GIOVAN ARAÚJO DE MARCO

PROPOSTA DE UMA ARQUITETURA PARA WORKFLOW TRANSACIONAL DISTRIBUÍDO UTILIZANDO WEBSERVICES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

CURITIBA
2003

GIOVAN ARAÚJO DE MARCO

PROPOSTA DE UMA ARQUITETURA PARA WORKFLOW TRANSACIONAL DISTRIBUÍDO UTILIZANDO WEBSERVICES

Dissertação de Mestrado apresentado ao PPGIA na PUCPR como requisito parcial para a obtenção do título de Mestre em Ciências.

Área de Concentração: *Sistemas de Informação*

Orientador: Prof. Dr. Edgard Jamhour

CURITIBA
2003

De Marco, Giovan. Proposta de uma arquitetura para workflow transacional distribuído utilizando web services. Curitiba, 2003. 167p.

Dissertação de Mestrado – Pontifícia Universidade Católica do Paraná.
Programa de Pós-Graduação em Informática Aplicada.

1.Arquitetura 2. Workflow Transacional 3. Ambiente Distribuído 4.Web Services.
I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t

Esta página é reservada à ata de defesa e termo de aprovação que serão fornecidos pela secretaria após a defesa da dissertação e após efetuadas as correções solicitadas.

Dedico este trabalho aos meus amados pais (João Carlos & Rosemary)
que sempre apoiaram e deram todas as condições
para eu render bons frutos

Giovan Araújo De Marco

AGRADECIMENTOS

Ao meu orientador, prof. Dr. Edgard Jamhour, pela paciência e disposição em participar desta dissertação guiando cada passo deste trabalho concedendo parte de seu valioso tempo, e por ter acreditado sempre no potencial deste trabalho.

A presença e disposição dos professores Dr. Hilton José Silva de Azevedo e Dr. Lau Cheuk Lung em participar na avaliação e defesa deste trabalho.

Ao prof. Dr. Carlos Alberto Maziero por toda a colaboração e ajuda nos tranmites institucionais para obtenção deste título.

A todos meus parentes e amigos que sempre acreditaram e torceram por mim.

Ao grande PAI que me revelou valiosas lições durante a elaboração deste trabalho, dando-me também forças e inspiração para vencer nos momentos mais difíceis.

Muito obrigado.

“O que somos é a conseqüência do que pensamos”
(Siddhartha Gautama, o Buda)

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
DTD	*Document Type Definition*
EDI	*Eletronic Document Interchange*
GPS	*Global Position System*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transport Protocol*
LRT	*Long-Running Transactions*
PDA	*Personal Digital Assistant*
RPC	*Remote Procedure Call*
SOAP	*Simple Object Access Protocol*
TWF	*Transactional Workflow*
UDDI	*Universal Description, Discovery and Integration*
URL	*Uniform Resource Locator*
WAP	*Wireless Application Protocol*
WAPI	*Workflow Application Programming Interface*
WFMC	*Workflow Management Coalition*
WFMS	*Workflow Management System*
Wf-XML	*Workflow eXtensible Markup Language*
WPDL	*Workflow Process Definition Language*
WS IF4T	*Web Services Interface 4 Transacional*
WSDL	*Web Services Description Language*
WSEL	*Web Services Endpoint Language*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*
XPDL	*XML Process Definition Language*

RESUMO

A presente dissertação enfoca o domínio de sistemas de workflow. Sistemas de workflow podem ser definidos como sendo sistemas que apresentam facilidades computadorizadas de automação total ou parcial de um processo de negócio [HOLL95]. A WFMC (*Workflow Management Coalition*) é a principal entidade padronizadora de workflow. O WFMC definiu um modelo de referência de onde se apresentam os principais componentes e interfaces de um sistema de workflow. Neste modelo a interface responsável pela interoperação de workflows distribuídos é conhecida como interface 4 (IF4).

A arquitetura apresentada nesta dissertação propõe uma interface de interoperação (IF4) para sistemas de workflow distribuídos em um cenário Hierárquico (*Nested Subprocesses*). Essa interface é chamada de WS IF4T (Web Services Interface 4 Transacional). A WS IF4T é baseada da especificação abstrata de interface 4 proposta pela WFMC. A WS IF4T interage com o ambiente distribuído através de Web Services. A WS IF4T possui também um Módulo Transacional responsável por coordenar a execução de múltiplos processos em um cenário de workflows distribuído. O Módulo Transacional utiliza o modelo transacional relaxado proposto no SAGAS [GMS87].

A WS IF4T oferece uma interface de interoperabilidade para sistemas de workflows distribuídos, de características transacionais, e com a versatilidade dos web services.

Palavras-Chave: Arquitetura, Workflow Transacional, Ambiente Distribuído, Web Services

ABSTRACT

This work focuses the domain of workflow systems. A Workflow system presents the computerised facilitation or automation of a business process, in whole or part [HOLL95]. The Workflow Management Coalition (WFMC) is the major organization whose mission is to define the workflow standards. WFMC was defined the Workflow Reference Model to describe major components and interfaces within the workflow architecture. In Workflow Reference Model, interface 4 is an interface to distributed workflow interchange.

We present an architecture model to define an interface for distributed workflows systems in a Hierarchical scenario (Nested Subprocesses). We call this interface WS TIF4 (Web Services Transactional Interface 4). WS TIF4 is based on WFMC´s interface 4 abstract specification. WS TIF4 interacts with the distributed environment using Web Services. WS TIF4 has a Transactional Module to coordination among multiple-process in distributed workflow scenario. Transactional Module uses the SAGAS [GMS87] model to implement relaxed transactions.

WS TIF4 is an iteroperability interface for distributed workflow systems with transactional features and Web Services facilities.

Keywords: Architecture, Transactional Workflow, Distributed Environment, Web Services

Capítulo 1

INTRODUÇÃO

1.1. Motivação

Com o crescimento da internet surgiu um fenômeno conhecido como *e-commerce*. Segundo [AND99] pode-se definir *e-commerce* como sendo a distribuição de produtos, serviços e/ou informações que envolvem um requisitante, um canal de entrega e um fornecedor utilizando uma metodologia aceita e uma suposição aceitável de negócio. Essa definição indica que a maioria das interações comerciais computacionais se enquadra no conceito de *e-commerce*.

A realização de negócios através do *e-commerce* envolve a implementação de processos de negócios[1] no qual workflow[2] se apresenta naturalmente como uma solução tecnológica candidata [AND99][LIU01]. O WFMC (*Workflow Management Coalition*) é o principal orgão padronizador de workflow e este defende a implementação de *e-commerce* através de processos de workflow distribuídos, e esta abordagem também será adotada neste trabalho.

É necessário que sistemas de workflow possam interoperar entre si, pois a obtenção de bens e serviços através do *e-commerce* irá necessariamente envolver operações de processos de negócio que serão ativados dentro e entre as organizações. Essa ativação implementa o que se conhece na prática como cadeias de valores (*Value Chains*). *Value Chain* não é, em si mesmo uma nova idéia. EDI (*Eletronic Document Interchange*) oferece também suporte de interoperabilidade entre organizações participantes de uma cadeia de valores. No modelo EDI a interoperabilidade é alcançada através de troca de mensagens contendo objetos de negócios (documentos eletrônicos,ordens de compra, fundos eletrônicos, etc) que são tratados como

[1] Um processo de negócio é o conjunto de atividades que conduzem a um propósito de negócio. Isto inclui quase tudo o que acontece em uma organização envolvendo os processos de produção e produtos [DAVE90].
[2] Um workflow é uma facilidade computadorizada de automação total ou parcial de um processo de negócio [HOLL95].

sendo o *input* dos sistemas de informação das organizações participantes da cadeia de suprimentos. Aplicações EDI são recomendadas para grandes volumes de dados com um suporte seguro para as aplicações de negócios. Contudo sistemas de EDI, têm se mostrado na prática, caros e uma vez em operação eles se tornam inflexíveis [AND99].

A popularidade da internet, seus padrões abertos, bem como a não necessidade de suportar soluções proprietárias caras fez com que as empresas migrassem de EDI para internet [WIL00]. A utilização de *e-commerce* pela internet é mais barato e conveniente, e aumenta a possibilidade de seleção de parceiros entre as empresas [LIU01]. A tecnologia de workflow apresenta-se como uma solução de suporte para esses objetivos [LIU01].

1.1.1 Porque Interoperabilidade de Workflow?

Processos de negócios que operam dentro e entre as organizações para implementar cadeias de valores podem ser usados para proporcionar transações de *e-commerce* que podem ser implementadas através de um conjunto de definições de processos de workflow que são criadas para suportar discretos segmentos do processo como um todo [AND99]. Neste cenário surge a questão: como evitar a criação de ilhas de automação entre os processos de negócios? Esse problema é resolvido através da interoperabilidade de workflow, que possibilita diferentes sistemas de workflow "conversem" entre si através da troca de mensagens que interferem nos processos orientando o gerenciamento das operações que ocorrem na cadeia de valores (*Value Chain*). A interoperabilidade de workflow possibilita que o proprietário da cadeia de valores controle toda a cadeia do início ao fim através de vários sistemas de workflow. A interoperabilidade de workflow permite que as organizações automatizem bastante suas atividades, isso significa que as organizações poderão expandir suas transações de *e-commerce* substancialmente sem ter que aumentar seus custos. *E-Commerce* já é uma atividade global significante e há indícios que os avanços na tecnologia de workflow irá prover meios para que essa atividade cresça ainda mais.

1.1.2 Gerenciamento da Cadeia de Suprimentos (*Supply Chain*)

O gerenciamento da cadeia de suprimentos (*Supply Chain*) é um subconjunto do gerenciamento da cadeia de valores (*Value Chain*) onde o foco é o suprimento de bens manufaturados. A cadeia de suprimentos lida com todas as atividades significantes que ocorrem desde a compra até a entrega de um produto manufaturado. Com isso ela envolve a participação de várias entidades como por exemplo: lojistas, fabricantes, transportadores, banqueiros e também os clientes.

O cenário apresentado na **Figura 1.1** descreve um processo global em uma cadeia de suprimentos onde um lojista encomenda um móvel (um sofá) de um fabricante que por sua vez irá necessitar de uma transportadora para entregar a encomenda.

Figura 1.1 – Um exemplo de *Supply Chain*

A tecnologia de workflow pode ser aplicada para o gerenciamento da cadeia de suprimentos, onde cada participante da cadeia de suprimentos utiliza um sistema gerenciador de workflow (WFMS) para criar, controlar e interagir entre as instâncias[3] dos processos.

O exemplo da **Figura 1.1** ilustra também um cenário de interoperabilidade **hierárquico** (*Nested Subprocess*) onde o processo executado no WFMS do fabricante tem uma atividade encapsulada "Agenda Entrega" que envoca um outro processo completo executado no WFMS do transportador (ver **Figura 1.2**). O mesmo ocorre para a atividade "Envia Ordem p/ Fabricante" do processo no WFMS do Lojista.

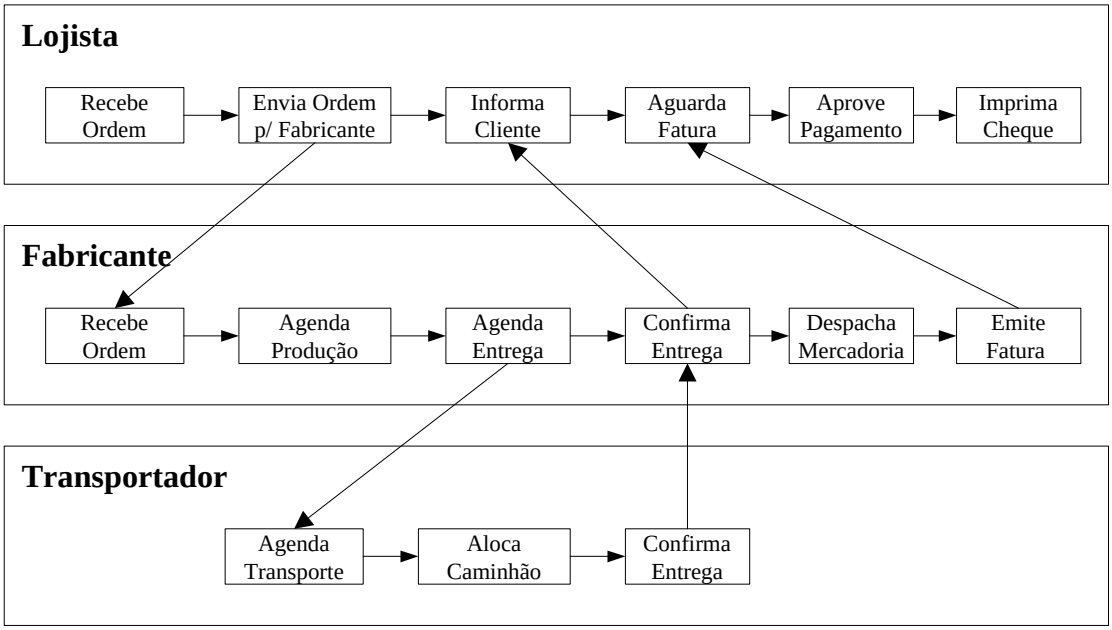

Figura 1.2 – Um exemplo de cenário Hierárquico (*Nested Subprocesses*)

[3] Cada novo processo que é inicializado no WFMS é conhecido como uma instância de processo. Uma instância de processo é a "materialização" de uma definição de processo existente no sistema.

1.1.3 Utilização de padrões

Com a utilização de padrões tem-se um referencial claro de comparação entre as soluções existentes. O WFMC (*Workflow Management Coalition*) é o principal orgão normatizador de tecnologias na área de workflow. A utilização dos padrões propostos pelo WFMC possibilitam que usuários de produtos de workflow possam implementar processos que atravessem as barreiras organizacionais e tecnológicas.

No modelo de referência (ver item 2.2) proposto pelo WFMC tem-se uma interface denominada interface 4 que é a responsável pela padronização de operações que possibilitam a interoperabilidade entre sistemas de workflow.

1.1.4 Transações

Em uma cadeia de suprimentos (*Supply Chain*) é necessário que certos processos sejam tratados como uma transação. Por exemplo, no cenário da **Figura 1.1** o lojista resolveu cancelar o pedido durante a execução do processo. Após esse cancelamento é necessário que o processo seja revertido e finalize em um estado consistente para que não haja nenhum incoveniente para os participantes da cadeia de suprimentos. Este é um problema clássico nesta área de sistemas de informação.

Para que isso seja possível é necessário que o sistema de workflow dos participantes seja transacional. Um workflow transacional constitui um tipo específico de workflow que possui certas características transacionais [RUS95]. Há vários tipos de workflow transacional porém vale frisar que nenhuma das soluções de workflow transacional existentes apresenta uma solução ideal para o problema. Portanto esta é uma área que necessita ainda de muita pesquisa e desenvolvimento para se buscar uma solução ideal que consiga satisfazer as exigências de semântica dos sistemas de workflow sem prejudicar sua eficiência e flexibilidade.

1.1.5 Web Services

Web services surgiram como uma proposta para otimizar o uso da internet através da comunicação entre programas *(program-to-program communication)* [NEW02]. Através da expansão e da utilização de web services, as aplicações localizadas em vários sites distribuídos ao longo da internet poderão ser diretamente integradas e interconectadas como se fossem parte de um único e imenso sistema de informação [NEW02].

Com isso pode-se perceber que web services apresentam uma plataforma viável para prover interoperabilidade para sistemas de workflow operando em uma cadeia de valores através da internet.

1.2. Objetivos

Modelagem, implementação e avaliação de uma interface de interoperabilidade para sistemas de workflow que apresente:

i. Facilidade de adaptação para sistemas de workflow participantes de uma cadeia de valores (*Value Chain*);
ii. Compatibilidade de operação com a interface 4 padronizada pelo WFMC;
iii. Suporte para um cenário de interoperabilidade Hierárquico (*Nested Subprocesses*);
iv. Um módulo transacional para controle de processos que tenha características transacionais herdadas de um modelo transacional existente;
v. Tecnologia de comunicação baseada na plataforma web services.

1.3. Organização do documento

O Capítulo 2 trata dos recorrentes conceituais básicos para a confecção do presente, tratando sobre tópicos pertinentes à Workflow: sua conceituação, modelo de referência, componentes, interfaces, tipos de abordagens (centralizada ou distribuída), topologias e os padrões de interoperabilidade.

O Capítulo 3 apresenta o modelo de workflow transacional abordando temas relacionados como, transações relaxadas e operações ACID com intervenção de usuários, resumindo as soluções pesquisadas para a implementação de workflow transacional. Este capítulo apresenta também uma análise de todas as soluções pesquisadas na área de workflow transacional, apresentando um comparativo entre elas e destacando os requisitos que um bom sistema de workflow transacional deve possuir.

O Capítulo 4 apresenta resumidamente o conceito de web services e sua relação com workflow. Apresentam-se também algumas linguagens para descrição de processos utilizando web services (WSFL, XLANG e BPEL4WS).

O Capítulo 5 traz a proposta de uma arquitetura para workflow transacional utilizando web services. Descreve-se em detalhes o cenário de interoperabilidade suportado, a arquitetura e seus componentes (interfaces e módulo transacional).

O Capítulo 6 apresenta um estudo de caso e uma simulação para validar o funcionamento da proposta.

O Capítulo 7 finaliza com a conclusão apresentando uma explanação geral sobre a proposta, objetivos alcançados e os trabalhos futuros.

Capítulo 2

WORKFLOW

Inicialmente Workflow foi um termo atribuído à tecnologia de ponta da automação de escritórios. A troca de informação sem papeis, informatização de tarefas, sistemas virtuais de arquivo e a cooperação entre pessoas em localizações remotas, tanto em escritórios residenciais como em escritórios internacionais ou sedes de empresa, são uma realidade há já algum tempo. A usual utilização de processadores de texto, bases de dados e folhas de cálculo, vem em muitas empresas a ser substituída pelo conceito de aplicações integradas de workflow.

Para entender-se do que se trata o conceito de workflow este capítulo apresenta o que são sistemas de workflow conceituando e ilustrando seus principais componentes, características, abordagens e padrões de interoperabilidade.

2.1. Workflow Management Coalition (WFMC)

A Workflow Management Coalition (WFMC), fundada em agosto de 1993, é uma organização internacional sem fins lucrativos de vendedores, usuários, analistas e grupos de pesquisa em workflow.

A missão da coalizão é promover e desenvolver o uso de workflow pelo estabelecimento de padrões para a terminologia de software, interoperabilidade e conectividade entre produtos de workflow. Atualmente a coalizão conta com mais de 300 membros, estabelecendo-se como a entidade primária de referência na tecnologia de software em Workflow que se encontra em constante expansão.

Dentre os objetivos específicos da missão da WFMC podemos destacar: aumentar os investimentos dos clientes na tecnologia de workflow, reduzir os riscos em utilizar produtos

de workflow e a expandir o mercado através da estimulação e conscientização para o uso desta tecnologia.

2.2. Workflow – Modelo de Referência WFMC

O Workflow é centrado na automação de procedimentos onde documentos, informações ou tarefas são passados entre participantes de acordo com um conjunto de regras definido para alcançar, ou contribuir parcialmente com uma meta do negócio. Um workflow é uma facilidade computadorizada de automação total ou parcial de um processo de negócio [HOLL95].

Um processo de negócio é o conjunto de atividades que conduzem a um propósito de negócio. Isto inclui quase tudo o que acontece em uma organização envolvendo os processos de produção e produtos [DAVE90].

Um sistema de gerenciamento de workflow (Workflow Management System - WFMS) provê automação procedural de um processo de negócio através do gerenciamento da seqüência de atividades e invocando os responsáveis ou recursos de TI (Tecnologia de Informação) para execução dessas atividades. Pela definição formal tem-se que um WFMS é um sistema que define, gerencia e executa workflows pela execução de software cuja ordem está orientada por uma representação computacional da lógica do workflow [HOLL95].

Um processo de negócio pode ter uma duração de alguns minutos até alguns dias ou meses, dependendo de sua complexidade e duração das atividades constituintes. Tais sistemas podem ser implementados de várias formas, utilizando uma enorme variedade de TI e infraestrutura de comunicações, operando em um pequeno grupo de trabalho local até um grupo de trabalho distribuído de uma grande corporação.

De uma maneira geral os WFMS podem ser caracterizados como um provedor de serviço em três áreas funcionais [HOLL95]:

- Funções *Build-Time*, que se concentram nas áreas de definição e modelagem de um processo de workflow;
- Funções *Run-Time*, para gerenciamento e controle dos processos de workflow dentro de um ambiente operacional e seqüenciamento das atividades participantes de cada processo;
- Interações *Run-Time*, realizadas entre usuários e aplicações de T.I. para o processamento das várias atividades presentes em um workflow.

A **Figura 2.1** ilustra o relacionamento entre essas três características principais de um WFMS.

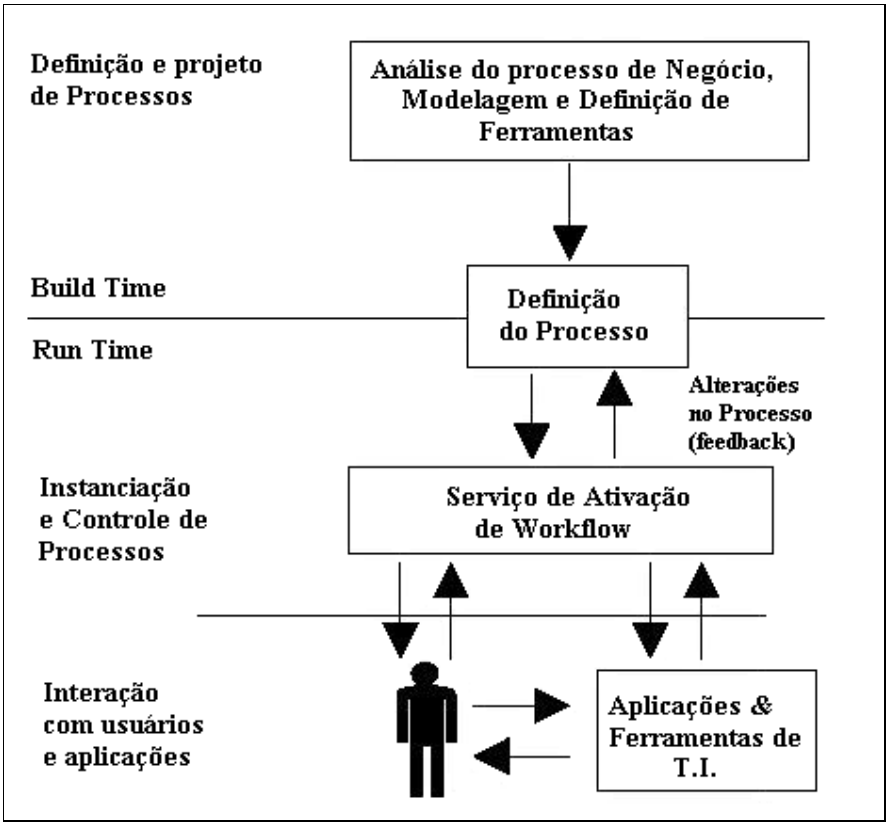

Figura 2.1 – Características de um sistema de workflow [HOLL95]

Todos os sistemas de workflow possuem um número de componentes genéricos que interagem de maneiras correlatas, produtos distintos apresentarão diferentes níveis de compatibilidade com cada um desses componentes. Para alcançar a interoperabilidade entre os produtos de workflow, um conjunto padronizado de interfaces e troca de dados faz-se necessário. Um número distinto de cenários de interoperabilidade pode ser construído tomando-se como referência tais interfaces. A **Figura 2.2** ilustra os principais componentes e interfaces da padronização da arquitetura de workflow.

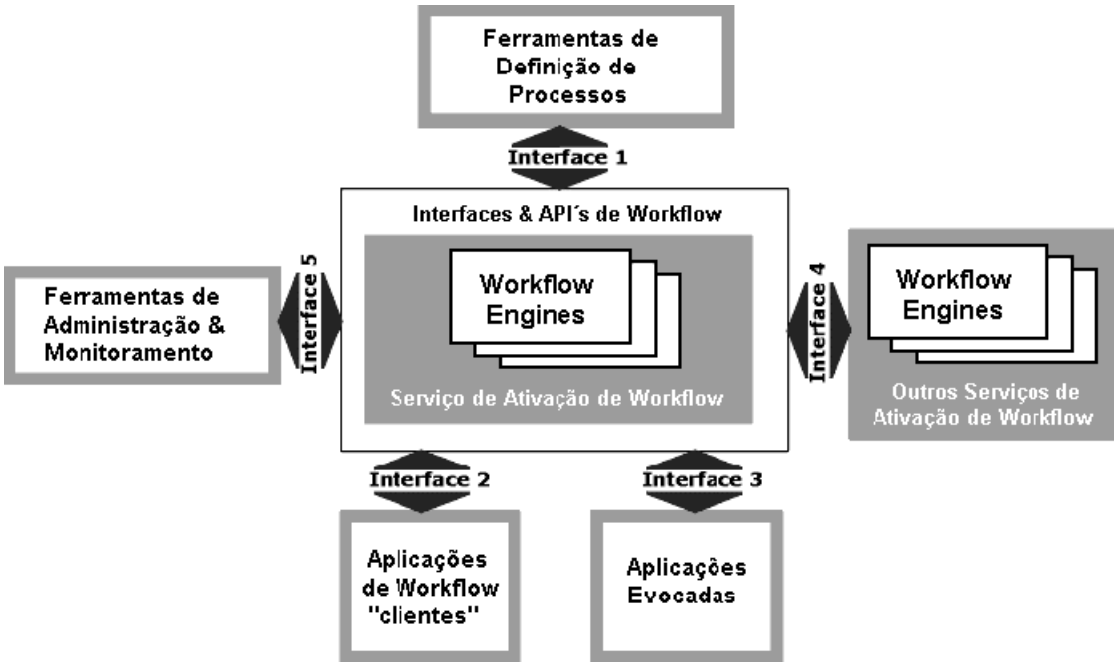

Figura 2.2 – Workflow Reference Model: components & interfaces [HOLL95]

2.2.1 Componentes do Workflow Reference Model

Faz-se necessário descrever os principais componentes do Workflow Reference Model para que haja um melhor entendimento do sistema como um todo.

- **Serviço de Ativação de Workflow *(Workflow Enactment Service)***: consiste em um serviço de software composto de um ou mais workflows *engines* responsável por criar, gerenciar e executar instâncias de workflow. As aplicações podem interagir com este serviço através do *Workflow Application Programming Interface* (WAPI);
- ***WorkFlow Engine***: serviço de software ou "engine" que provê o ambiente de execução em tempo real para uma instância de um processo de workflow;
- **Ferramentas de Definição de Processos *(Processs Definition Tools)***: toda variedade de ferramentas que podem ser utilizadas para analisar, modelar, descrever e documentar um processo de negócio. Tais ferramentas podem ser desde lápis e papel até uma sofisticada ferramenta de modelagem computacional. O modelo de workflow não se preocupa com a natureza e nem como essas ferramentas irão interagir durante o processo de construção;
- **Aplicações de Workflow Cliente *(Workflow Client Applications)***: softwares que permitem aos participantes do workflow interagir com os serviços de ativação do

workflow, a fim de iniciar processos, mostrar listas de tarefas, lançar aplicações e acessar dados relativos ao workflow;

- **Aplicações Evocadas *(Invoked Applications)***: softwares que são disparados pelo serviço de ativação de workflow, de acordo com a definição do processo, para iniciar ou executar uma atividade;
- **Ferramentas de Administração & Monitoramento *(Administration & Monitoring Tools)***: softwares que suportam o monitoramento, controle, configuração e otimização da execução de um workflow em tempo real.

2.2.2 Interfaces do Workflow Reference Model

A WFMC definiu as principais interfaces do *Workflow Reference Model* para que haja uma estruturação e padronização entre os diversos fabricantes de sistemas de workflow.

