 2010.

[39] OASIS UDDI Specifications TC. UDDI specification. [http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm]. Acessado em março de 2010.

[40] Oracle. Open ESB. [http://open-esb.dev.java.net/]. Acessado em março de 2010.

[41] Jinhui Qin and M.A. Bauer. An improved job co-allocation strategy in multiple hpc clusters. In *21st International Symposium on High Performance Computing Systems and Applications, 2007. HPCS 2007*, pages 18–18, May 2007.

[42] Jinhui Qin and Michael Bauer. A study on job co-allocation in multiple hpc clusters. *Annual International Symposium on High Performance Computing Systems and Applications*, 0:3, 2006.

[43] SHARCNET. [http://www.sharcnet.ca/]. Acessado em março de 2010.

[44] Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF grid workflows. *Concurr. Comput. : Pract. Exper.*, 18(10):1229–1241, 2006.

[45] Koon Leai Larry Tan and Kenneth J. Turner. Orchestrating grid services using BPEL and Globus Toolkit 4. In *PGNet 2006 - The 7th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, pages 31–36, 2006.

[46] Top500. [http://www.top500.org/]. Acessado em março de 2010.

[47] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, C. Pautasso, T. Heinis, R. Gronmo, Hjordis Hoff, Arne-Jorgen Berre, M. Glittum, and S. Topouzidou. Developing scientific workflows from heterogeneous services. *SIGMOD Rec.*, 35(2):22–28, 2006.

[48] S. Tuecke et al. Open grid services infrastructure (OGSI) version 1.0, 2003. Global Grid Forum Draft Recommendation.

[49] W3C. XML schema, 2004. `[http://www.w3.org/XML/Schema.html]`. Acessado em março de 2010.

[50] W3C. Web services policy 1.2 - attachment (WS-PolicyAttachment), 2006. `[http://www.w3.org/Submission/WS-PolicyAttachment/]`. Acessado em março de 2010.

[51] W3C. Web services policy 1.2 - framework (WS-Policy), 2006. `[http://www.w3.org/Submission/WS-Policy/]`. Acessado em março de 2010.

[52] Workflow management coalition. `[http://www.wfmc.org/]`. Acessado em março de 2010.

[53] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *ArXiv Computer Science e-prints*, March 2005.