- **Interface 1:** *Workflow Definition Interchange;*
- **Interface 2:** *Workflow Client-Application Interface;*
- **Interface 3:** *Invoked Application Interface;*
- **Interface 4:** *Workflow Programming Interoperability Functions;*
- **Interface 5:** *Administration and Monitoring Interface.*

2.3. Software de Ativação de Workflow *(Workflow Enactment Software)*

O software de ativação de workflow consiste em um ou mais workflows *engines*, que são responsáveis por gerenciar individualmente as instâncias dos processos. O software de ativação de workflow pode ser implementado segundo uma abordagem centralizada ou com uma abordagem distribuída.

2.3.1 Workflow Centralizado

Nesta abordagem um único workflow engine é responsável por gerenciar todos os processos de workflow da empresa. A **Figura 2.3** ilustra tal cenário.

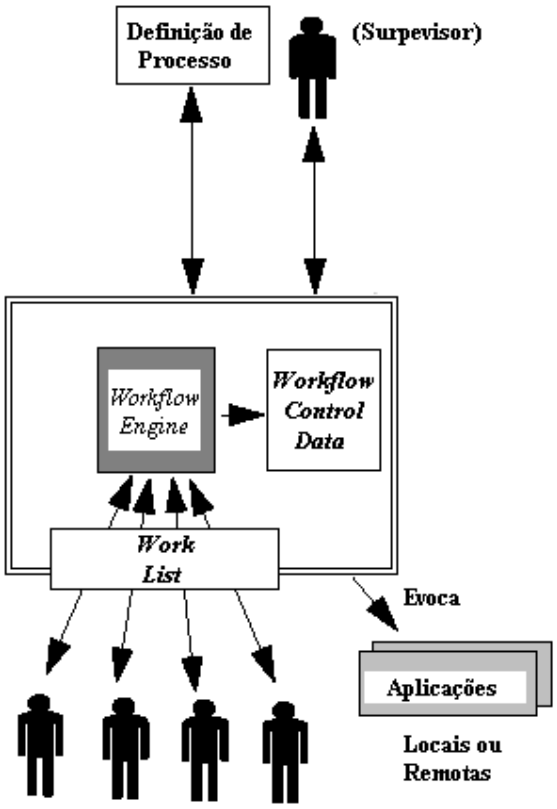

Figura 2.3 – Workflow centralizado

Workflow Control Data*:* dados internos gerenciados pelo WFMS ou workflow *engine*.
Work List: uma lista com os itens de trabalho que compreendem as atividades a serem cumpridas.

2.3.2 Workflow Distribuído

Nesta abordagem são utilizados dois ou mais workflows *engines* dentro de uma mesma empresa. Sendo que cada workflow *engine* é responsável por um subconjunto de processos da empresa ou, cada workflow *engine* pode gerenciar partes em separado de um mesmo processo. A Figura 2.4 descreve o esquema de um workflow distribuído[4].

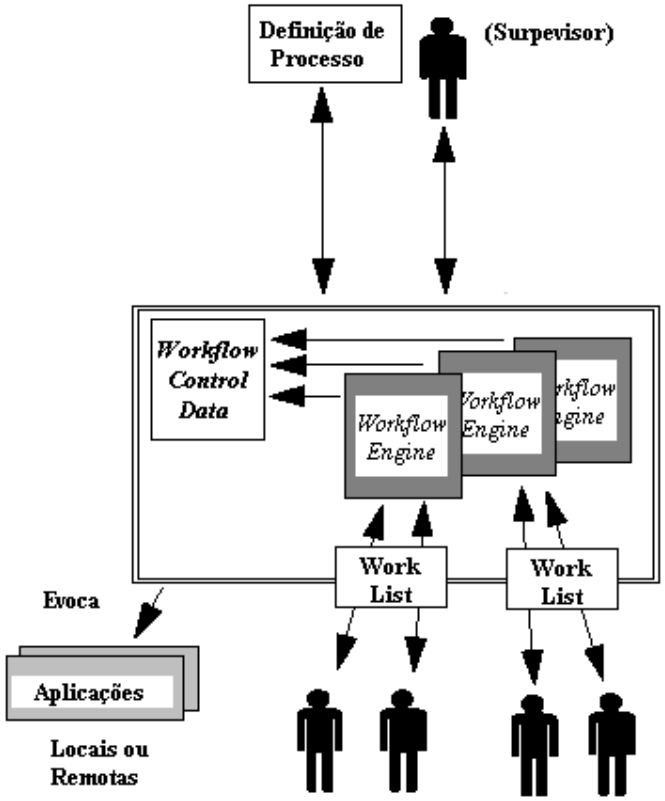

Figura 2.4 – Workflow distribuído

[4]A proposta desta dissertação trabalha com a hipótese de um cenário de workflow distribuído.

2.4. Interoperabilidade entre Serviços de Workflow

Apesar da variedade, sistemas de workflow apresentam características comuns. Essas características representam a base para o desenvolvimento da interoperabilidade entre produtos diferentes.

O modelo de referência descreve um modelo padrão para construção de sistemas de workflow e identifica quatro alternativas de implementação que cobrem os principais cenários de execução de um processo de workflow.

2.4.1 Interoperação em Cascata (*Chained*)

Neste modelo o serviço de workflow A controla o processo A e o serviço de workflow B é responsável pelo processo B. Quando há a migração de A para B o controle do serviço de workflow A é passado integralmente para o serviço de workflow B. Vale ressaltar que nenhum serviço de workflow controla o processo como um todo. A **Figura 2.5** abaixo ilustra tal situação.

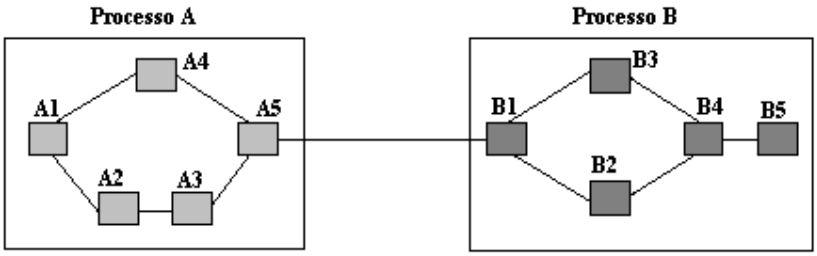

Figura 2.5 – Interoperação em cascata *(Chained)*

2.4.2 Interoperação Hierárquica (*Nested Subprocesses*)

Neste modelo permite-se que um processo executado em um serviço de workflow (*Workflow Service A)* tenha uma ou mais de suas atividades (A3) encapsulada como sendo um processo completo (Processo B) que é executado em um outro domínio de workflow (*Workflow Service B)*. Neste modelo o controle do workflow sempre retorna ao serviço de workflow que controla o processo original. Com isso o serviço de workflow primário mantém uma visão do processo como um todo. A **Figura 2.6** mostra um cenário deste modelo.

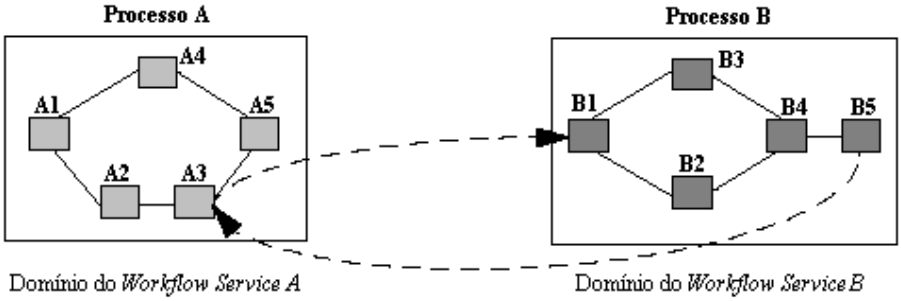

Figura 2.6 – Intertoperação Hierárquica

2.4.3 Interoperação Indiscretamente Conectada (Peer-to-Peer)

Esse modelo permite um ambiente misto onde um processo (Processo C) que contém atividades que podem ser executadas através de múltiplos serviços, constituindo assim um cenário de domínio compartilhado. Na **Figura 2.7** as atividades C1, C2 e C5 são coordenadas pelo servidor A enquanto que as atividades C3, C4 e C6 são coordenadas pelo servidor B. Deste modo o controle das atividades do fluxo, passa de um serviço de workflow para outro, alternadamente ao longo do processo.

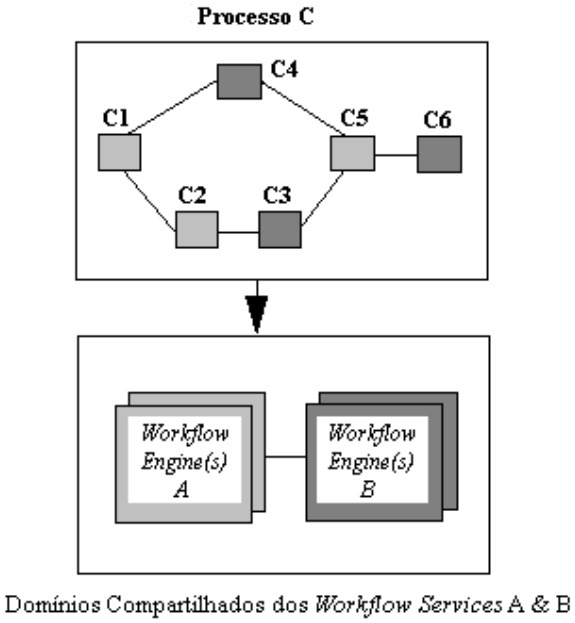

Figura 2.7 – Interoperação Peer-to-Peer

2.4.4 Interoperação de Sincronismo em Paralelo (Parallel Synchronised)

Este modelo permite que dois processos operem independentemente, mas que eventualmente necessitarão de pontos de sincronização para garantir a consistência de seqüenciamento dos processos. Este mecanismo facilita a implementação de várias funcionalidades tais como, scheduling de atividades em paralelo, checkpointing para recuperação de dados ou transferência de informações relevantes entre diferentes instâncias de workflow. A **Figura 2.8** abaixo ilustra este cenário.

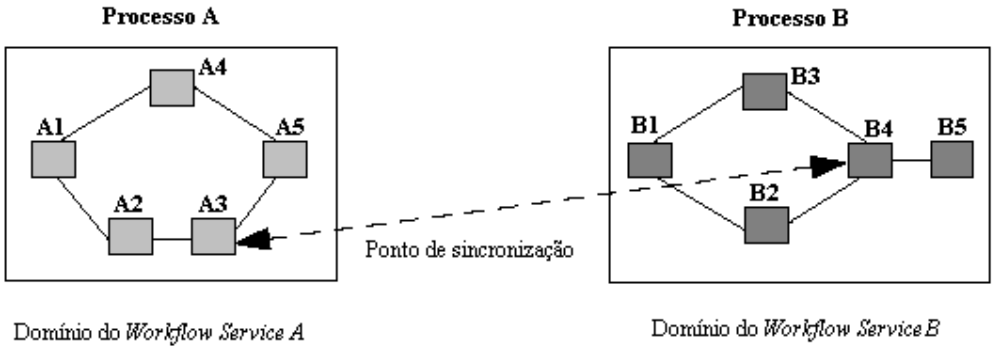

Figura 2.8 – Interoperação de sincronismo em paralelo *(Parallel Synchronised)*

2.5. Padrões de Interoperação entre Workflow Engines

O WFMC ao longo destes anos definiu alguns padrões de comunicação para a interoperabilidade de sistemas de workflow distintos ou distribuídos. A interface 4 é responsável por apresentar os mecanismos responsáveis pela interoperação entre sistemas de workflow. Dentre as propostas de padronizações realizadas para esta interface podemos citar em ordem cronológica de criação:

- Interoperability Internet e-mail MIME Binding (Out 1996);
- Interoperability Wf-XML Binding (Abr 1999).
- Interoperability XML-HTTP Binding (Fev 2000, em submissão);

2.5.1 Internet e-mail MIME Binding

Essa proposta provê uma especificação abstrata para alcançar-se a interoperabilidade entre dois ou mais workflow engines através da definição de tipos e formatação de mensagens utilizando internet e-mail por meio da codificação MIME como mecanismo de transporte.

A mensagem MIME é baseada na especificação MIME (Multipurpose Internet Mail Extension - RFC-2045 a RFC-2049). Este padrão define mecanismos para que o receptor possa identificar o tipo de dado sendo transmitido (texto, imagem, áudio, vídeo, programas binários, etc).

Nesta arquitetura cada workflow engine é identificado unicamente pela sua caixa postal (mail box) que recebe todos os processos de outros workflow engine externos. Cada mensagem que um workflow engine envia para outra é numerada, e a identificação de uma mensagem ocorre de acordo com o seu tipo e pela sua posição na seqüência de mensagens recebidas. A **Figura 2.9** ilustra o cenário deste padrão de interoperabilidade.

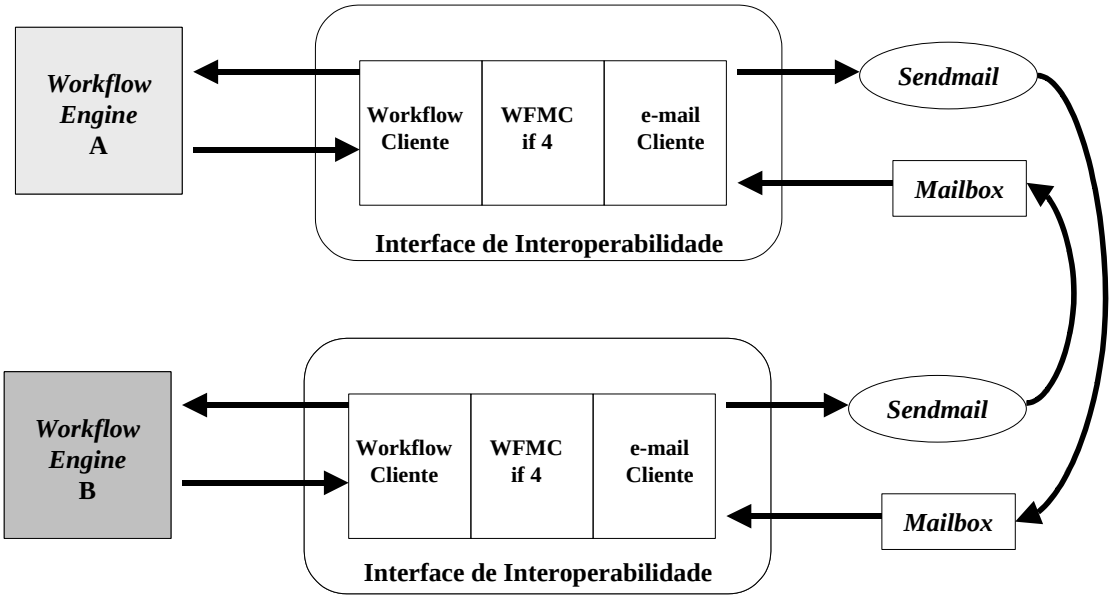

Figura 2.9 – Interoperabilidade através Internet mail & MIME Binding

2.5.2 Internet Wf-XML Binding

Esse modelo de interoperabilidade baseia-se na troca de mensagens do tipo pedido e resposta *(request-response)*. Através do envio e recebimento de mensagens apropriadas pode-se obter a interoperabilidade entre os serviços de ativação de workflow.

Uma função interoperável individual é conhecida como uma operação. Cada operação pode ser passada através de um conjunto de parâmetros de pedidos *(request parameters)* e retornando um conjunto de parâmetros de resposta *(response parameters)*. As operações são divididas em grupos diferentes para melhor identificar o seu contexto. O grupo primário de operações requeridas para interoperabilidade é composto pelas seguintes operações: *ProcessDefinition, ProcessInstance* e *Observer*.

As operações de *ProcessDefinition* são o grupo fundamental para interação entre serviços genéricos. Elas representam a descrição das funções mais básicas de serviço, que são os recursos primários de um processo que será criado. Todo processo definido é ativado segundo um identificador único, seja do requisitante ou o do serviço de interoperabilidade. O recurso de identificador é utilizado para referenciar todos os processos que serão executados no sistema a partir de uma definição de processos.

As operações de *ProcessInstance* representam a ativação atual de uma dada definição de processo que terá em separado seu próprio identificador de recurso separado do identificador de sua definição. Quando um processo é instanciado, ele ganha um identificador único do sistema, o que permite que várias instâncias de uma mesma definição de processos sejam criadas simultaneamente. Sendo cada instância um processo único no sistema, essa instância existirá somente uma única vez e uma vez criada poderá ter os seguintes estados: inicializada *(Started)*, executada *(Running)*, ativa *(Active)*, suspensa *(Suspended)*, completada *(Completed)* ou terminada *(Terminated)*.

O grupo do *Observer* provê meios para que uma instância de um processo possa comunicar ao sistema o seu estado atual. Em um cenário de interoperação hierárquica (vide item 2.4.2), deve haver uma maneira do comunicar o processo primário quando um de seus subprocessos for completado. O grupo do *Observer* é responsável por notificar tais informações. As operações pertencentes aos respectivos grupos são:

- **CreateProcessInstance:** essa operação é utilizada para instanciar uma definição de processos conhecida. A instância será criada de acordo com os seus dados de entrada e em seguida será inicializada;
- **GetProcessInstanceData:** essa operação é utilizada para obter-se valores de certas propriedades de uma dada instância;
- **ChangeProcessInstanceState:** operação usada para modificar o estado de uma certa instância de um processo; por exemplo passar uma instância de um estado executando *(Running)* para um estado suspenso *(Suspended)*;
- **ProcessInstanceStateChanged:** essa operação é utilizada para notificar mudanças que ocorram em um estado de uma instância. Essas mudanças serão notificadas independentemente da situação da instância, seja ela finalizada normalmente ou abortada.

A **Figura 2.10** ilustra um diagrama que mostra o relacionamento entre os grupos e operações explicados acima.

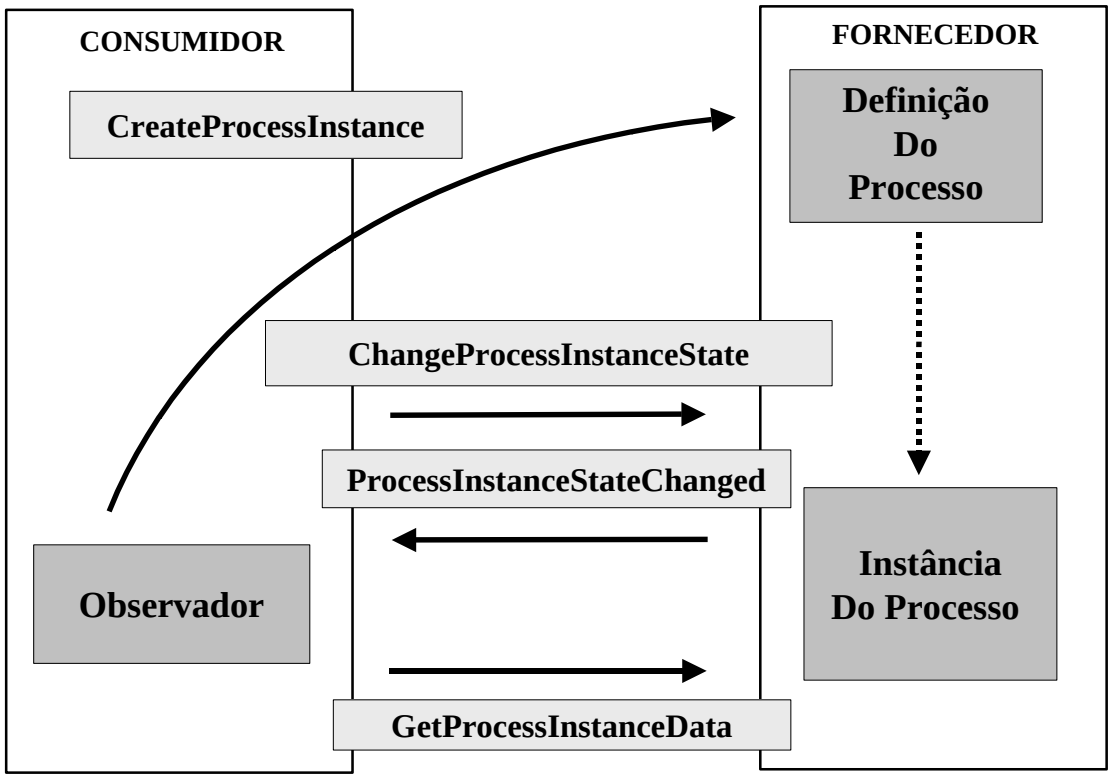

Figura 2.10 – Operações Wf-XML

O DTD *(Document Tpye Definition)* da **Figura 2.11** define um a estrutura de uma mensagem Wf-XML:

```
<!ELEMENT WfMessage (WfTransport?, WfMessageHeader, WfMessageBody)>
<!ATTLIST WfMessage Version CDATA #REQUIRED>
```

Figura 2.11 – Estrutura básica de uma mensagem Wf-XML

O elemento principal da mensagem Wf-XML é chamado de *WfMessage*. Este elemento carrega consigo o atributo de versão (v*ersion*) que indica a versão em particular na qual a mensagem estará em conformidade. Cada mensagem Wf-XML contém uma sessão opcional para informação de transporte *WfTransport*, um campo obrigatório de cabeçalho *WfMessageHeader*, e um campo que contém a mensagem em si *WfMessageBody*. Se for necessário, o campo de *WfTransport* pode ser utilizado para transportar informação relevante a uma implementação particular de protocolo. O cabeçalho *(WfMessageHeader)* contém informação relevante para roteamento e pré-processamento da mensagem. O corpo da mensagem contém a operação específica da mensagem. A **Figura 2.12** ilustra um esquema de uma mensagem Wf-XML.

```
<?xml version=”1.0”?>
<WfMessage Version=”1.0”>
<WfTransport/>
<WfMessageHeader>
    • <Request> OU <Response>
</WfMessageHeader>
<WfMessageBody>
    • <CreateProcessInstance> OU
    • <GetProcessInstanceData> OU
    • <ChangeProcessInstanceState> OU
    • <ProcessInstanceStateChanged> OU
</WfMessageBody>
</WfMessage>
```

Figura 2.12 – Esquema de uma mensagem Wf-XML

A **Figura 2.13** ilustra a arquitetura de um sistema de workflow interoperando com a interface Wf-XML.

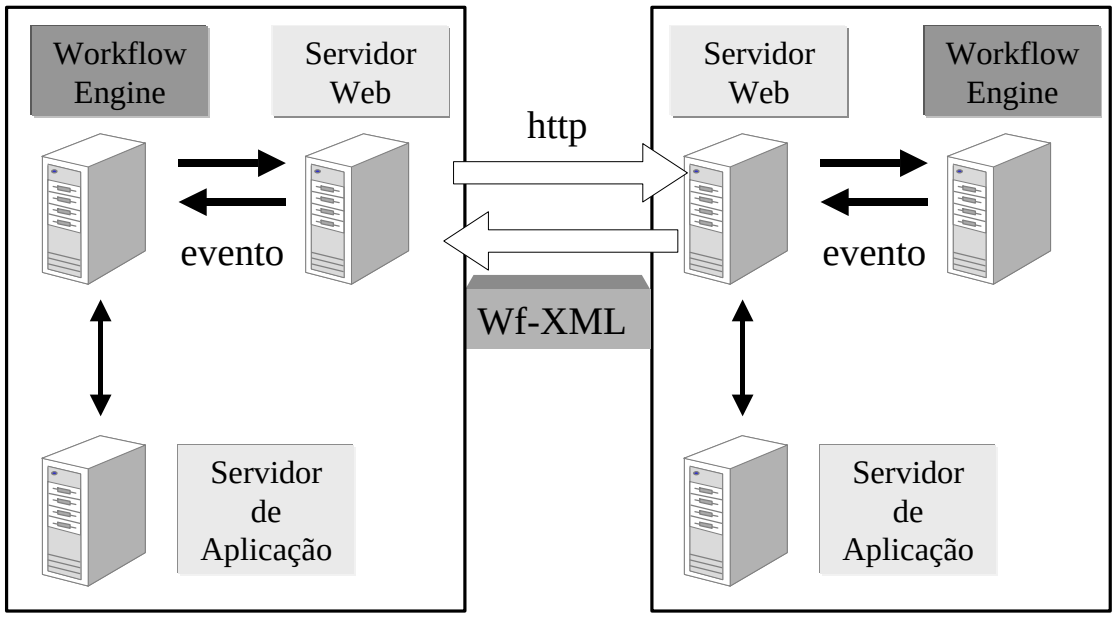

Figura 2.13 – Interoperabilidade Wf-XML

2.5.3 XML-HTTP Binding

Essa proposta apresenta um protocolo baseado em Wf-XML com extensões para aplicações EDI *(Electronic Document Interchange)* empresariais. Este protocolo apresenta algumas definições e interfaces novas para o Wf-XML porque o Wf-XML não é satisfatório em algumas funcionalidades de sistemas EDI, como por exemplo, mensagens assíncronas e múltiplas. Em cima disto foi proposto pelo WFMC um conjunto de melhorias e interfaces para incrementar a potencialidade da Wf-XML em sistemas EDI [VAR00]:

- **Mensagens assíncronas com garantia de transferência:** define um protocolo para mensagens assíncronas em que uma confirmação é retornada na página HTTP de resposta e o resultado do método executado pelo servidor é transferido para outro pedido HTTP. Assim o servidor pode notificar o cliente dos erros ocorridos durante a execução de um método e o cliente pode consultar o *status* do método que será executado;
- **Múltiplas mensagens**: define um mecanismo para múltiplas mensagens em que um cliente pode monitorar e controlar o *status* de cada pedido e um servidor pode notificar o cliente de erros ocorridos em cada pedido, além disso, este cliente pode transferir múltiplos pedidos em um único pedido HTTP, o que é necessário em sistemas EDI;

- **Formato de dados para Workflow e DTD:** especifica duas maneiras para especificar dados no workflow *ContextData* e *ResultData*. A primeira é utilizada para especificar DTD externos para workflows e a segunda provém uma definição de tipos para melhor interoperabilidade de dados;
- **Interface de Observer para atividades:** para mapeamento do *status* de subprocessos que são essenciais para sistemas EDI;
- **Adição de facilidades de rastreamento e logging**: além das facilidades oferecidas pela Wf-XML, esta especificação adiciona mais dois elementos que provém melhoramentos para rastreamento e *logging*, a *GlobalKey* que é uma tag sempre herdada do processo primário para os processos filhos, facilitando assim a identificação da seqüência, e a *Logging Tag* que apresenta recursos para armazenar uma coleção de *logs*;
- **Validação com DTD**: define formatos de dados com DTD para validar um documento XML.

Este modelo de interação com os servidores workflow é independente do modelo de recursos Wf-XML. O modelo inclui três idéias: missão, operação e fase. A missão inicia quando um servidor faz um pedido Wf-XML e acaba quando ele recebe a resposta deste pedido. As operações podem ser de três tipos: *Register*, *Refer* e *Control*. Uma operação pode ter 4 fases: *Request*, *Acknowledge* (inbound), *Response* e *Acknowledge* (outbound). A **Tabela 2.1** apresenta um sumário das operações, fases e direções do modelo de interação XML-HTTP Binding.

Operação	Fase	Direção	Comentários
Register	*Request*	*outbound*	Para enviar páginas de pedidos Wf-XML.
	Acknowledge	*inbound*	Utilizado para uma evocação assíncrona.
	Response	*inbound*	Para enviar páginas de resposta Wf-XML.
	Acknowledge	*outbound*	Utilizado para uma evocação assíncrona.
Refer	*Request*	*outbound*	Pedido para envio do status de uma requisição Wf-XML. Utilizado para uma evocação assíncrona.
	Acknowledge	*inbound*	Utilizado para uma evocação assíncrona.
	Response	*inbound*	Utilizado para enviar o status de uma requisição Wf-XML. Utilizado para uma evocação assíncrona.
	Acknowledge	*outbound*	Utilizado para uma evocação assíncrona.
Control	*Request*	*outbound*	Pedido para um cancelamento, paralização, ou reinicialização de uma evocação Wf-XML. Utilizado para uma evocação assíncrona.
	Acknowledge	*inbound*	Utilizado para uma envocação assíncrona.
	Response	*inbound*	Para reportar o resultado de uma operação de controle. Utilizado para uma evocação assíncrona.
	Acknowledge	*outbound*	Utilizado para uma evocação assíncrona.

Tabela 2.1 – Sumário das operações, fases e direções do modelo XML-HTTP Binding

A missão começa quando o servidor recebe um pedido Wf-XML do tipo *ProcessDefinition::CreateProcessInstance* ou *ActivityObserver::Complete*, e acaba quando o receptor envia a resposta para o pedido. Na **Figura 2.14**, a missão começa quando a mensagem (1) e recebida pelo *Request-Hander-Y* e acaba quando a mensagem (10) é eviada por ele. O *Request-Handler* tem o papel de gerenciar a missão, interagindo com outros *handlers* como um agente de recursos Wf-XML resources para pedidos assíncronos. No modelo síncrono de mensagens esses handlers não são necessários pois as mensagens podem ser trocadas diretamente pelos recursos Wf-XML.

Há dois tipos de direção das mensagens: *inbound* e *outbound*. Quando um servidor A realiza um pedido Wf-XML para um servidor B, a direção de *outbound* indica que a mensagem é enviada do servidor A para o servidor B, enquanto que uma direção *inbound* indica que a mensagem é enviada de B para A. Na troca de mensagens assíncronas no protocolo HTTP, um pedido estará na direção *outbound*, enquanto que o seu resultado estará na direção *inbound*.

A **Figura 2.14** apresenta um cenário com exemplo de missão, operações e mensagens em um pedido Wf-XML assíncrono. O exemplo descreve como um servidor A realiza um pedido Wf-XML para o servidor B, mostrando o controle de execução e o *status* do pedido juntamente com a resposta Wf-XML retornada pelo servidor B. A missão deve ter pelo menos uma operação, porém o exemplo da figura ilustra três operações.

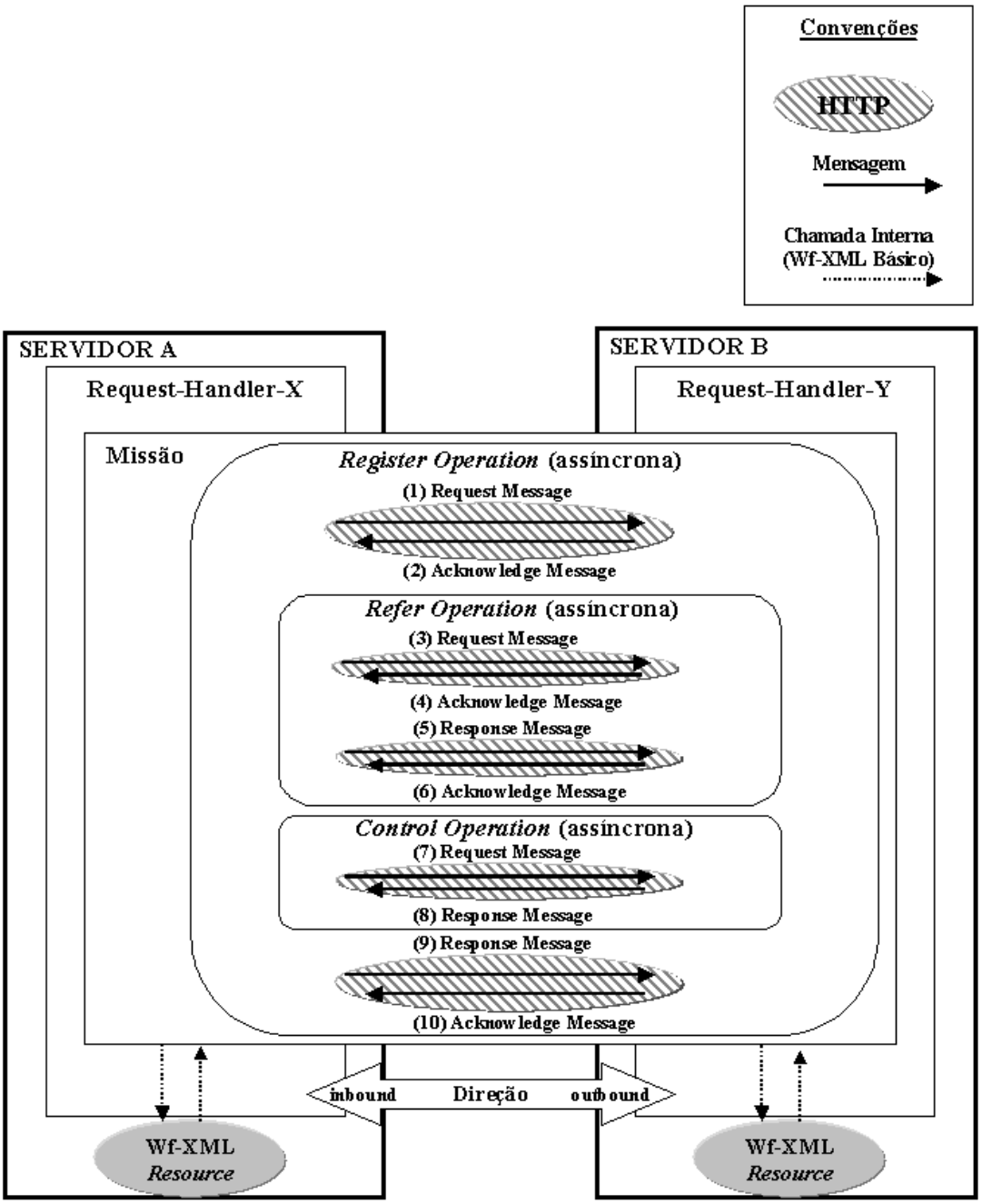

Figura 2.14 – XML-HTTP Binding em mensagens assíncronas

A **Figura 2.15** apresenta um exemplo de missão, operações e mensagens em um pedido Wf-XML síncrono. O exemplo ilustra como o servidor A faz um pedido Wf-XML para o servidor B via uma chamada HTTP.

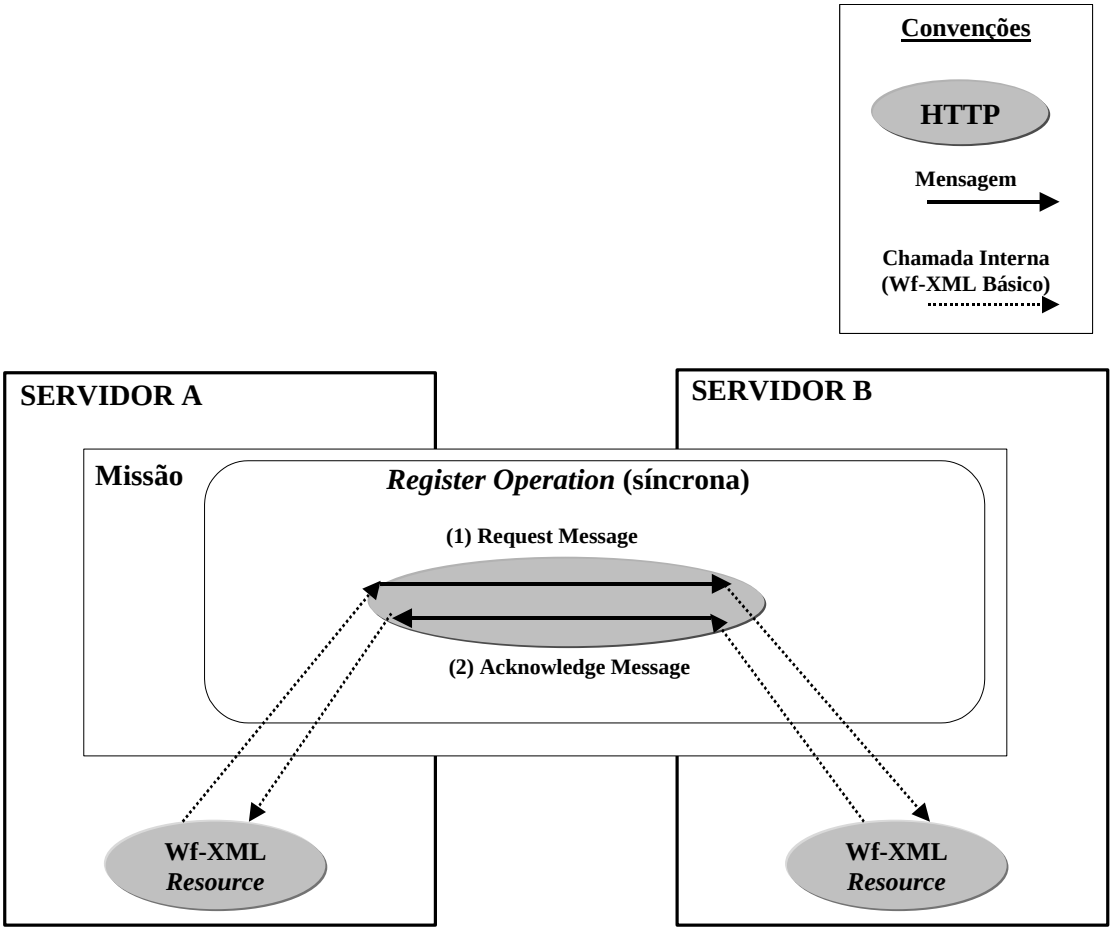

Figura 2.15 – XML-HTTP Binding em mensagens síncronas

Para uma requisição Wf-XML síncrona os formatos das mensagens são similares a pedidos e respostas Wf-XML. A mensagem de requisição é enviada por um recurso Wf-XML e a mensagem de resposta é retornada por uma conexão HTTP. Os *request handlers* não são necessários neste tipo de pedido. A missão apresenta um único registro síncrono e duas mensagens.

2.6. Resumo

O capítulo em questão apresentou os tópicos básicos relacionados com o conceito de workflow.

No item 2.1 apresentou-se a entidade responsável pela padronização e incentivo do workflow (WFMC).

No item 2.2 apresentou-se o modelo de referência da WFMC para os sistemas de workflow, bem como todos os componentes, interfaces.

No item 2.3 apresentam-se as duas abordagens utilizadas para implementação de um sistema de workflow: centralizada ou distribuída.

O item 2.4 ilustra todos os tipos de interoperabilidade para serviços de workflow identificando as quatro alternativas de implementação que cobrem os principais cenários de execução de um processo de workflow: cascata, hierárquico, *peer-to-peer* e sincronismo em paralelo.

Para finalizar o item 2.5 os padrões de interoperação entre workflow *engines* propostos pela WFMC para prover mecanismos de interoperação entre sistemas de workflow

Capítulo 3

WORKFLOW TRANSACIONAL

3.1. Introdução

Atualmente os sistemas de Workflow ganharam abrangência no cenário tecnológico e muitos já são de vital importância para o desempenho das organizações. Desde então tais sistemas vêm sofrendo uma série de melhorias para satisfazer às exigências crescentes das aplicações envolvidas. Um dos recursos chaves desta evolução é a utilização do conceito de transações. Várias aplicações exigem que um processo de negócio seja tratado como uma transação, como é o caso do comércio eletrônico através da Internet.

Apesar da popularidade as gerações atuais de WFMS possuem ainda muitas limitações. Uma dessas limitações é no que diz respeito ao conceito de transação. As aplicações de workflow que necessitam de transações representam apenas uma fração do total e o sucesso alcançado pelos WFMS se deve a seus outros atributos, e não aos aspectos transacionais que, obviamente, carecem [ALON96]. Em contrapartida há uma demanda crescente por aplicações que sejam capazes de implementar um processo de negócio como uma transação, e a tecnologia para atingir este objetivo ainda não está definida.

Este capítulo apresenta o conceito de workflow transacional e os tipos de transações existentes juntamente com algumas soluções propostas na área de workflow transacional.

3.2. Workflow Transacional

O conceito de transação originou-se com pesquisas em banco de dados centralizados, para lidar com os dois maiores problemas que são: gerenciamento de falhas e gerenciamento de execuções concorrentes [KUO96]. Transações têm o objetivo de prover propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Atomicidade garante que a transação é completada por inteiro ou ela não é realizada. Consistência indica que a transação sai de um estado consistente no sistema para outro estado que também o seja. Isolamento garante que a execução de uma transação não é afetada por outras transações executadas concorrentemente. Durabilidade afirma que se uma transação obtém sucesso *(committ)* então os seus efeitos são permanentes.

Pesquisas na área de workflow propõem a noção de workflow transacional (TWF) onde requisitos adicionais podem ser encorporados no topo da especificação tradicional de workflow [WIET01]. Um workflow transacional constitui um tipo específico de workflow que possui certas características transacionais [RUS95]. Um TWF pode ser composto por várias tarefas *(tasks)* de longa duração e tais tarefas podem ser uma transação, um serviço, uma atividade humana ou um outro TWF [KUO96].

O item seguir ilustra os principais modelos de transações utilizadas no contexto prático para workflow.

3.3. Modelos de Transações Estendida e Relaxada

Segundo [RUS95] os modelos transacionais podem ser classificados de acordo com suas características de estrutura, concorrência, dependência, visibilidade, durabilidade, isolamento, atomicidade e critérios de execução concorrente. O termo transação tradicional *(traditional transaction)* é utilizado para transações que garantem as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). As transações estendidas *(extended transactions)* permitem o agrupamento de operações transacionais em estruturas hierárquicas. O termo transações relaxada *(relaxed transactions)* é usado para indicar que o modelo transacional “relaxa” em alguma das propriedades ACID [RUS95].

Primeiramente é necessário que se apresente os modelos transacionais (estendidos e relaxados) existentes para depois introduzir-se os modelos de workflow que utilizam tais conceitos. A seguir é apresentado o resumo dos principais tipos de modelos transacionais estendidos e relaxados:

- ***Nested Transaction* [MOS85]*:*** apresenta um conjunto de subtransações que podem conter outras subtransações recursivamente de maneira a formar uma árvore de transações. Uma transação filho só pode iniciar depois que sua transação pai seja inicializada e a transação pai só termina se todos os seus filhos terminam. Se uma transação pai é abortada, então todos os seus filhos também o são *(roll back*). Esse tipo de transação provém isolamento completo em um nível global, o que permite aumentar a modularidade proporcionando um maior grau de concorrência intratransacional do que as transações tradicionais.
- ***Open Nested Transactions* [WEIK92]:** esses modelos relaxam os requisitos de isolamento pelo resultado do *commit* de uma subtransação que é visível para as outras transações que estiverem sendo executadas de maneira concorrente. Eles também utilizam relações de precedência entre as transações ao invés de um esquema rígido de serialização, o que permite focalizar o "relaxamento" em requisitos de atomicidade ou isolamento. A maioria desses modelos utiliza alguma forma de compensação, pois uma subtransação pode efetuar o commit e liberar os recursos bloqueados antes que a transação global seja completada, assim se a transação global posteriormente abortar após o *commit* de sua subtransação, o efeito da subtransação deve ser desfeito executando uma subtransação de compensação *(compensating subtransactions)*. Há também as compensações horizontais *(horizon of compensation)* propostas por [KRY92] onde uma transação filho pode ser compensada somente se a transação pai ainda não resultou em *commit*. Uma vez que a transação pai tenha confirmado o *commit*, a única maneira de desfazer os effeitos da transação filho é compensando por inteiro a transação pai.
- **Modelo SAGAS [GMS87]:** este modelo foi projetado para resolver problemas de transações com longo tempo de duração. O SAGAS consiste em um set de subtransações $T_1,...,T_n$ com uma ordem predefinida de execução juntamente com um conjunto de subtransações de compensação *(compensating subtransactions)* $CT_1,...,CT_{n-1}$ correspondente as transações $T_1,...,T_{n-1}$. Uma saga se completa com sucesso se todas suas subtransações $T_1,...,T_n$ forem *commited*. Se uma das subtransações falhar, por exemplo T_k, então as subtransações $T_1,...,T_{k-1}$ são desfeitas através da execução das subtransações de compensação $CT_{k-1},...,CT_1$. Este modelo "relaxa" nos requisitos de isolamento e aumenta a concorrência intertransacional. Em 1991 ainda foram propostas extensões neste modelo por Garcia-Molina para o SAGAS em paralelo e o SAGAS generalizado.

- ***Flexible Transactions*** **[RUS92][ELM90]:** foram propostas para atuar em ambientes *multidatabase*. Uma transação provém caminhos alternativos de execução. Por exemplo, se uma subtransação aborta, pode-se envocar uma outra subtransação como alternativa para que se consiga o *commit*. Assim uma transação flexível só ira retornar sucesso *(commit)* se todas as suas subtransações e suas alternativas tiverem sucesso. De acordo com [ZNB94] uma transação flexível é uma ordem parcial de subtransações.

3.4. Problema dos Modelos Transacionais no Contexto de Workflow

Não é apropriado aplicar diretamente a tecnologia de transações, que tem sido utilizada nos bancos de dados, em um sistema de workflow [DONG99]. Uma transação em um workflow possui características diferentes das dos bancos de dados para que se possa utilizar diretamente o modelo de transações ACID. Uma transação convencional é uma atividade orientada a dados, e um workflow é uma atividade orientada a processos [DONG99].

Um workflow ainda apresenta duas diferenças relevantes que são: o longo tempo de vida de suas transações e o aspecto cooperativo dos seus dados. Nos bancos de dados relacionais estes aspectos têm importância secundária, pois as transações são de curta duração e quando uma transação é executada vários recursos permanecem bloqueados até que ela seja concluída ou abortada. Tal cenário seria inviável no contexto de workflow, pois há processos concorrentes que necessitam de recursos compartilhados e bloquear tais recursos não seria aconselhável visto que uma transação pode demorar desde algumas horas até alguns meses. Nas aplicações de banco de dados tradicionais, o tempo de vida de uma transação é suficientemente pequeno para que, em condições normais de funcionamento, o nível de concorrência por recursos seja baixo, facilitando assim a modelagem de transações ACID. Para contornar este problema, vários modelos de transações foram propostos (ver item 3.3) de forma a relaxar algumas das exigências do modelo ACID e expandir sua funcionalidade para sistemas de workflow [RUS95].

Vale frisar que as tarefas humanas ou tarefas não transacionais, geralmente não possuem uma compensação computacional correspondente. A compensação é realizada manualmente através de intervenção humana utilizando reportes para o administrador do sistema ou para as pessoas responsáveis [DONG99].

Vários trabalhos foram feitos no sentido de trazer o conceito de transação para o modelo de workflow. Nos itens a seguir são comentadas de maneira resumida as principais soluções desenvolvidas nesta área de workflow transacional.

3.5. Soluções para Workflow Transacional

Uma gama de sistemas gerenciadores de workflow (WFMS) têm se tornado produtos comercias: *OPEN/Workflow* da *Wang Laboratories*, *ProcessIT* da *AT&T*, GIS da Fujitsu, *Action Workflow* da *Action Tecnologies*, *Xerox´s InConcert*, *IBM´s FlowMark* e muitos outros [ALON96]. Em suas concepções e projetos a maioria destes sistemas são ortogonais aos modelos transacionais avançados e aos workflows transacionais [ALON96].

A seguir é mostrado um resumo dos modelos e projetos propostos nesta área de workflow transacional. Foram escolhidos os modelos e projetos mais comumente citados nas fontes pesquisas nais quais se teve acesso (internet, artigos e livros).

3.5.1 The WIDE Project

O projeto WIDE *(Workflow on Intelligent and Distributed database Environment)* propõe um modelo conceitual rico, incluido um modelo organizacional como base para tarefas elaboração de tarefas com recursos avançados para exeções *(exception handling)*, juntamente com conceitos de multitarefa e supertareras[5] para a modularização do workflow, e integração da semântica para transações extendidas [CAS99].

O *Global Transaction Manager* – GTM é o módulo responsável pelo gerenciamento de transações no nível de workflow. Especificamente, o GTM controla transações "relaxadas" utilizando um mecanismo de compensação baseado no modelo transacional SAGA de [GMS87]. O GTM utiliza o módulo LTM *(Local Transaction Manager)* para gerenciamento de baixo nível das transações. O GTM é independente do sistema de banco de dados utilizado e é localizado em uma das camadas *(layers)* do servidor WIDE [CAS99].

O *Local Transaction Manager* – LTM é o módulo que faz o gerenciamento de transações no nível de uma *supertask*. Ele controla o isolamento e atomicidade baseando-se no conceito de transações hierárquicas. O LTM utiliza mecanismos transacionais de banco de dados e está localizado na camada BAL *(Basic Access Layer)* da interface WIDE.

A arquitetura do sistema WIDE é mostrada na figura **Figura 3.1**. Os servidores WFMS do WIDE são baseados no sistema FORO que provém funcionalidades para executar localmente as tarefas de acordo com os processos de WF armazenado nos servidores, incluindo também suporte para exeções. Os servidores WIDE gerenciam também a interação com outros servidores (para distribuição de tarefas, e execução remota), e o gerenciamento global de transações para assegurar o gerenciamento transacional no nível de workflow.

[5] Uma supertarefa *(supertask)* é composta por um conjunto de tarefas correlatas que podem ser consideradas como um item de trabalho em um nível mais alto de abstração.

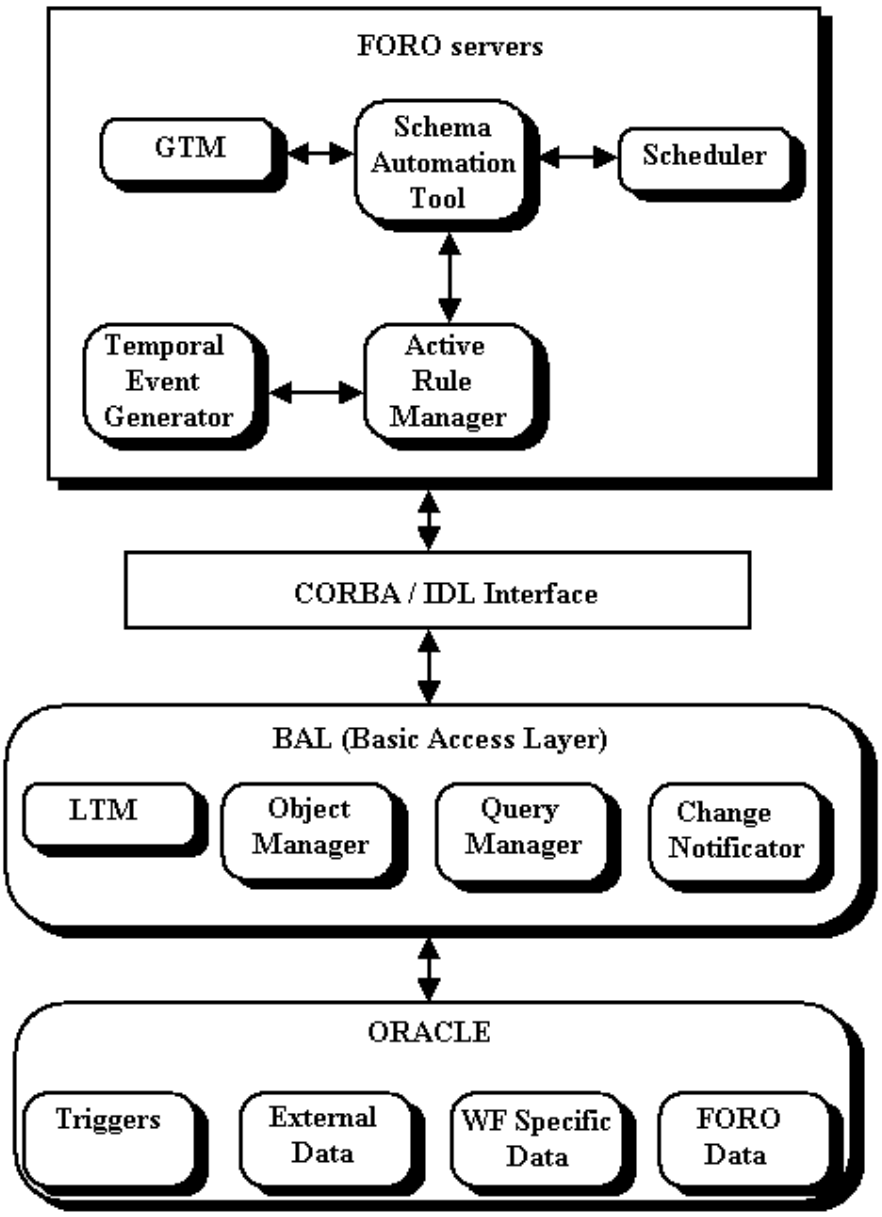

Figura 3.1 – Arquitetura WIDE e seus módulos [CAS99]

3.5.2 Modelo DTWS

Esse modelo proposto por [DONG99] serve para designar e implementar sistemas de workflow transacionais distribuídos que são chamados de DTWS *(Distributed Transactional Workflow Systems)*. Esse modelo foi projetado para alcançar quatro objetivos principais:

- **Disponibilidade:** pela utilização de múltiplos servidores que continuam atuando e garantindo os serviços mesmo que algum servidor tenha problemas;
- **Escalabilidade:** implica que novas funções ou servidores podem ser facilmente adicionados ao sistema;
- **Confiabilidade:** pelas facilidades oferecidas para definir processos com recursos adicionais de compensação em caso de falhas e caminhos alternativos de execução;
- **Reconfiguração dinâmica:** que é permitida pelo sistema.

No DTWS há quatro tipos diferentes de tarefas para situações normais e dois tipos de tarefas para situações anormais de erro. As tarefas seguintes tarefas fazem parte do modelo:

- **Tarefas Normais**
  - ***Human Task****:* tarefa que requer intervenção humana e é manipulada pelo gerenciador de *worklist (worklist handler)*;
  - ***Transactional Task****:* compreende todas as tarefas que podem ser compensadas tanto por uma aplicação de banco de dados quanto por uma aplicação qualquer;
  - ***Non-transactional Task***: esse tipo de tarefa pode acessar dados controlados por um gerenciador de recursos que não tem propriedades transacionais como um, por exemplo, um sistema de arquivos. Geralmente esse tipo não pode ser compensado por falhas.
  - ***Compound Task***: esse tipo é utilizado para modelar uma atividade com múltiplas tarefas correspondentes.
- **Tarefas Anormais**
  - ***Alternative Task***: essa tarefa é utilizada para gerenciar falha lógica ou exceções de tarefas automáticas, mesmo que seja uma tarefa transacional ou não transacional;
  - ***Compensating Task***: é o tipo de tarefa utilizado para desfazer os efeitos de uma tarefa incompleta decorrente de uma falha do sistema. Se a tarefa for do tipo *transactional task* é necessário que a sua *compensating task* seja sempre definida.

O DTWS foi desenvolvido no ambiente CORBA utilizando WTS *(Workflow Transactios Services)* que são uma extensão especialmente desenvolvida para sistemas de workflow. Os componentes do DTWS são o TMIF *(Task Managing Instance Factory)*, GTMIG *(Global Task Managing Instance Generator),*Simulador, Construtor de processos, Serviços de Administração e Monitoramento, e *worlist handler*. A **Figura 3.2** ilustra a arquitetura e os componentes de um DTWS.

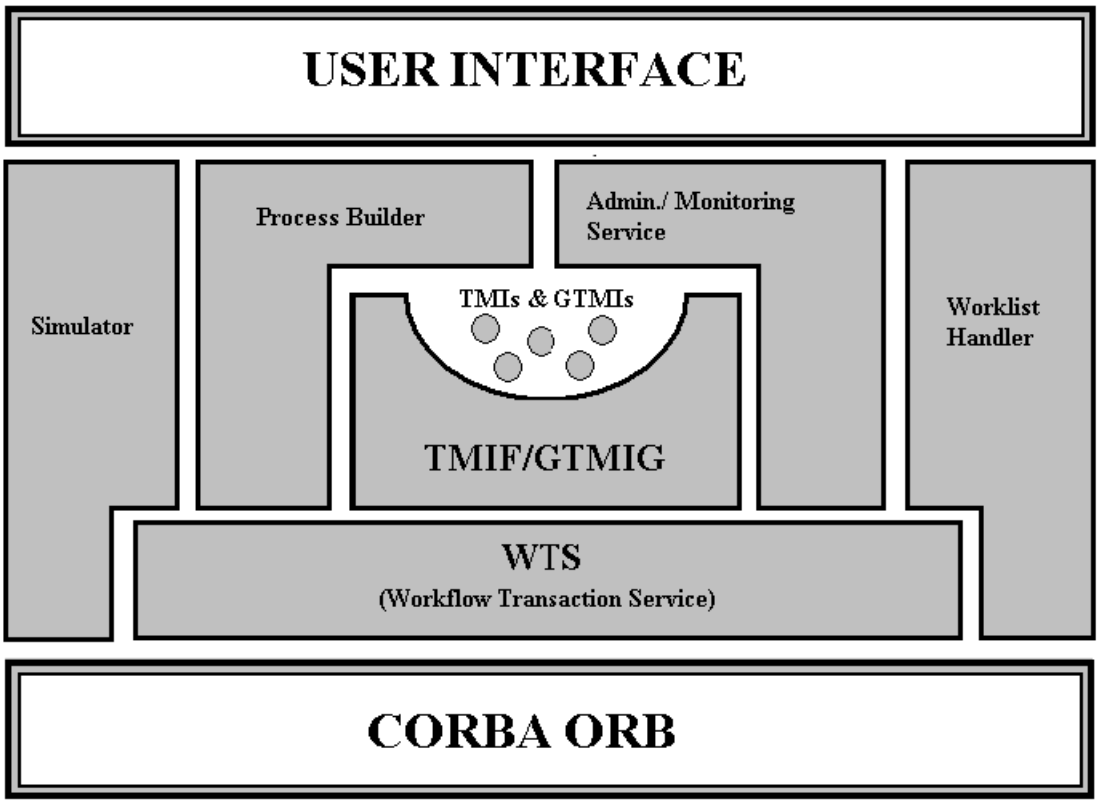

Figura 3.2 – Arquitetra do sistema DTWS [DONG99]

3.5.3 EXOTICA Project

Este modelo utiliza o conceito de transações flexíveis *(Flexible transactions)* juntamente com o modelo linear SAGA para modelar um workflow de características transacionais (ver item **3.3**). O EXOTICA segue o modelo de descrição e funcionalidade sugerido pela WFMC *(Workflow Management Coalition).*Para maiores explicações consulte o item **2.1**e **2.2**. Para a modelagen de processos é utilizado o *FlowMark* que é uma ferramenta de workflow da IBM é que se aproxima bastante do modelo de referência da WFMC. Este modelo conta com um módulo *middleware* chamado *EXOTICA/FMTM* que atua como um pré-processador que converte as especificações de modelos transacionais em processos de workflow. A **Figura 3.3** ilustra uma modelagem de uma transação SAGA em um processo do *FlowMark.*

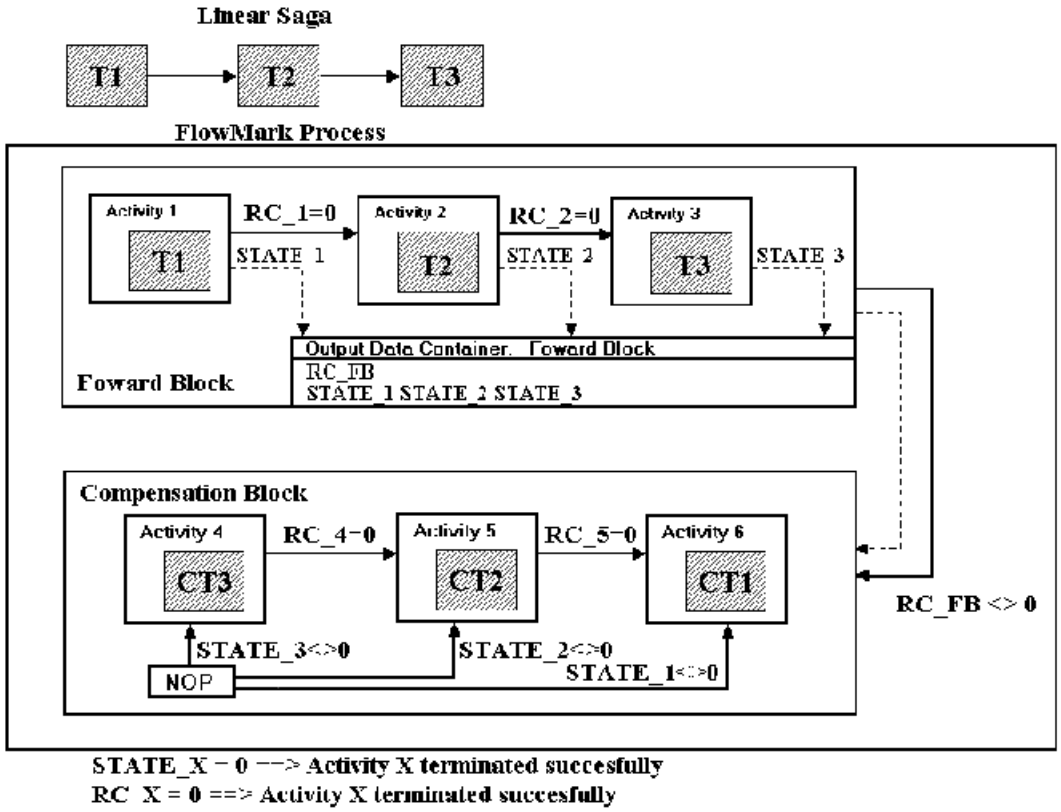

Figura 3.3 – Modelagem de uma transação SAGA utilizando *FlowMark* [ALON96]

3.5.4 A General Model for Nested Transactional Workflows

Este modelo é proposto no trabalho de [KUO96]. Esse modelo "relaxa" a atomicidade das transações especificando quais tarefas são não criticas e que não requerem compensação. O workflow efetua um *commit*[6] se todas as suas tarefas críticas efetuarem o *commit* e também se todas as suas tarefas não críticas efetuarem um *commit* ou um *abor*[7]*t*.

Cada tarefa associada ao modelo apresenta uma máquina de estados finita com um conjunto visível de estados. Todas as tarefas apresentam um estado inicial, um conjunto de estados intermediários, um conjunto de estados para *commit* e outro para *abort.*A razão de múltipos estados de *comit* e *abort* é para se modelar planos de alternativa e contingência mais facilmente. As tarefas possuem duas propriedades essênciais que são forçabilidade (*forcibility*) e compensabilidade (*compensatability*). Uma tarefa ou uma transação é forçavel se o sistema assegurar que na presença de falhas, essa tarefa ou essa transação terá a possibilidade de sucesso. Uma tarefa ou transação é compensável se ela pode ser logicamente desfeita depois que um *commit* é executado. Este modelo apresenta uma coleção de cinco construções básicas para se implementar um workflow transacional:

- **Ordenação:** especifica a ordem em que as tarefas podem ser executadas;
- **Contingência:** um plano de contingência é composto por duas tarefas, uma principal e outra como sendo a alternativa. Assim se a tarefa principal falhar a tarefa alternativa é acionada e podendo ainda obter sucesso na transação.
- **Alternativa:** uma alternativa é composta por um número arbitrário de tarefas, que especifica um número de caminhos possíveis para se alcançar um objetivo. Diferentemente de um plano de contingência uma alternativa não necessita que uma tarefa aborte para que outra seja acionada, há o critério de livre escolha para que o sistema escolha quais tarefas executar.
- **Condicionais:** esse tipo de construção possibilita a implementação de *if statements*. São compostos por uma condição e duas tarefas, e durante a execução o sistema irá decidir qual das duas tarefas será executada.
- **Iterações:** uma iteração possui uma condição e uma tarefa. Enquanto a condição for verdadeira a tarefa é continuamente envocada. É semelhante à estrutura *for* encontrada nas linguagens de programação.

---

[6] Uma operação de *commit* indica o término de uma transação bem-sucedida. Ela indica que todas as atualizações feitas podem agora ser validadas, ou seja, se tornarem permanentes [MAL03].

[7] Uma operação de *abort* indica que a transação terminou sem sucesso, de forma que quaisquer alterações ou efeitos que a transação possa ter aplicado no sistema devem ser desfeitos.

A definição formal de uma tarefa é dada na **Figura 3.4**. O estado inicial representa o estado da tarefa antes de sua execução. Os estados de *commit* e *abort* representam se a tarefa foi comprida com sucesso ou não. Uma transição é composta por um estado antigo (*old state*) uma ação (*action*) e um novo estado (*new state*). Toda tarefa deve apresentar pelo menos uma transição para o estado inicial (*intial state*), uma para o estado de *commit* ou uma para o estado de *abort*.

```
Task_name_t: string;

task_state_t: string;

action_t: string;

transition_t = record
                old_state : task_state_t;
                action: action_t;
                new_state: task_state_t;
end;

task_t = record
               name: task_name_t;
               initial : task_state_t;
              intermediate: set of task_state_t;
              commit: set of task_state_t;
              abort: set of task_state_t;
             transitions: set of transition_t;
end;
```

Figura 3.4 – Modelagem de Tarefas [KUO96]

A **Figura 3.5** apresenta a descrição formal de um workflow transacional. Ele é representado por um nome, um conjunto de tarefas e um conjunto de estados ilegais (*illegal states*).O conjunto de estados ilegais é composto pelo conjunto de tarefas e seus estados e auxilia a construção da ordem de execução entre as tarefas. Maiores detalhes estão disponíveis em [KUO96].

```
twf_state_t : set of <task_name_t, task_state_t>;

twf_t = record
                name: task_state_t;
                tasks: set of task_t;
                illegal: set of twf_state_t;
end;
```

Figura 3.5 – Workflows Transacionais [KUO96].

3.5.5 CrossFlow

O CrossFlow é um projeto que usa tecnologia de workflow para ativar processos de negócios entre organizações consumidoras e organizações fornecedoras (ou provedoras) [HOF00]. O CrossFlow é um projeto de parceria fundado por organizações da União Européia e o Departamento Federal de Educação e Ciência Suíço.

Processos de negócios são chamados *cross-organizational* se há a possibilidade que pelo menos uma de suas atividades está ligada a uma organização diferente [KLI00]. O projeto do CrossFlow propoe extensões na tecnologia de workflow tradicional para possibilitar os processos de negócios *cross-organizational*.

O conceito central na abordagem proposta no CrossFlow é a noção de workflows *cross-organizacional* através do estabelecimento de um contrato. O contrato define o relacionamento bilateral entre as organizações em termos de responsabilidade, cobrança e confiabilidade. Um contrato abrange quaisquer ações realizadas por uma das partes. O consumidor do serviço pode utilizar diversos provedores de serviços externos para realizar várias partes do seu processo de negócio.

O contrato especifica todos os detalhes de um relacionamento em particular entre um consumidor e um provedor, ambas as partes devem possuir uma visão compartilhada do processo através de um processo interno definido no contrato. No lado do consumidor, o seviço a ser contratado é descrito no contrato como parte de um processo de negócios que é especificado de acordo com as políticas e objetivos deste consumidor. No lado do provedor, o serviço oferecido é descrito no contrato de acordo com a especificação de workflow adaptada para o sistema e infraestrutura do requisitante. O contrato contém também informações sobre decisões, qualidade de serviço (QoS – *Quality of Service*) e outras informações adicionais desenvolvidas para o projeto do CrossFlow.

Para se realizar um contrato um provedor de serviço deve oferecer seus serviços através de um gerenciador de contratos (*Contract Manager*) enviando a sua *template* de contrato[8] para uma entidade anunciadora (*Trader*). Quando um consumidor a um necessita de um serviço externo este procura os serviços através das *templates* de Contratos presentes no *Trader*. Quando os serviços oferecidos pelo provedor satisfazem as necessidades do consumidor, o contrato eletrônico é realizado através do preenchimento das informações do *template* de contrato do provedor. Baseado nas especificações de contrato monta-se uma arquiterura dinâmica de interoperação. A arquitetura possui *Gateways* de *Proxy* que

[8] Uma template de contrato é um contrato estruturado que possui alguns campos definidos para armazenar as informações específicas que serão preenchidas pelos participantes do contrato [HOF99].

controlam toda a comunicação e os suportes de serviço (*Support Services*) que são utilizados para as operações de cooperação. Após a conclusão do processo de negócio, todos os módulos criados dinamicamente podem ser liquidados. A **Figura 3.6** ilustra a arquitetura básica do CrossFlow [JAC01].

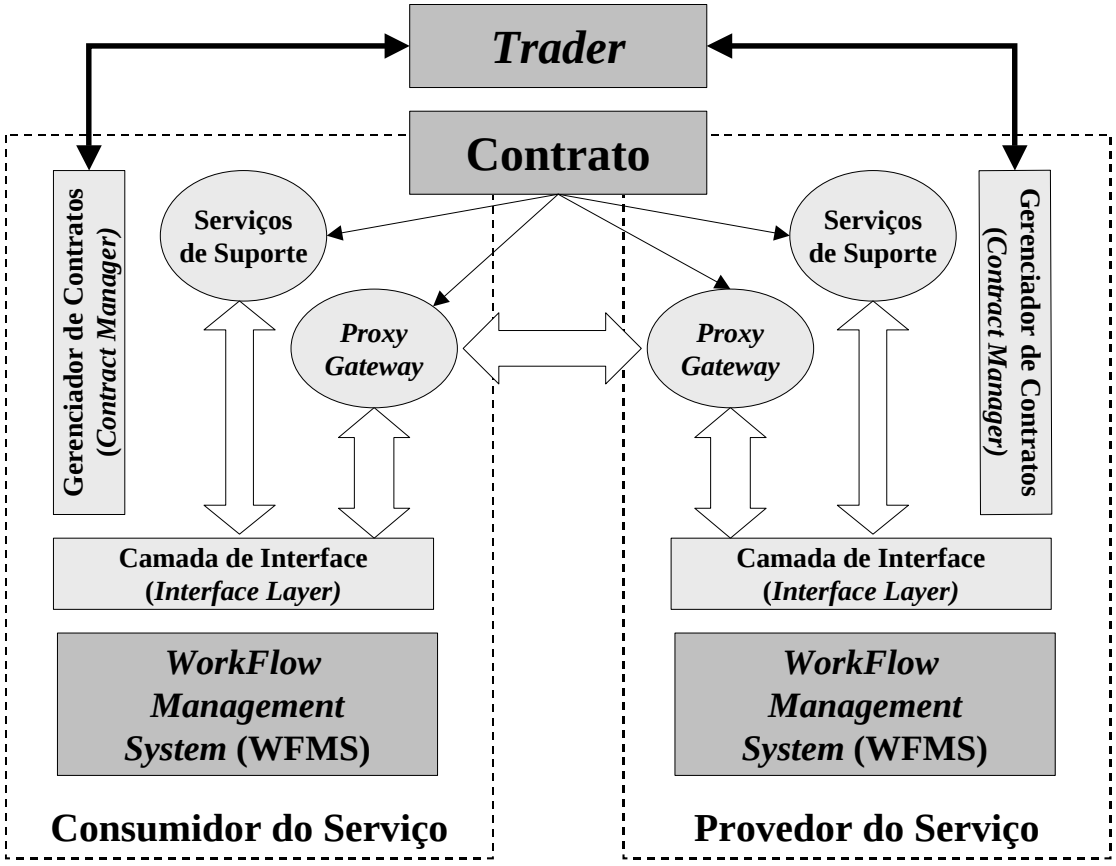

Figura 3.6 – Arquitetura do CrossFlow [JAC01]

O CrossFlow oferece ao consumidor uma uma série de primitivas chamadas de *Process Control Primitives* (PCP´s) que permitem que a organização consumidora tenha um certo controle sobre o processo de workflow que está sendo executado no provedor. As primitivas oferecidas são:

- **Stop:** a execução do processo no provedor é paralizada;
- **Continue:** depois de paralizar a execução pode ser necessário que o processo seja reativado através desta primitiva;
- **Abort:** utilizado quando a execução do processo não deve continuar e este é finalizado, mas a decisão se o processo irá ou não ser compensado fica a cargo do provedor;
- **Change X,Y:** primitiva utilizada para mudar o valor da variável X e atribuindo a ela o valor Y;

- **Rollback:** operação utilizada quando deseja-se compensar parte ou todo o processo de workflow do provedor. A compensação é realizada através de um conjunto de atividades que desfazem as alterações que foram efetuadas pela a execução do processo.

O CrossFlow adota o modelo SAGAS (ver item 3.3) com algumas modificações. Essas modificações estendem a transação SAGAS de modo a oferecer o conceito de *safe-points*, o que permite um *rollback* parcial no processo de workflow [VON99]. Um *safe-point* permite ao projetista indicar um ponto seguro no processo de workflow onde haja a execução do *rollback* deste ponto em diante. Isso significa que cada transação sagas executada antes do *safe-point* continuará válida, mas que todas as transações sagas executadas depois do *safe-point* serão compensadas e o processo será reiniciado na primeira transação (atividade) que vem após ao *safe-point*. Isso possibilita que em caso de falha em vez de desfazer todo o processo de workflow, seja desfeita somente a parte necessária relaxando assim os requisitos de atomicidade originais do modelo SAGAS.

A **Figura 3.7** apresenta um processo de workflow dividido em transações SAGAS com uma transação de *safe-point*. Cada transação é equivalente a uma atividade do processo e cada atividade tem também sua atividade de compensação equivalente que faz parte da própria especificação do workflow. No exemplo da Figura 3.7 é mostrado um processo de workflow que por alguma razão é abortado na transação "E" e precisa ser compensado (*rolled back*) para que o processo volte para um estado consistente. No caso de ser efetuado um *rollback* parcial o WFMS executará a compensação até o a transação "B" que é o *safe-point* deste exemplo. Isso implica que serão executadas as atividades de compensação "D^{-1}" e "C^{-1}" já que a atividade "E" não necessita de compensação visto que a mesma não estava em um estado de *commit* quando ocorreu a falha.

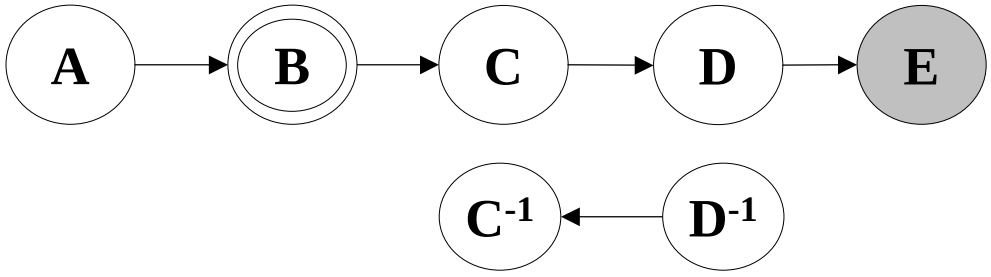

Figura 3.7 – Exemplo de uma Transação no CrossFlow

3.6. Análise das Soluções Existentes

Para realizar o estudo e análise das soluções existentes achou-se mais conveniente montar-se uma tabela de comparação. A tabela é montanda da seguinte maneira: as colunas apresentam os sistemas de workflow transacional a serem comparados, e as linhas apresentam as características/funcionalidades que serão avaliadas. Quando um workflow apresentar a característica/funcionalidade a qual se refere a linha haverá um "X" marcando a célula correspondente da tabela. A seguir é descrito cada uma das características/funcionalidades que foram avaliadas na montagem da tabela dos comparativos:

- **Interface de Interoperabilidade padronizada:** essa característica indica se o sistema de workflow possui a sua interface de interoperabilidade baseada em algum padrão aberto prosposto por entidades normatizadoras (ex: WFMC, OASIS, IEEE, etc);
- **Interface de Interoperação compatível com WFMC:** indica se o sistema de workflow tem a sua interface de interoperação baseada em algum padrão proposto pelo WFMC (ex: Mime Binding, Wf-XML, XML-HTTP Binding);
- **Independência de Fornecedor:** se o workflow apresentar tal característica significa que ele pode interoperar com qualquer ferramenta de workflow disponível no mercado independente do seu fornecedor (ex: IBM, Microsoft, ORACLE, etc);
- **Interoperação via Web:** se o sistema de workflow realizar a sua interoperação utilizado a internet como meio de comunicação esta funcionalidade estára marcada na tabela;
- **Utilização de Web Services:** se o sistema de workflow apresentar algum recurso de comunicação que faça a utilização da facilidade da tecnologia de web services esta opção estará marcada na tabela;
- **Escalabilidade:** característica que possibilita um sistema ser facilmente expansível a medida que o seu volume de operações cresça;
- **Baixo custo de adaptação:** indica se um sistema de workflow existente necessitar de ser adaptado para o workflow transacional correspondente haverá um baixo custo para que essa adaptação aconteça.

A seguir é mostrado na **Tabela 3.1** o comparativo das soluções existentes. Vale destacar que as análises foram realizadas a partir das informações recolhidas nas referências bibliográficas deste trabalho, portanto é possível que algumas delas estejam incompletas.

	Wide Project	Modelo DTWS	EXOTICA Project	General Model	CrossFlow
Interface de Interoperabilidade padronizada					
Interface de Interoperação compatível com WFMC					
Independência de Fornecedor				X	X
Interoperação via Web	X	X	X		X
Utilização de Web Services					
Escalabilidade	X	X	X		X
Baixo custo de adaptação					X

Tabela 3.1 – Tabela de Análise das Soluções de Workflow Transacional

3.7. Conclusão

Enquanto que um gerenciamento para falhas mais sofisticado tem sido muito discutido nos avançados modelos transacionais na literatura, poucas dessas técnicas estão sendo realmente utilizadas em grandes sistemas da indústria e comércio [ALON96].

Deve-se levar em conta que sistemas de workflow apresentam certas particularidades que dificultam a adaptação de modelos transacionais. As duas maiores variáveis a se pesar são que sistemas de workflow possuem um longo tempo de vida em suas transações, e que eles manipulam dados cooperativos. Esses problemas são contornados com o surgimento de propostas de modelos transacionais “relaxados”, que possibilitam a garantia de algumas condições ACID sobre certas circunstâncias.

Vale frisar que nenhuma das soluções de workflow transacional propostas apresenta uma solução ideal para o problema. Portanto esta é uma área que necessita ainda de muita pesquisa e desenvolvimento para se buscar uma solução ideal que garanta as propriedades

ACID e que consiga satisfazer as exigências de semântica dos sistemas de workflow sem prejudicar sua eficiência e flexibilidade.

Pode-se perceber também que nenhum dos workflows transacionais pesquisados apresentou uma interface de interoperação padronizada ou compatível com os padrões estabelecidos pelo principal orgão normatizador de workflow que é o WFMC.

Outro fator preponderante é o fato de nenhum dos sistemas de workflow transacional utiliza os recursos e flexibilidade oferecidos pelos web services para interoperação de aplicações via internet.

A proposta deste trabalho visa preencher essas lacunas oferecendo também uma solução que possua escalabilidade e que seja de baixo custo de adaptação em um sistema de workflow.

3.8. Resumo

O capítulo em questão apresentou os tópicos básicos relacionados com o conceito de workflow transacional.

Nos itens 3.1 e 3.2 apresentou-se uma introdução aos conceitos de transação e workflow transacional.

No item 3.3 apresentou-se os modelos mais populares para transações relaxadas e extendidas.

No item 3.4 comentou-se sobre as maiores dificuldades em se adaptar um modelo de transacional para o domínio de um workflow.

No item 3.5 apresenta um resumo dos modelos e projetos propostos na área de workflow transacional.

Para finalizar o item 3.6 apresenta uma análise das soluções existentes na área de workflow transacional.

Capítulo 4

WEB SERVICES E WORKFLOW

4.1. Introdução

A internet começou com o suporte para interação humana através de textos e gráficos. Diariamente pessoas utilizam a internet para verificar as ações do mercado financeiro, comprar produtos, ou ler as últimas notícias. Esse nível de interação satisfaz razoavelmente muitos propósitos de interação humana. Porém esse nível de interação baseado em textos e gráficos não oferece um suporte muito adequado para interação de softwares, especialmente quando se trata da troca de informação entre um ou mais sistemas através da internet.

Web services surgiram como uma proposta para otimizar o uso da internet através da comunicação entre programas *(program-to-program communication)* [NEW02]. Através da expansão e da utilização de web services, as aplicações localizadas em vários sites distribuídos ao longo da internet poderão ser diretamente integradas e interconectadas como se fossem parte de um único e imenso sistema de informação [NEW02].

Este capítulo apresenta o conceito de web services e os trabalhos existentes na área que enfocam o problema de workflow utilizando a tecnologia de web services.

4.2. Web Services

Assim como as ferrovias tiveram efeitos na economia dos transportes mundiais, os Web Services estão mudando a fundamentação do comércio pela Internet *(web commerce)*. Os Web Services conectam programas através de pontos distantes do globo, transportando grande quantidade de dados mais eficientemente e a um custo mais barato como nunca antes visto. O resultado é a rapidez e melhora da comunicação produtiva para os negócios e consumidores [NEW02].

Web services são basicamente aplicações XML *(Extensible Markup Language)* mapeadas para programas, objetos, banco de dados ou para funções de negócios [NEW02]. Utilizando um documento XML criado na forma de uma mensagem, um programa envia um pedido *(request)* para um web service localizado na internet, e, opcionalmente pode receber uma resposta *(reply)* também na forma de um documento XML enviado como uma mensagem. A padronização de web services define o formato dessa mensagem, especificando a interface para qual a mensagem é enviada, descrevendo as convenções para o mapeamento do conteúdo (de entrada e saída), e definindo os mecanismos para publicar e procurar as interfaces web services ao longo da internet.

Esta tecnologia pode ser utilizada de várias maneiras. Web services podem rodar em aplicações *desktop* e gerenciar clientes para acessar aplicações de reservas e pedidos na internet. Web services também podem ser utilizados para integração *business-to-business* (B2B), conectando aplicações de várias organizações pertencentes a uma mesma cadeia de suprimentos *(supply chain)* [NEW02]. Web services podem ainda resolver um problema mais amplo de integração de aplicações empresariais *(enterprise application integration – EAI)*, conectando várias aplicações de uma organização a outras aplicações sejam estas localizadas dentro ou fora do *firewall*.

O pacote de web services *(web services wrap)* apresenta a padronização para interfaceamento com sistemas *back-end*, como banco de dados, NET, J2EE, CORBA, e outros. A **Figura 4.1** ilusra o papel de web services como interface de *back-end systems*.

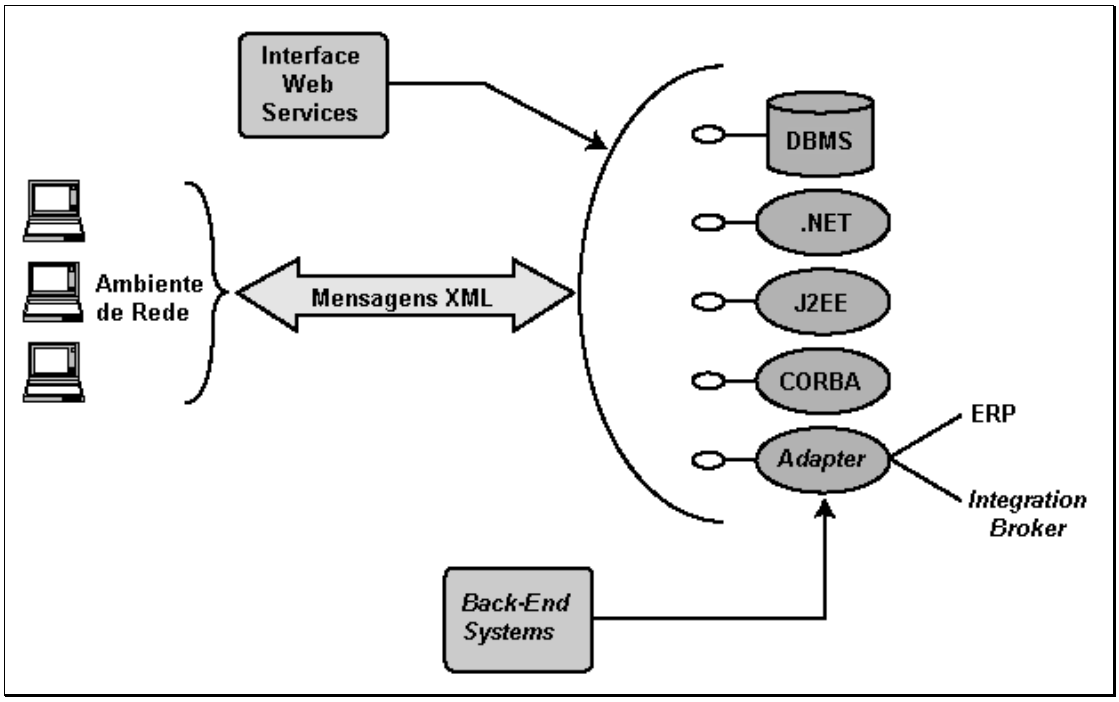

Figura 4.1 – Interface Web services com *back-end systems* [NEW02]

A interface de web services recebe uma mensagem XML padronizada da rede *(network environment)*, transformando os dados XML em um formato que seja entendível por dado sistema *back-end*, e opcionalmente, retorna uma mensagem de resposta. O software de implentação de web services pode ser criado utilizando-se qualquer linguagem de programação, sistema operacional, ou *middleware system*.

Atualmente as tecnologias de internet que tiveram sucesso, o tiveram porque elas definem um nível de abstração suficientemente grande para permitir a compatibilidade com qualquer sistema operacional, hardware ou software. As tecnologias baseadas em web services aproveitam-se deste nível de abstração incluindo também a semântica da informação associada com os dados. Assim web services definem não somente os dados, mas também como processa-los e mapeá-los para entrada e saída nas aplicações de software.

Web services tem a finalidade de estabelecer a interação humana através da internet com uma proposta completamente nova. Interações baseadas em software vão automaticamente realizar operações que antes eram efetuadas com intervenção manual, como por exemplo: procurar bens de consumo ao melhor preço possível, marcar passagens de viagem e reservar mesas em um restauramte, simplificando, qualquer processo de negócio que envolva operações de procura, aquisição e entrega. Um exemplo de um web service muito simples pode consistir, numa única operação que receba a sigla de uma empresa e que devolve o valor da sua cotação corrente na bolsa.

4.2.1 Tecnologias de Web Services

Web services requerem várias tecnologias baseadas em XML para transportar e transformar dados para dentro e fora dos programas e banco de dados envolvidos. Dentre as principais pode-se citar:

- **XML (Extensible Markup Language):** compreende a fundação básica na qual web services são elaborados. XML é a linguagem básica para definição de dados e como processá-los. Essa linguagem ainda faz parte da família de especificações do *World Wide Web Consortium* (W3C);
- **WSDL (Web Services Description Language):** linguagem baseada em XML que define as interfaces web services, dados, tipos de mensagens, padrões de interação, e mapeamento de protocolos;
- **SOAP (Simple Object Access Protocol):** coleção de tecnologias baseadas em XML que definem um envelope para comunicação de web services. Provém um formato de serialização para transportar documentos XML através da rede e também uma convenção para representação de interações RPC (*Remote Procedure Call)*;
- **UDDI (Universal Description, Discovery and Integration):** é um mecanismo no qual os web services são registrados e descobertos. É utilizado para armazenar e categorizar informações de negócios e para fornecer os "ponteiros" para as interfaces web services.

A **Figura 4.2** ilustra as cinco camadas tecnológicas que compoem a plataforma web services.

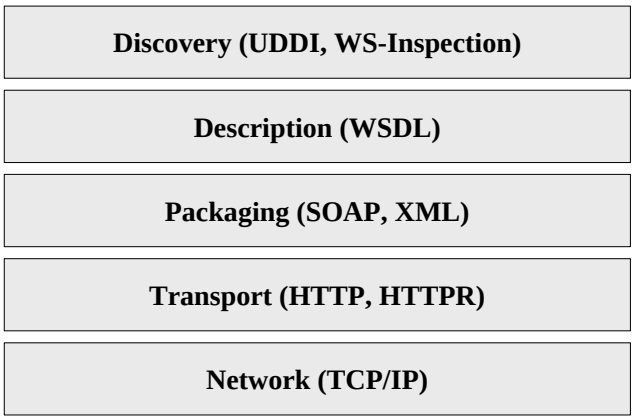

Figura 4.2 – Pilha das tecnologias web services [MTR02]

4.2.2 Funcionamento dos Web Services

Uma vez definidos os dados nas mensagens (XML), descrito os serviços que vão receber e processar as mensagens (WSDL), prover os mecanismos para enviar e receber essas mensagens (SOAP), necessita-se então de uma maneira de publicar os serviços que serão oferecidos bem como localizar outros serviços que poderão ser úteis (UDDI).

A **Figura 4.3** ilustra como funciona o sistema para publicar, localizar e interagir com os serviços de web services.

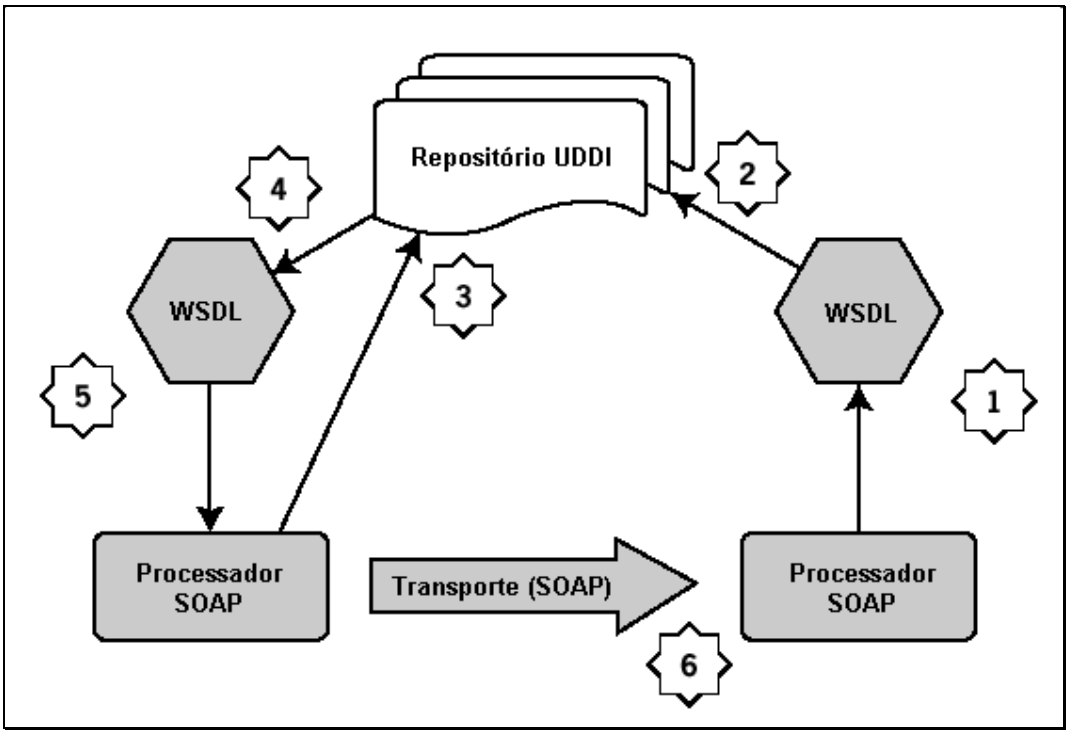

Figura 4.3 – Publicando, localizando e interagindo com web services [NEW02]

1. O serviço de negócio gera inicialmente um arquivo WSDL que descreve o web service suportado segundo o seu processador SOAP *(SOAP Processor)*;
2. Esse arquivo WSDL é registrado no repositório UDDI *(UDDI Repository)* através de APIs *(Application Programming Interfaces)* UDDI;
3. Depois de submeter os dados do web service para o registro, há outras informações de contrato, que informam a URL que aponta para o servidor SOAP deste arquivo WSDL que descreve o web service. Assim em (3) outro processador SOAP de negócios pode consultar o repositório UDDI;
4. Uma vez encontrado o serviço desejado obtém-se o esquema WSDL deste serviço;
5. De posse do esquema o SOAP processor do cliente pode interagir diretamente com o SOAP processor do serviço oferecido;

6. Essa interação é realizada através de mensagens apropriadamente formatadas que enviam operações específicas em cima de um protocolo previamente identicado.

4.3. Web Services e Workflow

A **Figura 4.2** apresentou a pilha de todas as tecnologias envolvidas no contexto de web services. Diferentes organizações têm trabalhado em uma última camada *(layer)* para esta pilha, com a finalidade de tratar workflow entre web services [DUI01]. Grandes empresas como IBM e Microsoft desenvolveram suas soluções com seus prórprios protocolos XML com suporte para workflow (WSFL da IBM e XLANG da Microsoft). Antes de introduzir WSFL e XLANG será mostrado um estudo de caso que descreve um exemplo de workflow entre diferentes web services que são fornecidos por vários parceiros externos. Depois de tal descrição serão apresentados os principais problemas para a integração de diferentes web services sobre um processo de workflow.

4.3.1 Estudo de Caso – Pizzaria do Futuro

Na pizzaria do futuro há dois tipos de funcionários: entregadores de pizza e os cozinheiros. Quando um entregador começa o seu turno ele se loga no pizza *server*. Ele faz isso utilizando o serviço de Autenticação (veja na **Figura 4.4**) e depois de ter suas credenciais checadas, o entregador de pizza é direcionado para um serviço de Perfil que define qual motocicleta que melhor se encaixa no seu perfil e atende os seus requisitos físicos. De posse de sua moto o entregador de pizza já está apto a atender os pedidos.

Depois de um dia duro de trabalho você quer comer uma pizza. Com o seu celular com suporte *WAP(Wireless Application Protocol)* você disca para a pizzaria, e depois de se identificar *(*Autenticação*)* o seu perfil *(*Perfil *Web Service)* determina os sabores de pizza que você mais gosta. Assim o catálogo de pizza lhe oferece somente quatro sabores; e antes de selecionar sua pizza favorita você deve concordar com o preço, e o seu cartão de débito (que está incluso no seu perfil) é validado pelo web service de Validação do Cartão. Se tudo estiver ok, sua conta bancária sofrerá um débito correspondente ao valor da pizza e este valor será creditado na conta bancária da pizzaria. Enquanto isso o cozinheiro é alertado do seu pedido que aparece no seu *browser*.

Depois desta transação o *web service GPS(Global Position System)* é chamado para fornecer sua localização através do seu celular. Suas coordenadas são passadas através do *web service* de Mapas, que processa as informações e envia um mapa com o caminho exato para o *PDA(Personal Digital Assistant)* do entregador de pizza. Então o entregador necessita somente pegar a pizza e entregar no local correspondente. Nesse meio tempo você recebe uma

mensagem SMS no seu celular avisando que a sua pizza será entregue dentro de 20 minutos. Simples.

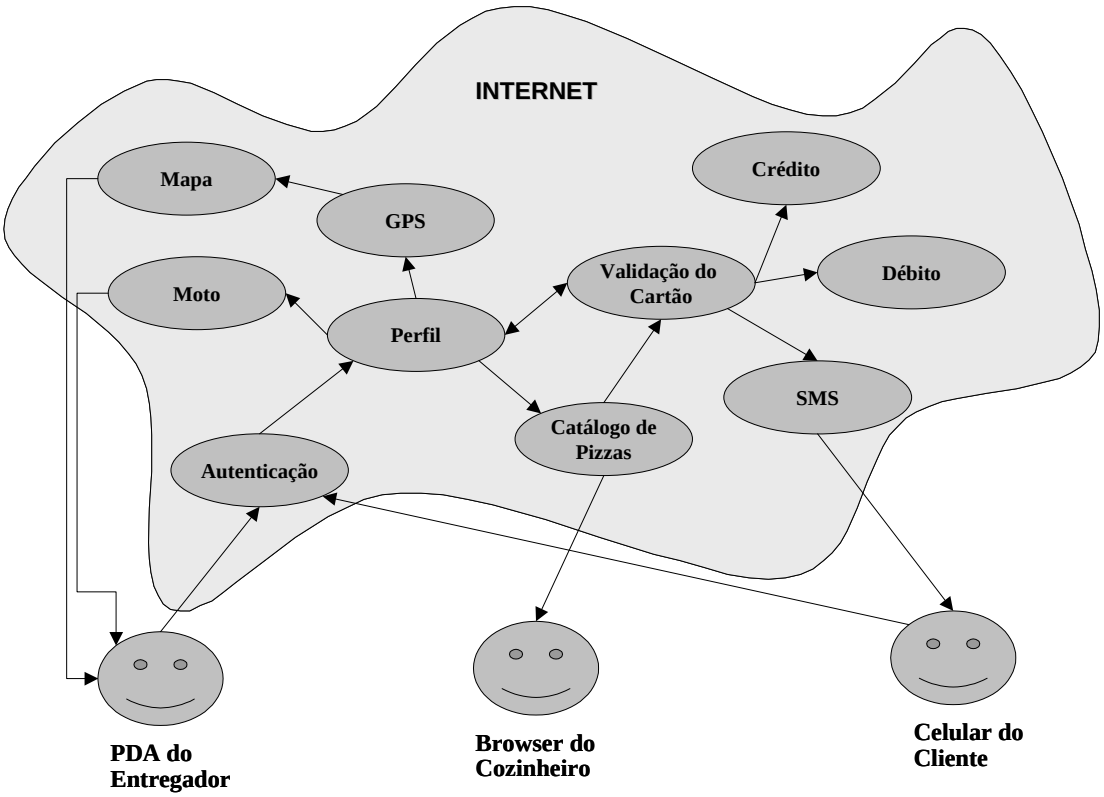

Figura 4.4 – Estudo de caso: pizzaria do futuro [DUI01]

Pode ser muito dispendioso que uma pizzaria do futuro desenvolva sozinha todos estes tipos de web services. Se ela tiver que desenvolver algum, possivelmente será o *web service* de Catálogo de Pizzas e o *web service* de Moto que são os mais específicos para o seu ramo de negócios. Portanto para que tal cenário seja viável é necessário que a pizzaria do futuro utilize outros web services disponíveis na internet.

4.3.2 O Problema da Integração

A pizzaria futurista terá a necessidade de integrar vários web services externos com o seu próprio processo de negócios. Durante esta integração ela deverá lidar com os seguintes problemas [DUI01]:

- **Processos com múltiplos parceiros *(Multiple Partner Processes)*:** quando há interação entre múltiplos parceiros deve-se levar em conta o tempo de processamento de cada web service. Se um web service leva muito tempo para processar um pedido, é necessário que se encontre outro parceiro com um web service mais rápido.

- **Manipulação de Transações *(Transaction Handling)*:** pode-se pegar como exemplo a validação do cartão. Depois de validar o cartão de débito do cliente o balanço entre as duas contas bancárias deve ser alterado. De uma conta deve-se debitar 20R$ correspondentes ao valor da pizza, e em outra conta deve-se creditar esses 20R$. Se alguma coisa der errado durante essa transação, por exemplo, o valor não foi debitado da conta do cliente, a transação necessita ser desfeita *(roll-back)* e nenhuma conta deve ser alterada, caso contrário, dinheiro será criado ou perdido.
- **Manipulação de Exeções *(Exception Handling)*:** se algum erro ocorrer durante o processo de negócio, o usuário final precisa ser notificado, assim é necessário que o processo de negócio saiba como lidar com as exeções.
- **Manipulação de Incompatibilidades *(Incompatibilities Handling)*:** até o momento há por cerca de 35 empresas fornecendo produtos que facilitam a mudança da aplicação tradicional em um web service. Todavia, cada produto apresenta sua própria implementação de especificação WSDL. Isso significa que haverá vários problemas de incompatibilidade durante a integração dos web services.
- **Dependência *(Dependency)*:** quando há a integração de web services de parceiros externos, o sistema se torna dependente. Por causa da natureza dos web services não se pode ter controle sob web services externos. Assim se um servidor de um parceiro externo cair, isso certamente afetará a cadeia de negócios do sistema. Quando há web services externos no sistema há sempre um elemento de confiança envolvido.
- **Flexibilidade *(Flexibility)*:** os negócios estão mudando rapidamente, logo, aplicações que utilizam web services necessitam ser flexíveis. Aplicações devem ser capazes de mudar para um outro web service que ofereça as mesmas funcionalidades, se o web service escolhido apresentar alguma deficiência ou problema.

O principal objetivo dos web services é que desenvolvedores de software possam facilmente integrar diferentes tipos de aplicações e serviços, sem ter que se preocupar em entender protocolos, interfaces, condições ambientais, etc [DUI01]. Portanto tem que haver também meios que aliviem os desenvolvedores de software em se preocupar com os problemas mencionados acima.

4.4. WSFL

WSFL é uma linguagem XML para a descrição de web services *compositions* [LEY01]. A WSFL considera dois tipos de composições *(compositions)* web services:

- ***Flow Models:*** conhecido como flow compositions, ou orquestração de web services, este modelo especifica uma composição de fluxo utilizando-se uma coleção de web services. Os Flow Models são utilizados para modelar processos de negócios ou worflows.
- ***Global Models:*** nesse caso não há especificação de seqüência e sim de como os web services compostos interagem entre si. As interações são modeladas através de links entre as interfaces web services.

A **Figura 4.5** apresenta um modelo de fluxo *(Flow Model)* chamado *totalSupplyFlow*. Note que no fluxo há uma seqüência de atividades: o processamento da ordem de compra *(processPurchaseOrder)* do fabricante deve preceder o pedido de entrega *(acceptShipmentRequest)* do transportador, porém o dinheiro pode ser aceito a qualquer hora *(processPayment)*. Todo o fluxo pode ser "empacotado" em um único web service com portas de saída chamado de *totalSupplyPT*.

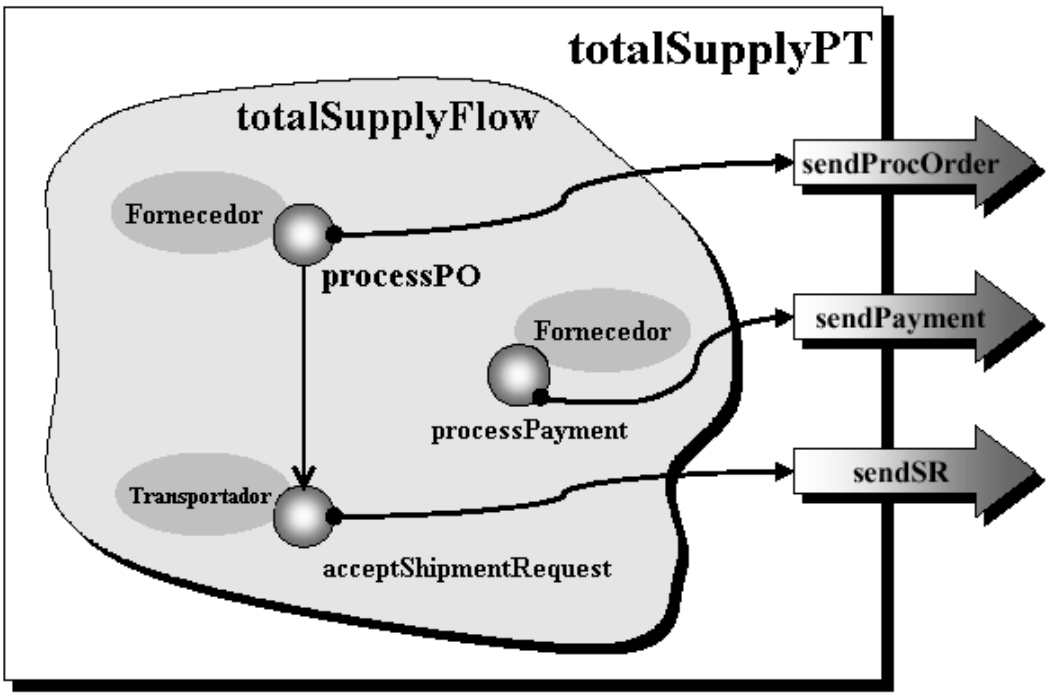

Figura 4.5 – Exemplo de um *Flow Model* [LEY01]

O modelo global *(Global Model)* apresenta os links e interações entre os diferentes *Flow Models* da cadeia de suprimentos *(supplyChain)*. A **Figura 4.6** apresenta um Exemplo de modelo global.

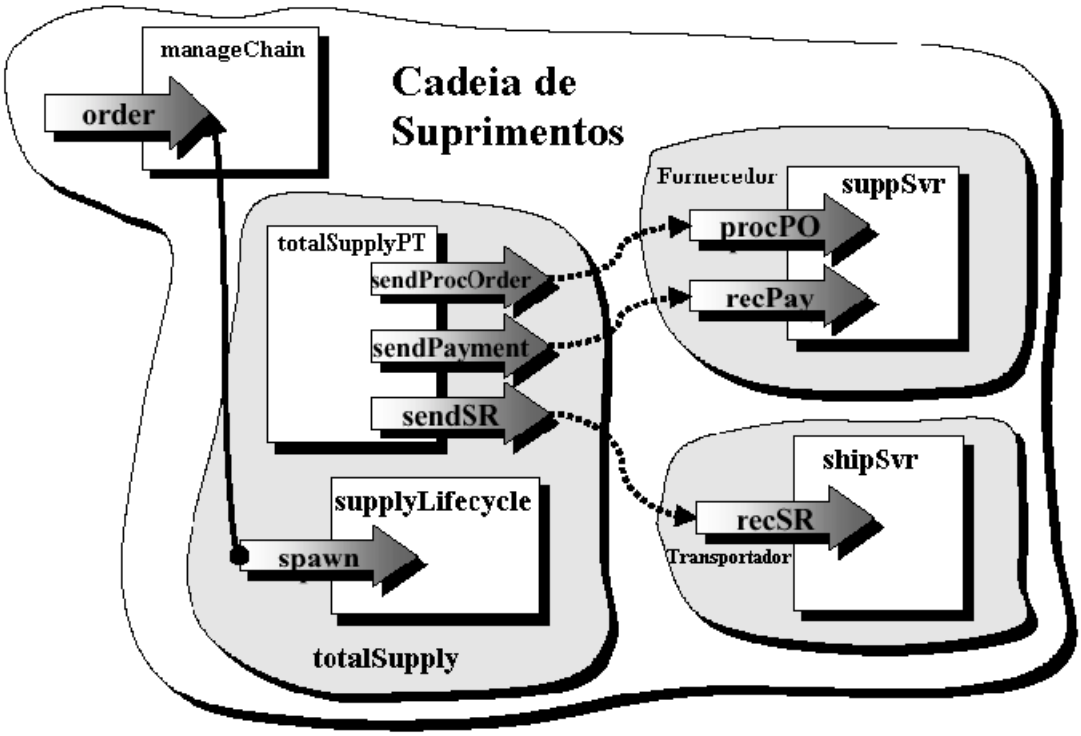

Figura 4.6 – Exemplo de um *Global Model* [LEY01]

O web service do fabricante deve possuir duas portas de entrada, uma para processar a ordem de compra *(procPO)* e outra para receber o pagamento *(recPay)*. O web service do transportador apresenta apenas uma porta para receber a ordem de transporte *(recSR)*. O modelo de fluxo de suprimentos *(totalSupply)* espera uma operação do tipo *spawn* que serve para criar uma nova instância do fluxo. Assim quando chega uma nova ordem de compra *(order)* a mesma dispara uma operação *spawn* que iniciará o fluxo *totalSupplyPT* que iniciará uma ordem de processamento *(sendProcOrder)* para o fabricante. Os outros links funcionarão de maneira similar de acordo com o fluxo do processo.

A linguagem WSFL apresenta algumas características importantes:

- Composição Recursiva;
- Interação Hierárquic*a* e *Peer-to-Peer*;
- Usa WSDL *(Web Services Description Language)* para descrição de protocolos e interfaces de serviço;
- Utiliza WSEL *(Web Services Endpoint Language)* para descrever as características e qualidade dos serviços.

4.5. XLANG

A linguagem XLANG é baseada em XML para descrever o sequenciamento lógico de processos de negócios utilizando várias tecnologias de componentes ou serviços. Ela foi originalmente desenvolvida para o Microsoft *BizTalkServer 2000*. Utilizando *BizTalk Orchestration tools* pode-se desenhar um processo de workflow e depois salvá-lo em um arquivo de XLANG *schema* [DUI01]. As principais características da linguagem XLANG são:

- Construções seqüenciais e paralelas de fluxo;
- Suporte para transações de longa duração com compensação;
- Manipulação flexível para exeções;
- Contratos *Multi-role*.

4.6. BPEL4WS

A BPEL4WS é uma linguagem para especificação formal de processos de negócios e interação de protocolos de negócio *(business interaction protocols)* [VAR03]. BPEL4WS representa a convergência de idéias contidas nas especificações do WSFL e do XLANG [VAR03]. Assim a especificação do BPEL4WS supera as soluções trazidas pela XLANG e WSFL.

A BPEL4WS depende de especificações baseadas em XML, dentre elas: WSDL 1.1, XML Schema 1.0 e XPATH 1.0. Dentre essas a que tem maior influência sobre a BPEL4WS é a WSDL. O coração da modelagem dos processos BPEL4WS é interação *peer-to-peer* entre serviços descritos em WSDL.

Os processos em BPEL4WS importam e exportam recursos exclusivamente por interfaces web services. Os processos são descritos através de duas formas:

- ***Executable Business Process*:** modelam o comportamento de um participante em uma interação de negócio;
- ***Abstract Process*:** utilizam a descrição dos processos para especificar quais os protocolos de negócio serão utilizados entre os parceiros envolvidos.

A BPEL4WS também provém um protocolo de compensação para transações baseado em *open nested transactions* e SAGAS (vide item 3.3) [VAR03]. Isto é implementado através de um *compensation handler* que possibilita especificar as atividades de compensação caso ocorra algum erro/cancelamento do processo. A **Figura 4.7** ilustra um exemplo de código de um *compensation handler*. Maiores detalhes sobre a especificação BPEL4WS são encontrados em [VAR03].

```
<scope>
 <compensationHandler>
    <invoke partner="Seller" portType="SP:Purchasing"
      operation="CancelPurchase"
      inputVariable="getResponse"
      outputVariable="getConfirmation">
      <correlations>
      <correlation set="PurchaseOrder" pattern="out"/>
      </correlations>
    </invoke>
 </compensationHandler>
    <invoke partner="Seller" portType="SP:Purchasing"
      operation="SyncPurchase"
      inputVariable="sendPO"
      outputVariable="getResponse">
  <correlations>
  <correlation set="PurchaseOrder" initiate="yes" pattern="out"/>
  </correlations>
    </invoke>
</scope>
```

Figura 4.7 – Exemplo de um *Compensation Handler* em BPEL4WS [VAR03]

4.7. Conclusão

Com a expansão dos serviços oferecidos pelos web services, a internet começou ser mais eficiente, especialmente para as interações de negócios [NEW02]. A próxima geração da

internet apresentará web services capazes de interações automáticas, combinando acesso direto a aplicações de software e documentos de negócios [NEW02]. Esta mudança de paradigma que possibilita a interação direta de aplicações na internet é o coração dos web services [NEW02].

Conforme apresentado pode-se observar que o workflow distribuído pode ser complementado com o auxílio de web services, pois os web services surgiram para facilitar os processos de negócios via internet através da interação das aplicações.

Vale apena frisar que nenhuma das linguagens apresentadas (WSFL, XLANG, e BPEL4WS) como tantas outras pesquisadas apresenta a preocupação de coordenar uma execução distribuída que agregue os resultados de processos entre vários web services. A proposta em questão utilizará um modelo que permite a execução e coordenação distribuída entre múltiplos participantes através de uma interface de comunicação web services.

4.8. Resumo

O capítulo em questão apresentou o conceito de web services e quais são as suas relações com o conceito de workflow.

Nos item 4.2 apresentou-se uma introdução básica sobre web services e suas tecnologias.

Nos itens 4.3 apresentou-se os relacionamento existente entre web services e workflow.

Nos itens 4.4 a 4.6 apresentou-se as soluções propostas pela IBM e Microsoft para a implementação de processos de negócios utilizando-se web services.

Capítulo 5

ARQUITETURA PROPOSTA (WS IF4T)

5.1. Introdução

A proposta deste trabalho é a elaboração de uma arquitetura de workflow transacional distribuído utilizando web services. A arquitetura proposta trabalha com a hipótese de um cenário de interoperação Hierárquico distribuído. Neste cenário parte-se do pressuposto que um workflow *egine* 1 executa um processo primário que depende dos resultados de um ou mais subprocessos distribuídos para concluir os seus objetivos. Os subprocessos distribuídos são aqueles que são instanciados em outros workflows *engine* (ex: workflow *engine* 2, workflow *engine* 3, etc). Doravante quando houver a referência a um processo pai refere-se a um processo primário e quando houver a referência a um processo filho refere-se a um subprocesso e vice-versa. A **Figura 5.1** ilustra um exemplo de tal cenário.

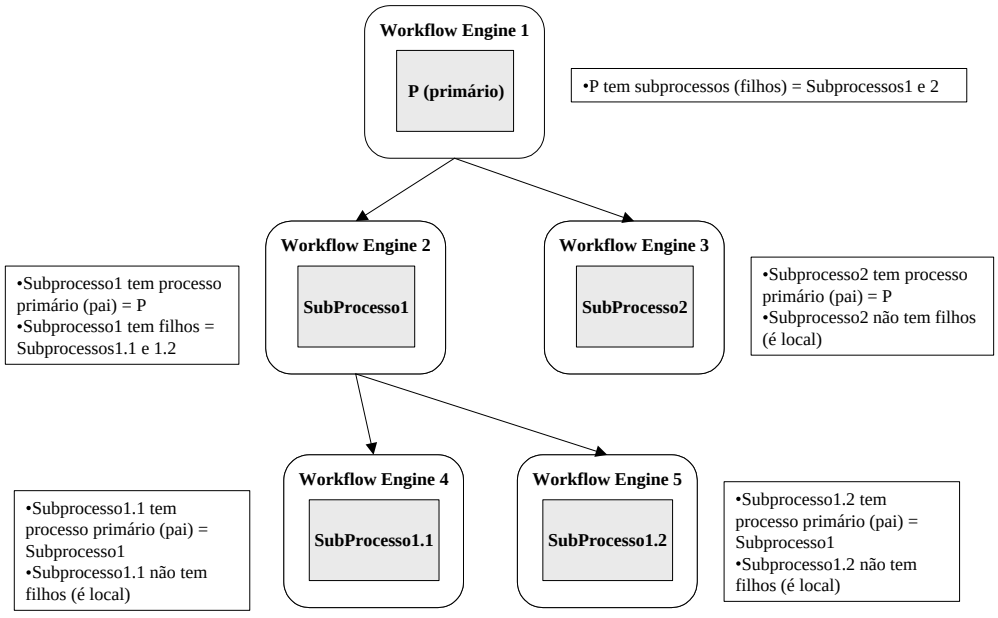

Figura 5.1 – Exemplo de um cenário de Interoperação Hierárquico Distribuído

A arquitetura proposta possibilita que o workflow *engine* do processo primário (Workflow *Engine* 1) interaja com os seus subprocessos (Workflow *Engine* 2 e 3) via web services. A arquitetura possibilita também controlar subprocessos, informando ao workflow *engine* do processo pai quando um subprocesso falhou ou quando este subprocesso se encontra pronto para ser efetivado, ou mesmo quando todos os subprocessos já se encontram "prontos" e aguardam somente uma confirmação do workflow *engine* do processo pai. Este controle é realizado por uma interface 4[9] de comportamento transacional que se comunica com o ambiente distribuído via web services (WS IF4T – Web Services Interface 4 Transacional).

Este capítulo tem o objetivo de apresentar a arquitetura que propõem a WS IF4T. Haverá primeiramente um item mostrando a visão geral da arquitetura. Logo após será apresentado em detalhes os componentes integrantes da arquitetura.

5.2. Arquitetura - Visão Geral

O modelo proposto para esta arquitetura de workflow transacional distribuído é ilustrado na **Figura 5.2**.

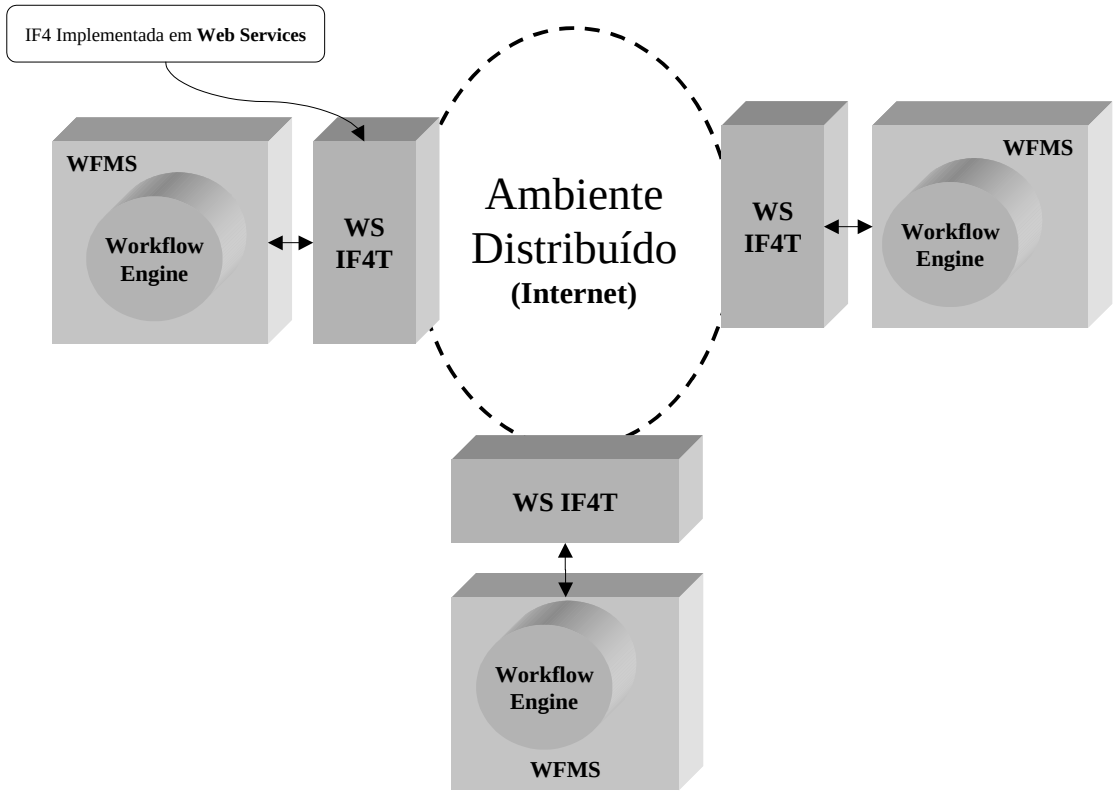

Figura 5.2 – Arquitetura do Sistema

[9] A interface 4 é a interface do Workflow *Reference Model* responsável pela interoperabilidade entre workflows *engines* (ver item 2.2.1).

A WS IF4T é uma fusão de três componentes em uma única interface de interoperabilidade. Os componentes são:

- Interface 4 Abstrata;
- Interface 4 Web Services;e
- Módulo Transacional.

A **Figura 5.3** apresenta os componentes integrantes da WS IF4T. A proposta em questão utiliza algumas funções básicas de consulta que são necessárias para o correto funcionamento do Módulo Transacional da WS IF4T. Estas funções estarão definidas em uma API[10] que deverá ser implementada no WFMS que adotar esta arquitetura.

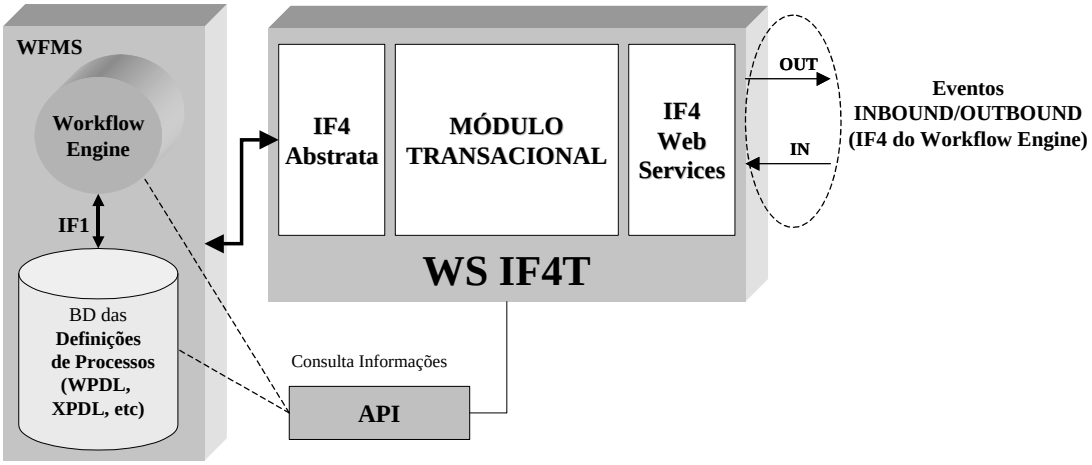

Figura 5.3 – Componentes integrantes da WS IF4T

A seguir será apresentado cada um dos componentes da WS IF4T em detalhes.

5.3. INTERFACE 4 (IF4)

A interface 4 define mecanismos para que um workflow *engine* faça pedidos para outro workflow *engine* para selecionar, instanciar e ativar definições de processos que são conhecidas por esse outro workflow *engine* [ALL96]. Para que isso seja possível é necessário que haja um certo nível de interoperabilidade entre os dois workflows *engines*.

[10] Não faz parte do âmbito desta proposta a implementação desta API, pois isso depende de vários fatores característicos do WFMS (por exemplo: Linguagem de Definição de Processos utilizada, WPDL[WOR99] ou XPDL [WOR02]).

5.3.1 Interoperabilidade

A seguinte terminologia foi retirada do *Workflow Management Coalition Terminology & Glossary* [HOLL99]:

Workflow Interoperability é descrito como: "Habilidade de dois ou mais *Workflow Engines* de se comunicar e trabalhar juntos em um trabalho coordenado." Em [AND99] tem-se uma definição mais detalhada que diz que *Workflow Interoperability* é a habilidade de dois ou mais *Workflows Engines* de se comunicar e interoperar em ordem para coordenar e executar instâncias de processos *(workflow process instances)* entre esses *Engines*.

Workflow Engine é: "um serviço de software ou "motor" que provê um ambiente de execução para uma instância de processo."

Workflow Process Instances é descrito como: "a instância de uma definição de um processo de *workflow*...criada e gerenciada pelo *Workflow Management System*".

***Workkflow Management System* (WFMS)** é descrito como: "um sistema que define, cria e gerencia a execução de *workflows* (ver item 2.2) utilizando softwares, rodando em um ou mais *workflow engines*, que estão aptos a interpretar a definição de processos, interagir com os participantes e, quando requerido, invocar o uso de ferramentas TI e aplicações.

5.4. Interface 4 Abstrata

A interface 4 abstrata da WFMC [AND99] define o conjunto e a padronização de operações necessárias para interoperabilidade de dois ou mais workflow *engines.* Como se trata de uma interface abstrata ela não existe na prática, o que existe são propostas de interoperação que utilizaram-na como base.

Várias propostas de interoperação (ex: Wf-XML, XML-HTTP, etc) utilizaram esta interface como base, portanto é conveniente que a IF4 Web Services da arquitetura proposta se baseie também na especificação da interface 4 abstrata. Assim futuras adaptações entre as interfaces existentes e a nossa serão facilitadas, pois todas se basearam na mesma origem.

5.5. Interface 4 Web Services

A interface 4 web services proposta é uma adaptação da interface 4 abstrata para a realidade dos web services. A adaptação necessária foi permuta dos endereços dos workflows *engines*, pois nas operações da interface abstrata eles são definidos por um número identificador e na IF4 Web services foi necessário que estes endereços fossem convertidos em uma URL que indentificasse o endereço da interface web services a qual pertence o workflow *engine*.

A adaptação de uma operação IF4 Abastrata para uma operação da IF4 Web Services é realizada pelo módulo transacional e será explicada mais adiante.

5.6. Estados dos Processos

Antes de apresentar as operações de interoperabilidade faz-se necessário apresentar os estados válidos para cada processo, pois estes serão referenciados nas operações. Baseando-se no diagrama de estados apresentado na especificação abstrata de interoperabilidade [AND99] foi proposto o diagrama de estados para esta arquitetura (ver **Figura 5.4**).

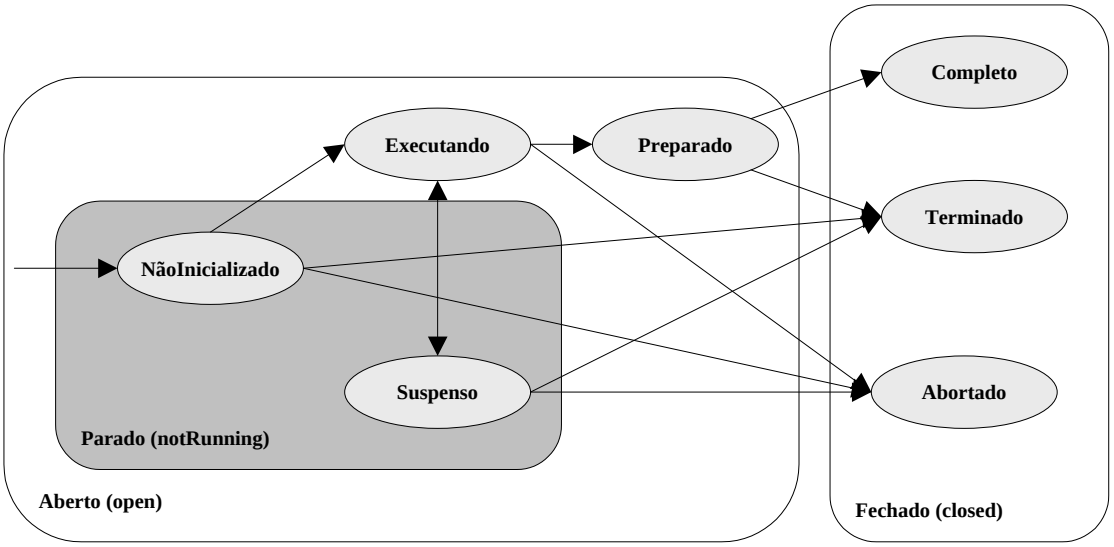

Figura 5.4 – Diagrama de Estados para um Processo de Workflow na Arquitetura

A única diferença apresentada é que se inseriu o estado de “Preparado” no diagrama original da especificação abstrata. O estado “Preparado” serve como um estado intermediário entre o *commit* e o *roll back* (ver **Figura 5.5**).

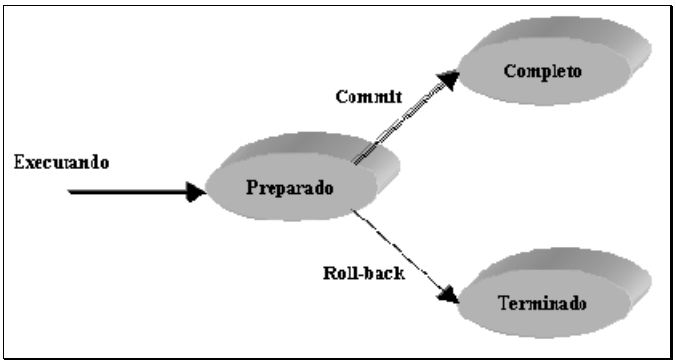

Figura 5.5 – Diagrama de estados para *Commit* e *Roll-Back*

Durante o estado "Preparado" o processo está em um estado que oferece condições para que ele ainda seja reversível. Este tipo de estado intermediário é um recurso comumente utilizado em protocolos de transações distribuídas, dentre eles um dos mais populares, o *two-phase commit protocol* (2PC) que pode ser visto em [KUO96]. A **Tabela 5.1** explica com detalhes cada um dos estados do processo.

Estado	**Descrição**
aberto.parado *(open.notrunning)*	O processo foi instanciado, mas se encontra parado.
Aberto.executando *(open.running)*	O processo está em execução normal do seu trabalho.
Aberto.parado.suspenso *(open.notrunning.suspended)*	O processo se encontra temporariamente interrompido, mas pode voltar a execução a qualquer momento.
Aberto.preparado *(open.prepared)*	O processo executou com sucesso o seu objetivo, mas está aguardando um *commit* externo para confirmar as operações, ou um *roll-back* para compensar as mesmas.
Fechado.completo *(closed.completed)*	O processo foi finalizado com sucesso, e cumpriu seu objetivo. Nesse estado não há mais como reverter as ações tomadas.
Fechado.terminado *(closed.terminated)*	O processo foi finalizado com sucesso, porém não cumpriu seu objetivo. Processos que foram devidamente compensados se encontrarão neste estado.
Fechado.abortado *(closed.aborted)*	Ocorreu algum erro com o processo e este não pode ser finalizado com sucesso. Nesta situação se encontram processos que tiveram algum erro e a compensação não é garantida neste estado.

Tabela 5.1 – Descrição dos Estados de um Processo

5.7. Cenário de Interoperabilidade Suportado

De acordo com o anteriormente citado, a arquitetura proposta trabalha com a hipótese de um cenário de interoperabilidade hierárquico distribuído. Esse cenário assume que uma instância de processo primário (P) de um workflow *engine* irá criar uma ou mais instâncias de um ou mais subprocessos (P_1, P_2,...,P_n) em workflows *engines* distribuídos. Um processo é constituído de atividades. Uma atividade de um processo é o passo lógico ou descrição de uma tarefa que contribua para o objetivo do processo [HOLL95]. A atividade de um processo pode ser uma atividade manual e/ou automatizada computacionalmente[HOLL95].

Em uma execução normal de um processo de workflow a atividade (A_1, A_2,...,A_n) do processo pai (P) responsável por invocar o subprocesso fica ativa e aguarda a finalização da instância do subprocesso para dar prosseguimento no processo primário (ex: A_1 aguarda finalização de P_1, sendo A_1 a atividade invocadora de P_1, ver **Figura 5.6**). Neste cenário a sincronização é alcançada através de notificação de mensagens sobre a mudança dos atributos ou estados das instâncias do subprocesso.

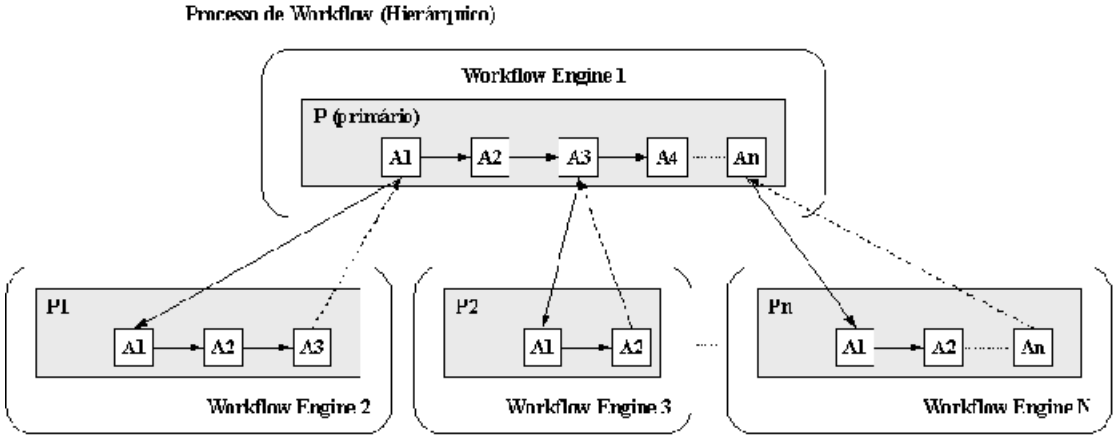

Figura 5.6 – Interoperação Hierárquica Distribuída

5.8. Operações de Interoperabilidade (IF4 Abstrata e IF4 WS)

Para entender as operações de interoperabilidade será utilizado um exemplo prático do funcionamento de um processo de workflow hierárquico distribuído. Neste exemplo um processo pai (do workflow *engine* A) necessita executar um subprocesso filho (no workflow *engine* B). Isso é necessário, pois o workflow *engine* A precisa dos resultados gerados pelo

subprocesso para continuar a execução do seu processo (pai). Para tal devem ocorrer os seguintes eventos:

1. O workflow *engine* A deve se comunicar com o workflow *engine* B para passar todas as informações necessárias para criar uma nova instância de subprocesso utilizando a operação *CreateProcessInstance*. O wokflow *engine* B responde a operação retornando o id da instância do subprocesso criado para o workflow *engine* A ou algum código de erro. A **Tabela 5.2** ilustra as operações *CreateProcessInstance* da IF4 Abstrata e da IF4 Web Services.

CreateProcessInstance		
Descrição:	Instrução para um workflow engine crie uma nova instância de um processo em outro workflow engine utilizando um identificador da definição de processo que se deseja criar.	
Racionalidade:	Operação primordial para que um workflow engine consiga comunicar qual o subprocesso que ele deseja inicializar em um outro workflow engine.	
CreateProcessInstance IF4 Abstrata		
Parâmetros de Entrada:	Engine_id:	identificador do worflow engine de destino.
	Process_Definition_id:	identifica o número da definição de processo que será instancada no engine de destino.
	Parent_id:	identifica o número único da instância do processo primário que está invocando o subprocesso.
	Activity_id:	identifica o número da atividade do processo primário que está invocando a criação de uma nova instância de processo em um workflow engine secundário.
Retorno:	Sub_process_id:	o número identificador correspondente a instância do subprocesso.
CreateProcessInstance IF4 Web Services		
Parâmetros de Entrada:	url_Engine_Origem:	identifica o endereço da IF4 workflow engine requisitante.
	url_Engine_Destino:	identifica o endereço da IF4 workflow engine de destino.
	id_Processo_Pai:	identifica o número único da instância do processo primário que está invocando o subprocesso.
	id_Atividade_Invocadora:	identifica o número da atividade do processo primário que está invocando a criação de uma nova instância de processo em um workflow engine secundário.
	id_Def_Processo_Filho:	identifica o número da definição de processo que será instancada no engine de destino.
Retorno:	id_Resposta:	especifica o retorno da função que pode ser: • Número da Instância do subprocesso (positivo >0) • WF_INVALID_ENGINE (-1) • WF_INVALID_PROCESS_DEFINITION (-2) • WF_ENGINE_ERROR (-3)

Tabela 5.2 – Operação *CreateProcessInstance* (IF4 e WS IF4)

2. Depois o workflow *engine* A precisa transmitir todos os dados relevantes[11] para que o subprocesso seja inicializado. Para isto utiliza-se a operação *SetProcessInstanceAttributes*. O workflow *engine* B responde notificando o workflow *engine* A se a operação teve sucesso ou falhou. A **Tabela 5.3** ilustra as operações *SetProcessInstanceAttributes* da IF4 Abstrata e da IF4 Web Services.

SetProcessInstanceAttributes		
Descrição:	Instrução para um workflow engine atribuir valores para uma instância de um subprocesso em outro workflow engine. Os atributos contêm a lista de um ou mais valores a serem setados na instância do subprocesso.	
Racionalidade:	Quando uma definição de processo é instanciada normalmente ela necessita de algumas informações adicionais antes que ela possa começar (ex: Processo de compra necessita do nome do cliente).	
SetProcessInstanceAttributes IF4 Abstrata		
Parâmetros de Entrada:	Engine_id:	identificador do worflow engine de destino.
	Root_pid:	o identificador da instância do processo invocador.
	Activity_id:	identificador da atividade do processo primário que está invocando o subprocesso.
	Process_id:	identificador da instância de subprocesso que terá seus valores atribuídos.
	Attributes:	uma lista de especificação de atributos contendo: • O nome do atributo; • O tipo do atributo; • O valor que será atribuído.
Retorno:	XXX	XXX
SetProcessInstanceAttributes IF4 Web Services		
Parâmetros de Entrada:	url_Engine_Origem:	identifica o endereço da IF4 workflow engine requisitante.
	url_Engine_Destino:	identifica o endereço da IF4 workflow engine de destino.
	id_Processo_Pai:	identifica o número único da instância do processo primário que está invocando o subprocesso.
	id_Atividade_Invocadora:	identifica o número da atividade do processo primário que invocou o subprocesso.
	id_Instancia_Processo:	identifica o número da instância do subprocesso que terá os valores atribuídos.
	typ_Atributos:	um tipo de variável (record) que contém a estrutura com a lista dos atributos a serem passados para o subprocesso.
Retorno:	flag_Retorno:	especifica o retorno da função que pode ser: • VERDADEIRO (Operação teve sucesso) • FALSO (Operação falhou)

Tabela 5.3 – Operações *SetProcessInstanceAttributes* (IF4 e WS IF4)

[11] Alguns autores chamam estes dados de DRW (Dados Relevantes ao Workflow) que é o conjunto de variáveis compartilhadas entre o workflow *engine* e as aplicações.

3. Este passo é opcional, pois em alguns sistemas, é o workflow *engine* B que pergunta para o workflow *engine* A os dados relevantes do subprocesso utilizando a operação de *GetProcessInstanceAttributes*. O workflow *engine* A responde providenciando para o workflow *engine* B os valores requeridos. A **Tabela 5.4** ilustra as operações *GetProcessInstanceAttributes* da IF4 Abstrata e da IF4 Web Services

GetProcessInstanceAttributes		
Descrição:	Instrução que retorna um conjunto de valores de atributos de um processo.	
Racionalidade:	Pegar os atributos de um processo pode de ser requerido para inicialização de um subprocesso. Essa operação também pode ser um recurso que permita o workflow engine (pai) acompanhar o progresso do seu subprocesso.	
GetProcessInstanceAttributes IF4 Abstrata		
Parâmetros de Entrada:	Engine_id:	identificador do worflow engine de destino.
	Process_id:	identificador da instância de subprocesso que terá seus valores lidos.
	Root_id:	o identificador da instância do processo invocador.
	Activity_id:	identificador da atividade do processo primário que invocou o subprocesso.
	Attributes:	uma lista de especificação de atributos contendo: • O nome do atributo; • O tipo do atributo;.
Retorno:	Attributes:	uma lista de especificação de atributos contendo: • O nome do atributo; • O tipo do atributo; • O valor que está atribuído.
GetProcessInstanceAttributes IF4 Web Services		
Parâmetros de Entrada:	url_Engine_Origem:	identifica o endereço da IF4 workflow engine requisitante.
	url_Engine_Destino:	identifica o endereço da IF4 workflow engine de destino.
	id_Processo_Pai:	identifica o número único da instância do processo primário que está invocando o subprocesso.
	id_Atividade_Invocadora:	identifica o número da atividade do processo primário que invocou o subprocesso.
	id_Instancia_Processo:	identifica o número da instância do subprocesso que terá retornará os valores pedidos.
	typ_Atributos:	Um tipo de variável (record) que contém a lista dos atributos que se desejam obter pelo processo invocador.
Retorno:	typ_Atributos:	Um tipo de variável (record) que contém a lista dos atributos que foram pedidos pelo processo invocador.

Tabela 5.4 – Operações *GetProcessInstanceAttributes* (IF4 e WS IF4)

4. O workflow *engine* A requer ao workflow *engine* B que comece a executar o subprocesso utilizando a operação *ChangeProcessInstanceState* para mudar o estado do subprocesso de *open.notrunning* para *open.running.* Uma vez que o subprocesso tenha sido inicializado o workflow *engine* B responderá a operação notificando o workflow *engine* A com novo estado do subprocesso. A **Tabela 5.5** ilustra as operações *ChangeProcessInstanceState* da IF4 Abstrata e da IF4 Web Services.

ChangeProcessInstanceState		
Descrição:	Instrução que modifica o estado original de uma instância de um processo atribuindo um novo valor de estado.	
Racionalidade:	È um requisito operacional que usuários de sistemas de workflow em interoperação possam iniciar, terminar/abortar ou suspender subprocessos de workflow sempre que seja necessário.	
ChangeProcessInstanceState IF4 Abstrata		
Parâmetros de Entrada:	Engine_id: Process_id: State:	identificador do worflow engine de destino. identificador da instância de processo que terá seu estado modificado. o novo estado que será atribuído ao processo.
Retorno:	XXX	XXX
ChangeProcessInstanceState IF4 Web Services		
Parâmetros de Entrada:	url_Engine_Origem: url_Engine_Destino: id_Instancia_Processo: st_Estado:	identifica o endereço da IF4 workflow engine requisitante. identifica o endereço da IF4 workflow engine de destino. identifica o número da instância do processo que terá o estado alterado. novo estado que será atribuído à instância que pode ser: • open.running • open.notrunning.suspended • closed.completed • closed.terminated • closed.aborted
Retorno:	flag_Retorno:	especifica o retorno da função que pode ser: • VERDADEIRO (Operação teve sucesso) • FALSO (Operação falhou)

Tabela 5.5 – Operações *ChangeProcessInstanceState* (IF4 e WS IF4)

5. Quando o subprocesso for finalizado, o workflow *engine* B comunicará ao workflow *engine* A que ele já se encontra "preparado". Isso é possível através da operação *ProcessInstanceStateChanged* em que o workflow *engine* B passa o seu estado *open.prepared* para o workflow *engine* A Então o workflow *engine* A pode perguntar ao workflow *engine* B pelos resultados utilizando a operação de *Get ProcessInstanceAttributes*. O workflow *engine* B responde providenciando para o workflow *engine* A as informações requeridas. A **Tabela 5.6** ilustra as operações *ProcessInstanceStateChanged* da IF4 Abstrata e da IF4 Web Services.

ProcessInstanceStateChanged		
Descrição:	Notificação para um workflow engine pai que um subprocesso seu mudou de estado.	
Racionalidade:	Em certas circustâncias é fundamental que o processo primário (pai) tenha um modo de saber o que está acontecendo com os seus subprocessos (filhos) para definir a estratégia de operação.	
ProcessInstanceStateChanged IF4 Abstrata		
Parâmetros de Entrada:	Engine_id: Process_id: New State:	identificador do worflow engine do processo invocador (pai). identificador da instância de subprocesso que teve seu estado modificado. o novo estado do processo.
Retorno:	XXX	XXX
ProcessInstanceStateChanged IF4 Web Services		
Parâmetros de Entrada:	url_Engine_Origem: url_Engine_Destino: id_Instancia_Processo: st_Estado:	identifica o endereço da IF4 workflow engine do subprocesso (filho). identifica o endereço da IF4 workflow engine de destino (pai). identifica o número da instância do subprocesso que mudou de estado. o novo estado da instância que pode ser: • open.notrunning • open.running • open.notrunning.suspended • closed.prepared • closed.completed • closed.terminated • closed.aborted
Retorno:	flag_Retorno:	especifica o retorno da função que pode ser: • VERDADEIRO (Operação teve sucesso) • FALSO (Operação falhou)

Tabela 5.6 – Operações *ProcesInstanceStateChanged* (IF4 e WS IF4)

6. Uma vez o processo executando em A obtenha sucesso, o workflow *engine* A pode então informar ao workflow *engine* B para que o seu subprocesso seja confirmado *(committed)* para depois desalocar toda a memória utilizada para o subprocesso. Isto pode ser feito através da operação *ChangeProcessInstanceState* em que o workflow *engine* A muda o estado do subprocesso do workflow *engine* B para *closed.completed*. Caso ocorra alguma falha no processo do workflow *engine* A e este necessite compensar as ações tomadas no subprocesso do workflow *engine* B haverá então uma operação de compensação *(roll-back)* onde o workflow *engine* A utilizará uma operação de *ChangeProcessInstanceState* mudando o estado do subprocesso do workflow *engine* B para *closed.terminated*. Assim o workflow *engine* B executará toda a seqüência de compensão das ações tomadas e desalocará todos os recursos utilizados no subprocesso. A seqüência da execução de tudo o que foi explicado pode ser visto na **Tabela 5.7** e na **Figura 5.7**.

Seqüência e Worklfow Engine	**(Operação ou ação)**
1º Workflow Engine A	*CreateProcessInstance*
2º Workflow Engine B	Responde ao *CreateProcessInstance*
3º Workflow Engine A	*SetProcessInstanceAttributes* *ChangeProcessInstanceState (open.running)*
4º Workflow Engine B	Responde ao *SetProcessInstanceAttributes* Responde ao *ChangeProcessInstanceState*
5º Workflow Engine B	Notifica *ProcessInstanceStateChanged (open.prepared)*
6º Workflow Engine A	Responde a notificação de *ProcessInstanceStateChanged*
7º Workflow Engine A	*GetProcessInstanceAttributes*
8º Workflow Engine B	Responde ao *GetProcess InstanceAttributes*
9º Workflow Engine A	*ChangeProcessInstanceState* *(closed.completed ou closed.terminated)*
10º Workflow Engine B	Responde ao *ChangeProcessInstanceState*

Tabela 5.7 – Seqüência e operações exigidas em um processo Hierárquico

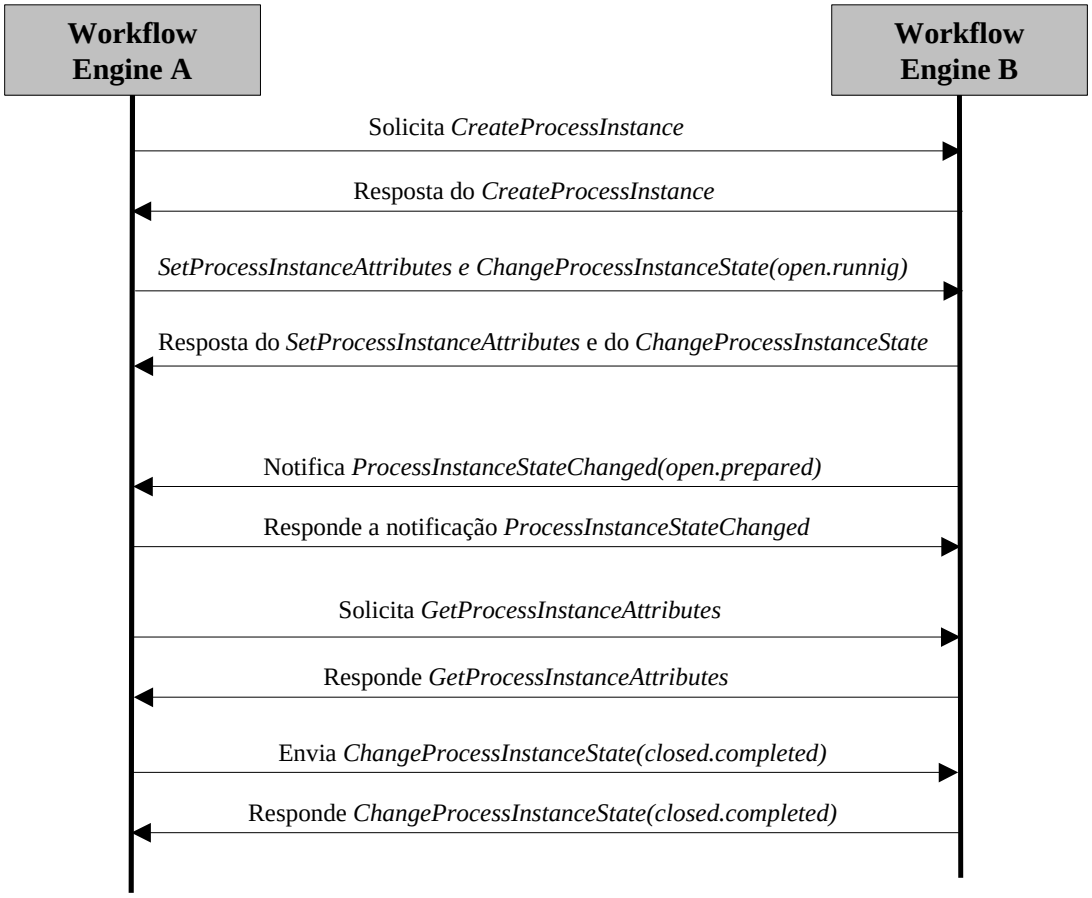

Figura 5.7 – Seqüência de operações em um processo Hierárquico

5.9. Interface Web Services da WS IF4T

Depois de definir as operações necessárias para interoperação será apresentada a interface WSDL que possibilita interoperação dos workflows *engines* via web services. A **Figura 5.8** apresenta o esquema gerado para a interface prosposta.

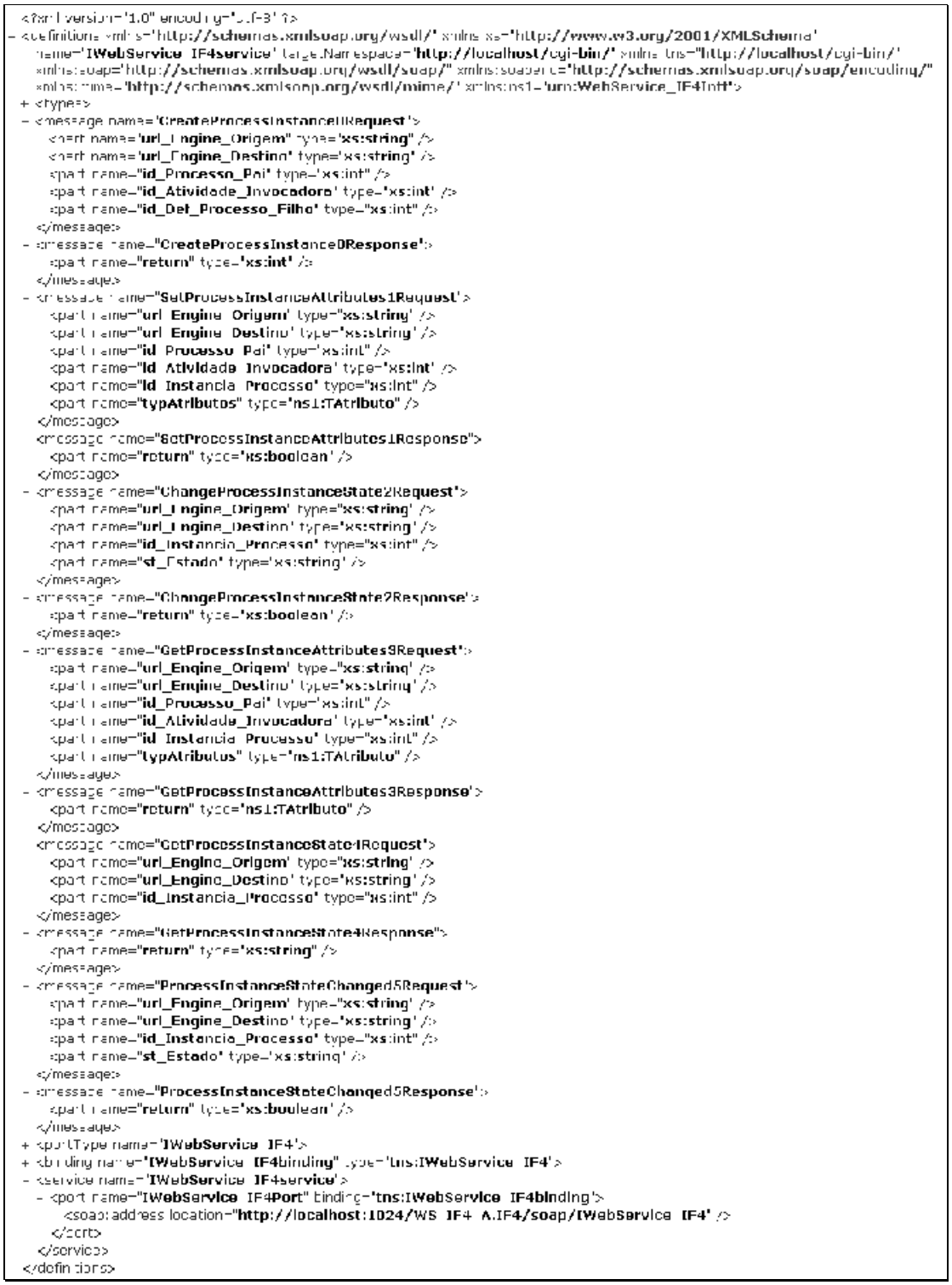

Figura 5.8 – Interface WSDL da WS IF4T

5.10. O Módulo Transacional

Como a arquitetura prevê que cada um dos workflows engines participantes possuam uma WS IF4T com módulo Transacional próprio (ver **Figura 5.2**), o modelo hierárquico pode ser estendido similarmente ao cenário das *Nested Transactions* (ver item 3.3). Assim um conjunto de subprocessos pode conter outros subprocessos recursivamente de maneira a formar uma árvore de processos (ver **Figura 5.9**). Um processo filho só pode iniciar depois que seu processo pai seja inicializado e o processo pai só termina se todos os seus filhos terminam. Se um processo pai é abortado, então todos os seus filhos também o são *(roll back*). E ímportante observar que cada Módulo Transacional é responsável pelos seus filhos e assim por diante (ex: módulo de P é responsável somente por P_1, P_2 e P_n).

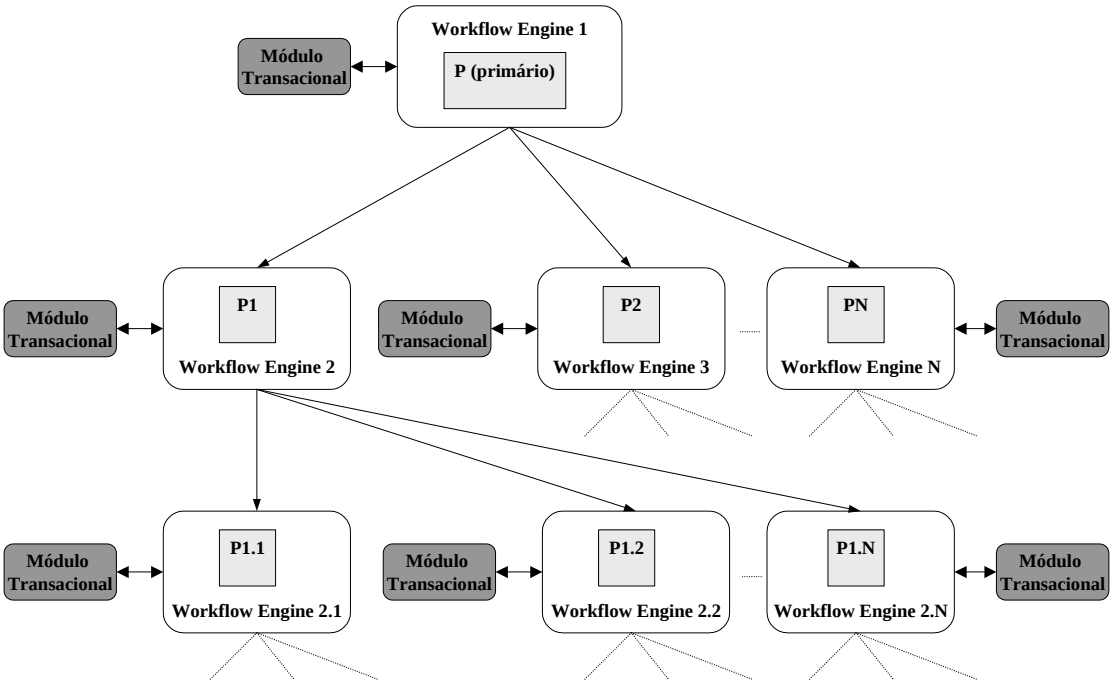

Figura 5.9 – Modelo hierárquico em vários níveis

Para que a arquitetura torne-se operacional é previsto que cada Módulo Transacional filho armazene em uma estrutura as informações do processso pai que invocou o subprocesso, assim o módulo transacional filho tem como manter a sincronização através de mensagens de notificação sobre a mudança dos atributos ou estados das instâncias do subprocesso.

5.10.1 Funcionamento do Módulo Transacional

O módulo transacional é o coração da arquitetura, sendo este reponsável pelas seguintes tarefas:

- Adaptação da operação IF4 Abstrata para IF4 Web Services (IF4→WS IF4) quando se tratar de uma operação OUTBOUND que tenha que ser transmitida do workflow *engine* para o ambiente distribuído;
- Adaptação da operação IF4 Web Services para IF4 Abstrata (WS IF4→IF4) quando se tratar de uma operação INBOUND que chegue do meio distribuído e precise ser repassada ao workflow *engine*;
- Gerenciamento dos eventos que envolvem criação (*CreateProcessInstance*) e estados de processos (*ChangeProcessInstanceState* e *ProcessInstanceStateChanged*)[12];
- Utilização de um algoritmo de princípios transacionais que utilize parâmetros de tempo e estado para o controle de subprocessos, abortando ou confirmando subprocessos e notificando o workflow engine pai quando se tratar de um Módulo Transacional Filho (aquele que está executando um subprocesso pertencente a um processo primário de outra WS IF4T).

A seguir, na **Figura 5.10** será apresentado a estrutura interna Módulo Transacional para o melhor entendimento de seu funcionamento.

[12] Para o restante das operações de interoperabilidade o Módulo Transacional funciona apenas como um conversor de interfaces repassando as operações entre as interfaces.

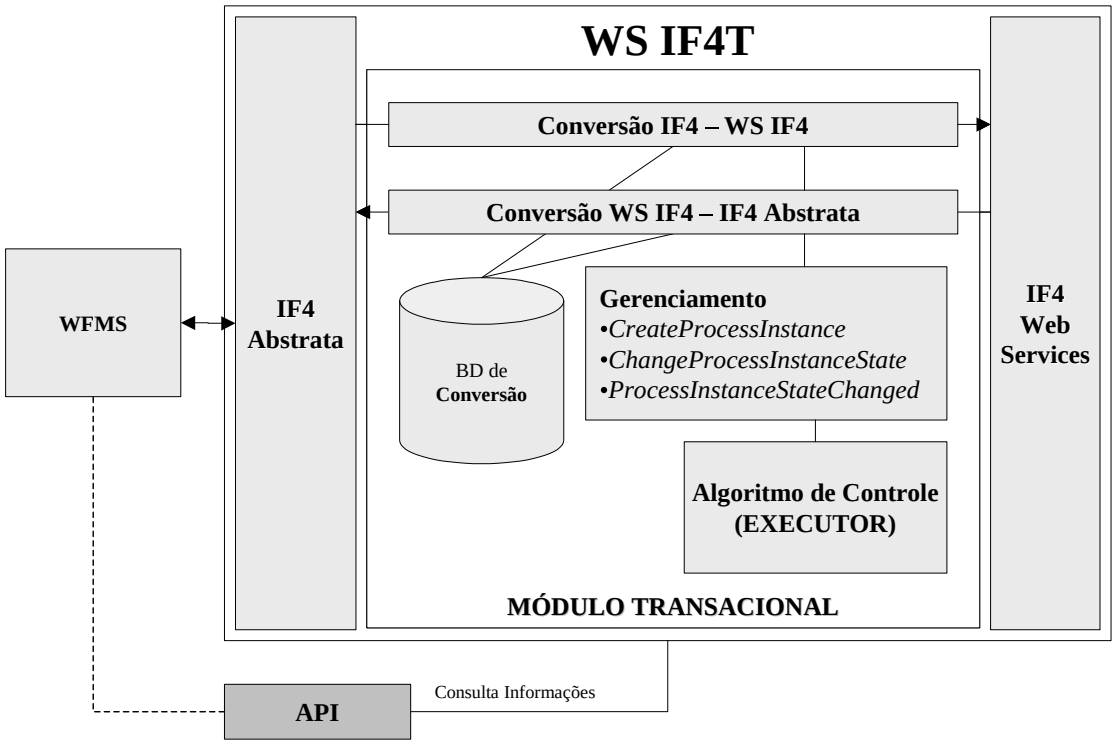

Figura 5.10 – Estrutura interna do Módulo Transacional

A seguir será explicado o funcionamento do Módulo Transacional em tópicos, obdecendo a seqüência: Adaptação das Operações, Algoritmo Executor e Gerenciamento das Operações (*CreateProcessInstance*,*ChangeProcessInstanceState* e P*rocessInstanceStateChanged*).

5.10.2 Adaptação das Operações (IF4↔WS IF4)

A adaptação das mensagens é realizada através de uma conversão. Esta conversão é realizada a partir de uma consulta SQL simples em uma Base de Dados de Conversão onde há uma tabela com os identificadores dos workflows *engines* e os suas respectivas URLs equivalentes.

A **Figura 5.11** apresenta um modelo desta base de Dados implementada em Microsoft *Access*. Durante uma operação de conversão de IF4 para IF4 WS consulta-se o identificador para descobrir a sua URL. Na operação de conversão de IF4 WS para IF4 substiu-se a URL pelo seu identificador equivalente.

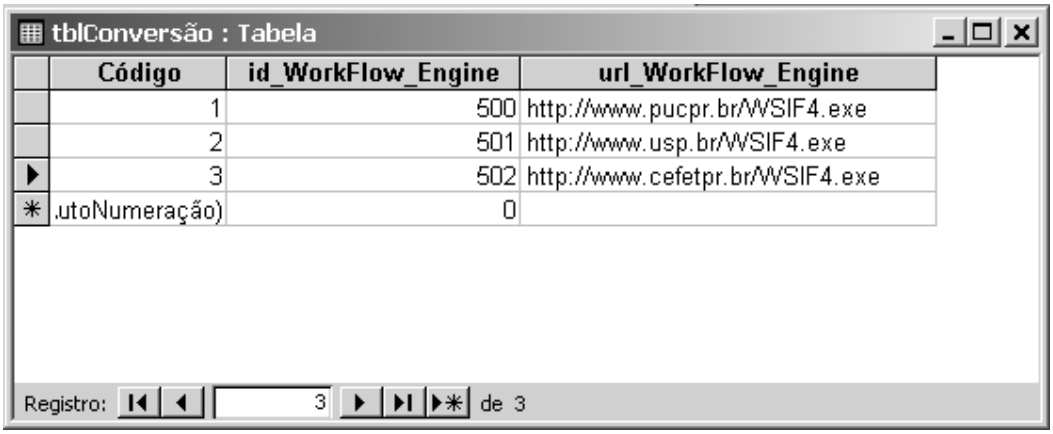

Figura 5.11 – Exemplo de uma Base de Conversão

5.10.3 Modelo Transacional Escolhido

O item 3.3 apresenta vários modelos transacionais e estendidos, dentre esses o modelo transacional escolhido para a proposta foi o **SAGAS**. Este modelo foi escolhido porque é um modelo que oferece facilidade de adaptação para o domínio de workflow sendo indicado para para transações de longa duração pois relaxa os requisitos de isolamento e com isso os resultados executados pelas atividades são visíveis ao mundo exterior [VON99]. O SAGAS utiliza a compensação transacional para garantir a atomicidade do processo, porém o SAGAS não é por si só um modelo perfeito, pois uma vez que ocorra algum erro no processo de workflow todo o trabalho que foi realizado é inteiramente desperdiçado através da compensação das atividades. Nesse aspecto o SAGAS pode ser melhorado através da extensão do modelo, relaxando-se os requisitos de atomicidade através da introdução de *safe-points* (ver item 3.5.5 para maiores detalhes).

O SAGAS também é utilizado em vários workflows transacionais, dentre eles pode-se citar o WIDE ,EXOTICA Project e CROSSFLOW (ver itens 3.5.1, 3.5.3 e 3.5.5). Esse modelo é utilizado também na linguagem BPEL4WS para a compensação de transações (ver item 4.6 para maiores detalhes).

A linguagem BPEL4WS utiliza SAGAS com *open nested transactions* atráves de um protocolo de compensação que permite um controle flexível para compensação. Isto é alcançando através da definição de *compensation handlers* o que de uma maneira específica resultou em uma funcionalidade chamada: LRT (*Long-Running Transaction*) [DUB03]. É importante destacar que a noção de LRT descrita na BPEL4WS é puramente local e ocorre com uma única instância de processo. Não há nenhuma coordenação distribuída que argregue os resultados de múltiplos participantes. A realização de uma coordenação distribuída é um

problema ortogonal fora do âmbito de BPEL4WS [DUB03]. A coordenação distribuída entre processos é implementada nesta proposta.

O modelo SAGAS assume que uma transação T é composta por uma ou mais subtransações, $T=T_1,T_2,...,T_n$. Cada subtransação deve ter também sua correspondente subtransação de compensação ($CT_1,CT_2,..,CT_n$). O modelo baseia-se na execução de duas fases:

1. A seqüência $T_1, T_2,...,T_n$ é inteiramente executada com sucesso; ou
2. A seqüência $T_1, T_2,...,T_j$ com suas correspondentes compensações $CT_j,...CT_2,T_1$ é executada (sendo $1 \leq j \leq n$).

   Se todas as subtransações ($T_1, T_2,...,T_n$) terminarem com sucesso *(commit)* a transação SAGA (T) terá sucesso *(commit)*. Se por algum motivo ocorrer um erro em alguma subtransação, a segunda fase entra em ação, compensando todas as subtransações que já tiverem sido efetuadas *(committed)*.

Segundo [HOLL95] um workflow é a automação/facilitação total ou parcial de um processo. Em um relacionamento hierárquico (ver item 2.4.2) um processo primário P pode ter um ou mais subprocessos ($P_1, P_2,...,P_n$). Partindo-se deste pressuposto é possível adaptar o modelo SAGAS tendo-se a correlação de que uma subtransação ($T_1, T_2,...,T_n$) da transação T será equivalente a um subprocesso ($P_1, P_2,...,P_n$) do processo de workflow P. A **Figura 5.12** ilustra graficamente esta adaptação. Vale frisar que nem todo processo de workflow pode ser tratado como uma transação. Isto se aplica, por exemplo, para processos que requeiram intervenção humana.

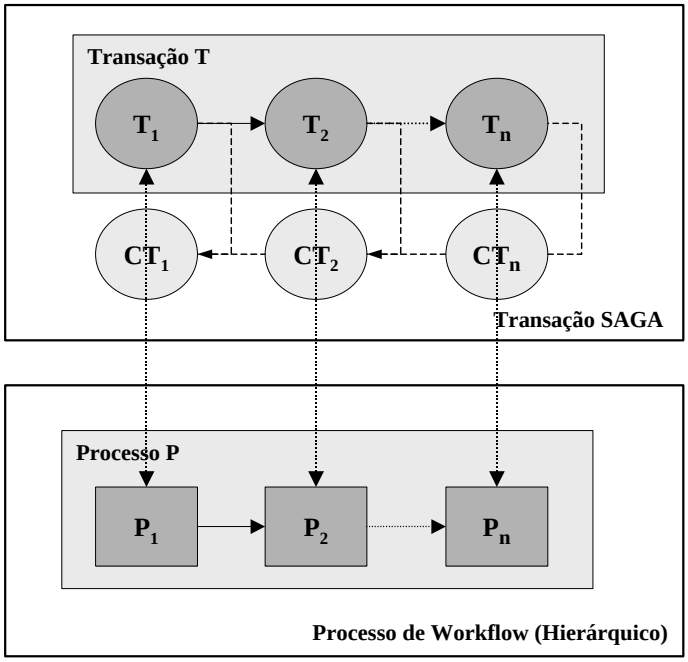

Figura 5.12 – Adaptação do Modelo SAGA para Workflow Hierárquico

Para o modelo ser totalmente funcional é necessário que haja uma compensação correspondente do subprocesso (CP_n) para cada subprocesso (P_n) do workflow. Esta proposta descreverá de maneira abstrata[13] os subprocessos de compensação, haja visto que a compensação de cada subprocesso exigirá um conjunto de atividades de compensação que deverão estar definidas na própria linguagem de definição de processos(que pode ser XPDL[WOR02],WPDL [WOR99] ou qualquer outra). A arquitetura proposta não se preocupa com este detalhe, pois não está vinculada a nenhuma linguagem de definição de processos em específico.

5.10.4 Algoritmo EXECUTOR

O Algoritmo Executor é o responsável por controlar o andamento e situação dos processos e subprocessos. Ele determina também se um processo está preparado, se vai ser confirmado (*committed*) ou abortado (*aborted*). O algoritmo utiliza uma lógica com princípios baseados no modelo SAGAS é utiliza parâmetros de tempo e estado dos processos para realizar o controle transacional. O algoritmo se baseia em três parâmetros: tempo de conclusão (tempo normal que um processo gasta para ser finalizado), tempo máximo de conclusão (tempo máximo tolerável para que um processo seja finalizado) e estado atual do processo. O algoritmo relaxa requisitos de isolamento (pois pode-se ter acesso a todos os estados de todos os processos durante sua execução) e garante que o sistema sempre seja finalizado com as propriedades de atomicidade, consistência e durabilidade asseguradas [VON99]. Para melhor entendimento o algoritmo será divido em três etapas distintas de tempo:

1. Quando Tempo Atual < Tempo de Conclusão;
2. Tempo Atual = Tempo de Conclusão;
3. Tempo Atual = Tempo Máximo de Conclusão.

A **Figura 5.13** apresenta o algoritmo executor quando o Tempo Atual < Tempo de Conclusão (os comandos em maiúsculo representam as operações de comunicação, ver **Tabela 5.8** para maiores detalhes).

[13] Não faz parte do âmbito desta proposta a formulação de uma linguagem de definição de processos que ofereça recursos de compensação.

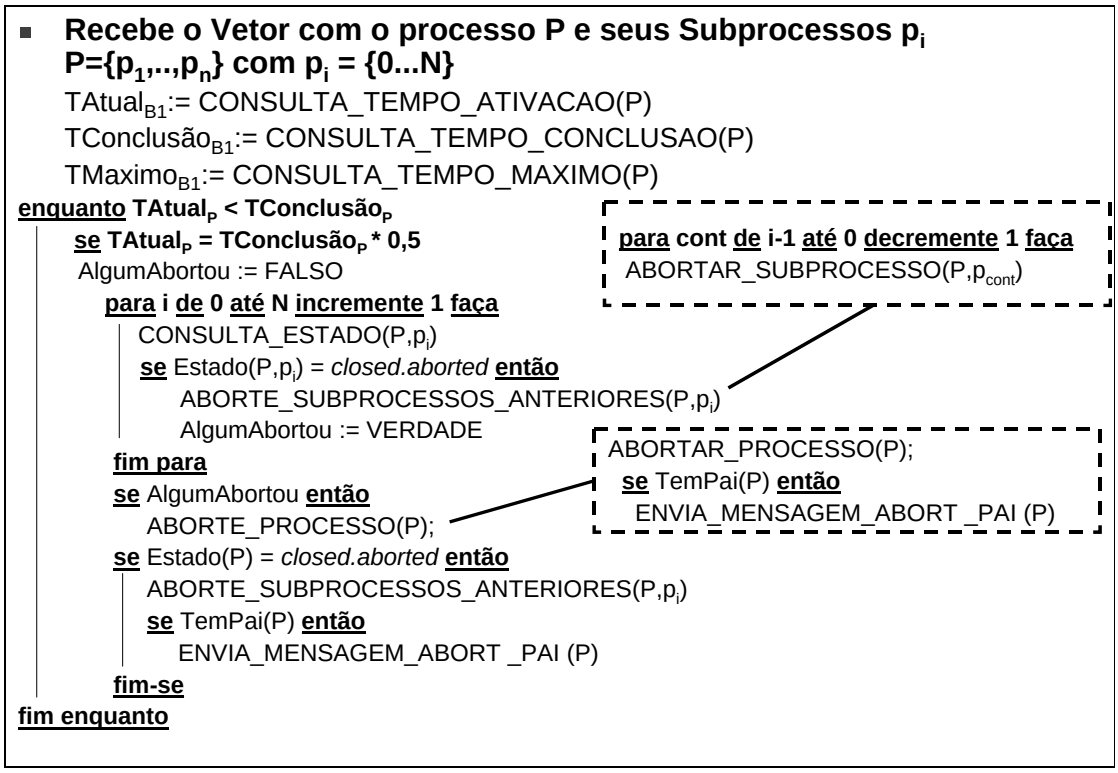

Figura 5.13 – Algoritmo Executor (Tempo Atual < Tempo Conclusão)

A **Figura 5.14** apresenta o algoritmo executor quando o Tempo Atual = Tempo de Conclusão (os comandos em maiúsculo representam as operações de comunicação, ver **Tabela 5.8** para maiores detalhes).

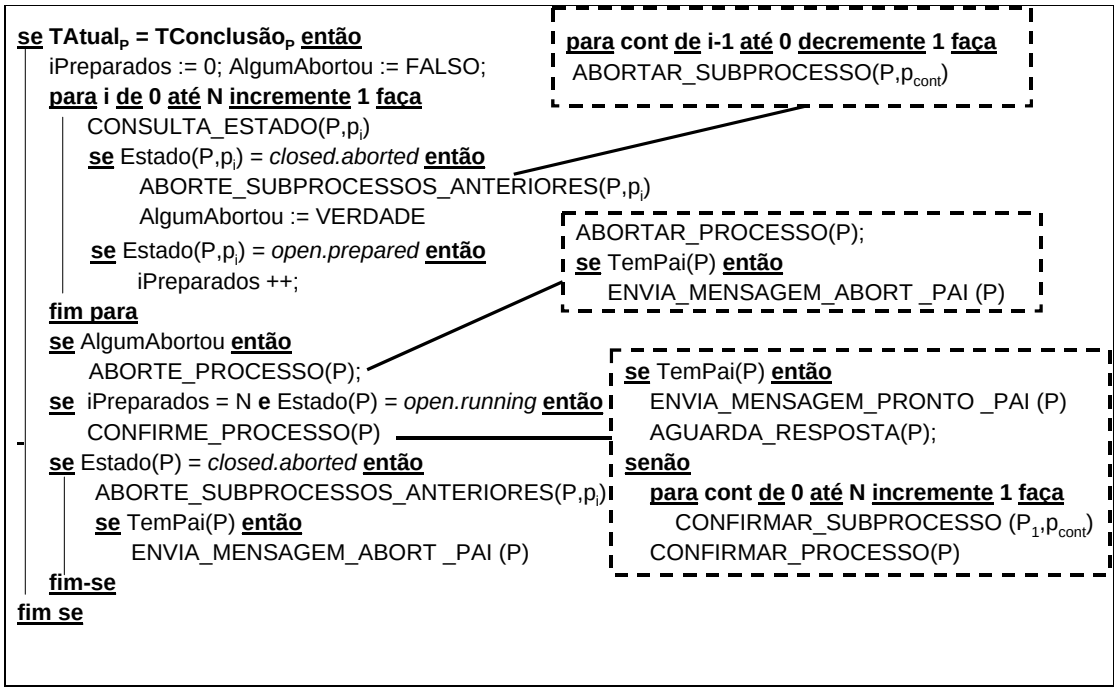

Figura 5.14 – Algoritmo Executor (Tempo Atual = Tempo Conclusão)

A **Figura 5.15** apresenta o algoritmo executor quando o Tempo Atual = Tempo Máximo de Conclusão (os comandos em maiúsculo representam as operações de comunicação, ver **Tabela 5.8** para maiores detalhes).

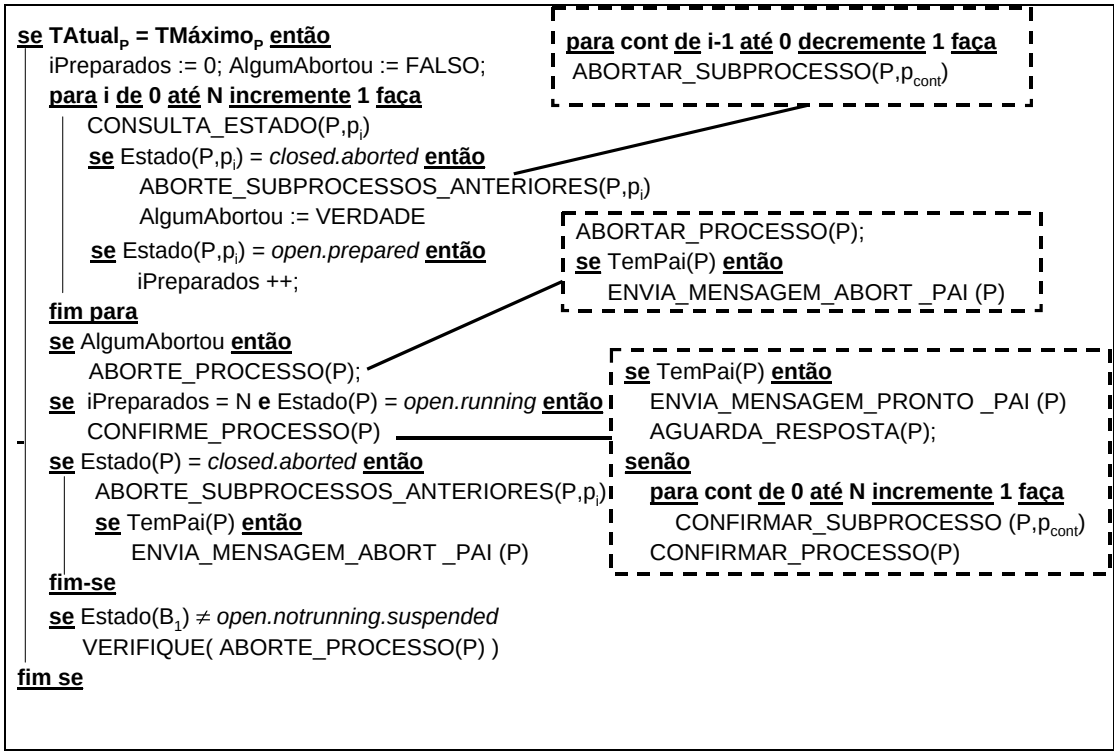

Figura 5.15 – Algoritmo Executor (Tempo Atual = Tempo Máximo de Conclusão)

OBS: Toda vez que aparecer o procedimento VERIFIQUE ou VERIFICA nos algoritmos indica um procedimento que verifica se o processo foi ou não abortado ou suspenso. Assim no exemplo da **Figura 5.15** na linha VERIFIQUE(ABORTE_PROCESSO(P)) será realizada a verificação se o processo P já foi ou não abortado, caso ele não esteja abortado o procedimento ABORTE_PROCESSO será executado.

5.10.5 Funções utilizadas pelo Algoritmo Executor

O algoritmo executor utiliza algumas funções que fazem acesso as operações de interoperabilidade da interface 4. Isto é necessário para que o Algoritmo possa consultar e interagir com os processos que estão em andamento no ambiente distribuído. A **Tabela 5.8** apresenta todas funções necessárias que o Executor utiliza.

Operação	Descrição	Operação utilizada na IF4
CONSULTA_TEMPO_ATIVACAO	Consulta e retorna o tempo em que um subprocesso foi ativado.	*GetProcessInstanceAttributes*
CONSULTA_TEMPO_CONCLUSAO	Consulta e retorna o tempo em que um subprocesso tem a previsão de estar concluído.	*GetProcessInstanceAttributes*
CONSULTA_TEMPO_MÁXIMO	Consulta e retorna o tempo máximo tolerável que um subprocesso tem a previsão de estar concluído.	*GetProcessInstanceAttributes*
CONSULTA_ESTADO	Consulta e retorna o estado atual de um processo/subprocesso.	*GetProcessInstanceState*
ABORTAR_PROCESSO	Envia um estado de closed.aborted ou closed.terminated para um subprocesso.	*ChangeProcessInstanceState*
ENVIA_MENSAGEM_PAI	Envia uma mensagem para o processo pai informando o estado atual do processo que está sendo controlado.	*ProcessInstanceStateChanged*
CONFIRMAR_PROCESSO	Envia um estado de closed.completed para um subprocesso.	*ChangeProcessInstanceState*

Tabela 5.8 – Operações utilizadas pelo Executor

5.10.6 Gerenciamento das Operações

A estratégia escolhida para o gerenciamento dos eventos é a de implementar-se um vetor em memória (ver anexo para maiores detalhes) que armazena informações referentes a cada processo distribuído que está sendo executado no workflow engine bem como os seus respectivos subprocessos. Este vetor também armazena a informação sobre o workflow engine pai quando for o caso.

O gerenciamento de eventos é composto por seis lógicas de funcionamento que funcionam em cima do vetor de processos e realizam a comunicação com o Algoritmo executor. Para tanto é necessário implementar-se uma lógica nas seguintes operações:

- *CreateProcessInstance* (sentido OUTBOUND);
- *CreateProcessInstance* (sentido INBOUND);
- *ChangeProcessInstanceState* (sentido OUTBOUND);
- *ChangeProcessInstanceState* (sentido INBOUND);
- *ProcessInstanceStateChanged* (sentido OUTBOUND);
- *ProcessInstanceStateChanged* (sentido INBOUND);

A **Figura 5.16** apresenta a lógica de funcionamento do *CreateProcessInstance* OUTBOUND (os comandos em maiúsculo representam as funções que são fornecidas pela API, ver item 5.10.7). Vale frisar que uma operação deste tipo sempre vai ser utilizada para um processo pai criar um subprocesso filho em um outro workflow *engine*. Assim toda vez que ocorra uma operação dessas é necessário que se armazene no vetor de processos do Módulo Transacional do workflow *engine* pai as informações de processo principal e quais são o(s) seu(s) subprocesso(s) que está sendo instanciado.

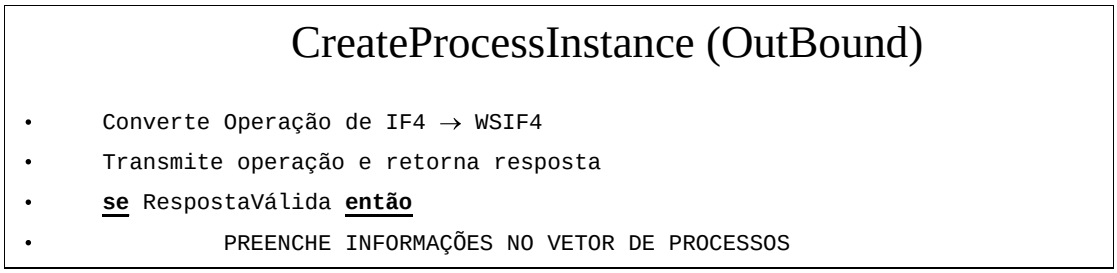

Figura 5.16 – Lógica de funcionamento do *CreateProcessInstance* (OUTBOUND)

A **Figura 5.17** apresenta a lógica de funcionamento do *CreateProcessInstance* INBOUND (os comandos em maiúsculo representam as funções que são fornecidas pela API). Vale frisar que uma operação deste tipo ocorre quando chega a ordem de um processo pai para criar um subprocesso filho no workflow *engine* filho. Assim toda vez que ocorra uma operação dessas é necessário verificar se o subprocesso que acabou de ser criado irá disparar outros subprocessos no futuro, pois se isso acontecer é necessário que o Módulo Transacional controle esses subprocessos que serão disparados, caso contrário o subprocesso criado pelo processo pai fica sendo local e não necessita ser controlado do Módulo Transacional.

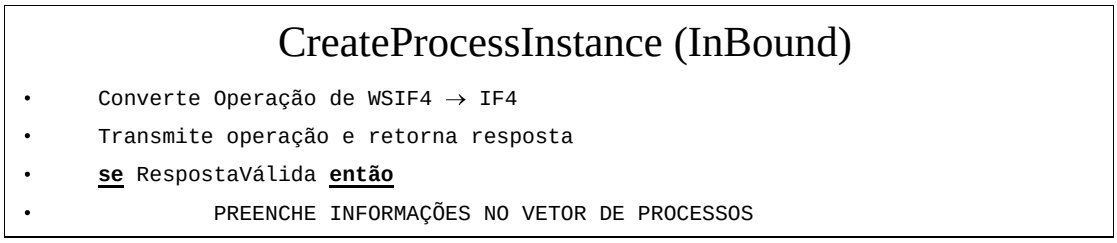

Figura 5.17 – Lógica de funcionamento do *CreateProcessInstance* (INBOUND)

A **Figura 5.18** apresenta a lógica de funcionamento do *ChangeProcessInstanceState* OUTBOUND. Uma operação deste tipo sempre ocorre quando um processo pai der a ordem para um de seus subprocessos mudar de estado. Assim o estado atribuído pelo processo pai também deve ser respeitado e alterado no Módulo Transacional.

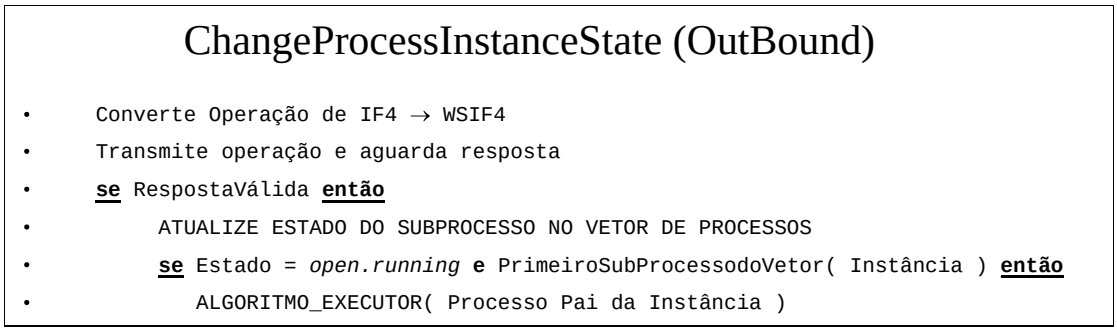

Figura 5.18 – Lógica de funcionamento do *ChangeProcessInstanceState* (OUTBOUND)

A **Figura 5.19** apresenta a lógica de funcionamento do *ChangeProcessInstanceState* INBOUND. Uma operação deste tipo ocorre quando chega a ordem de um processo pai para um processo mudar o estado de um subprocesso. Assim o estado atribuído pelo processo pai também deve ser respeitado e alterado no Módulo Transacional.

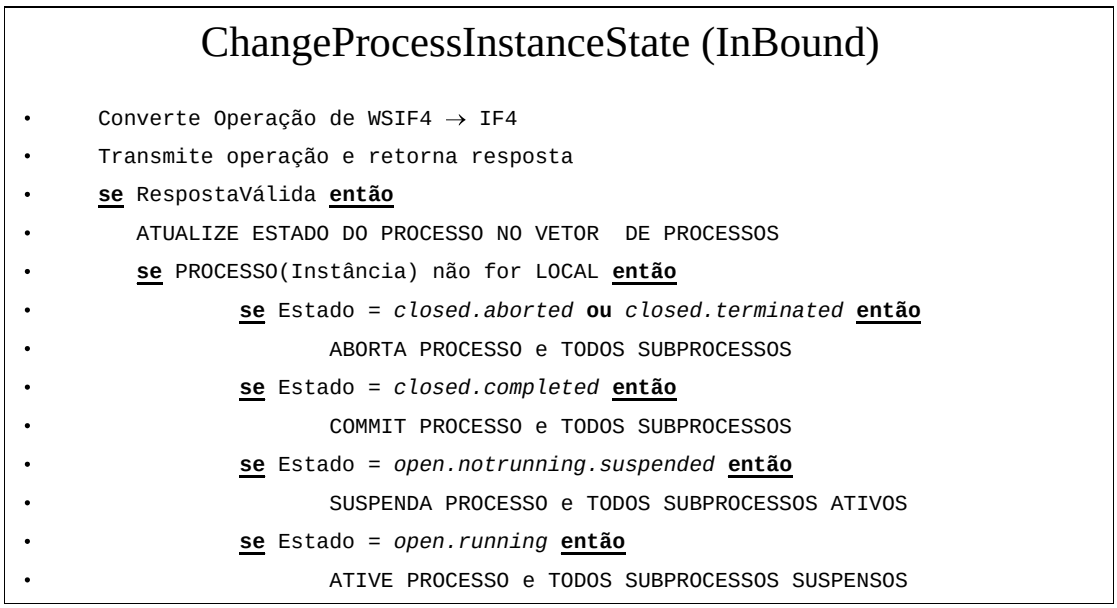

Figura 5.19 – Lógica de funcionamento do *ChangeProcessInstanceState* (INBOUND)

A **Figura 5.20** apresenta a lógica de funcionamento do *ProcessInstanceStateChanged* OUTBOUND. Uma operação deste tipo ocorre quando um subprocesso filho está notificando uma mudança de estado para seu processo pai. Assim as mudanças de estados enviadas pelo subprocesso ao processo pai devem ser alteradas no Módulo Transacional local do subprocesso.

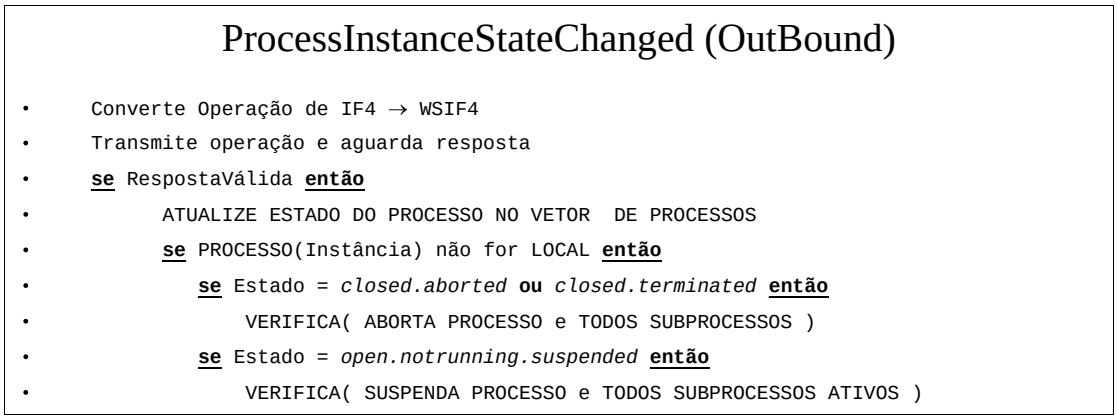

Figura 5.20 – Lógica de funcionamento do *ProcessInstanceStateChanged* (OUTBOUND)

A **Figura 5.21** apresenta a lógica de funcionamento do *ProcessInstanceStateChanged* INBOUND. Essa operação ocorre quando um processo pai recebe a notificação de uma mudança de estado de um subprocesso filho. Assim as mudanças de estados notificadas pelos subprocessos filhos devem ser alteradas no Módulo Transacional local do processo pai.

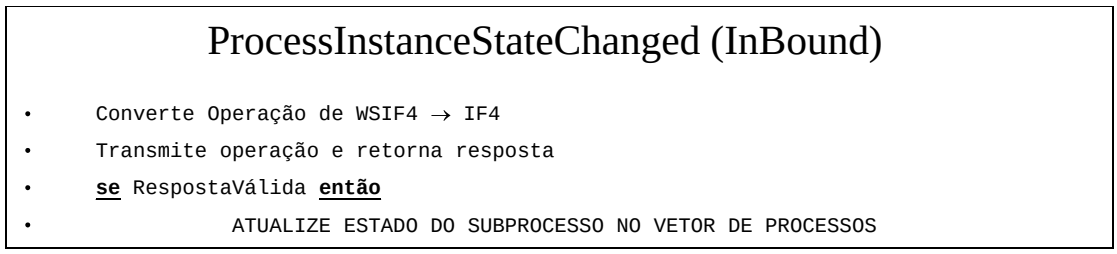

Figura 5.21 – Lógica de funcionamento do *ProcessInstanceStateChanged* (INBOUND)

5.10.7 Funções da API

Conforme observado, alguns algoritmos de Gerenciamento de operações necessitam de algumas funções de API para consultar algumas informações do WFMS local (no qual a arquitetura está implantada). A arquitetura propõe apenas a definição das funções desta API, visto que sua implementação fica a cargo do WFMS a ser implantado. A **Tabela 5.9** apresenta todas funções necessárias que esta API deve fornecer ao Módulo Transacional.

Operação	**Parâmetros**	**Descrição**
GET_PROCESS_DEFINITION	ENTRADA: id_Instância_Processo RETORNO: id_Definição_Processo	A partir de uma instância de um Processo retona de qual tipo é a instância (idDefinição)
GET_SUB_PROCESS	ENTRADA: id_Definição_Processo RETORNO: vetor com as Definições dos subprocessos	Função para descobrir quais são os subprocessos de um determinado processo
HAS_SUB_PROCESS	ENTRADA: id_Definição_Processo RETORNO: VERDADEIRO ou FALSO	Função utilizada para descobrir se um subprocesso é distribuído ou se é local.

Tabela 5.9 – Funções que deverão ser oferecidas na API

5.11. Resumo

O capítulo em questão apresentou uma arquitetura para a implementação de um workflow transacional distribuído utilizando-se de uma interface de interoperabilidade implementada em web services (WS IF4T).

Nos itens 5.3 até 5.9 apresentou-se os conceitos da interface quatro (IF4), suas características e operações, juntamente com o mapeamento de suas operações para uma interface WS IF4 que possibilita a implementação da interface 4 como um serviço de web services.

Nos itens 5.10 até 5.10.7 apresentou-se o coração da arquitetura, o Módulo Transacional, suas características e seus componentes.

Capítulo 6

ESTUDO DE CASO E SIMULAÇÃO

6.1. Introdução

Após apresentar a arquiterura proposta faz-se necessário a realização de um estudo de caso para ilustrar uma situação real de processos de workflow distribuído interagindo em um cenário de interoperabilidade Hierárquico (*Nested Subprocesses*). Após se contextualizar o cenário escolhido será realizado uma simulação de software para validar o funcionamento da interface proposta (WS IF4T).

6.2. O Cenário Escolhido (Loja Virtual de Compras pela Internet)

O cenário escolhido para simular o funcionamento da arquitetura proposta foi o de uma loja virtual de compras pela Internet. A loja virtual apresenta um web site que apresenta uma interface onde é possível realizar a compra de vários produtos via browser. Neste estudo de caso é apresentada a simulação da compra de um livro através desta loja virtual. Para tal o cliente (comprador) utilizará as seguintes operações no web site da loja virtual:

- **Procurar Livro**: nesta operação o cliente realiza a procura do livro desejado. Esta operação não será detalhada neste estudo de caso.
- **Processar Pedido**: nesta operação o cliente confirma a compra do livro pesquisado passando as suas informações pessoais necessárias para que a compra se efetive. Todos os processos e subprocessos que envolvem essa operação serão detalhados neste estudo de caso.
- **Cancelar Pedido**: nesta operação o cliente necessita cancelar o seu pedido de compra. Para tal, ele passa o número identificador do pedido e o WFMS (*Workflow Management System*) da loja virtual cancela todo os processos

envolvidos com a compra do referido cliente. Esta operação será detalhada neste estudo de caso.

A seguir é apresentada uma lista de todas as organizações que participam (direta ou indiretamente) no processo de compra de um livro através da Loja virtual:

- A própria loja virtual que será descrita neste estudo com o nome fictício de **NetBUY**;
- A livraria na qual a loja virtual encaminha seus pedidos de compra de livros: **MEGABOOKS** (nome fictício);
- A editora que a livraria encomenda os livros caso ela não o livro em estoque: **Brasil PRESS** (nome fictício);
- A empresa transportadora que realiza a entrega dos produtos comprados na loja virtual: **EXPRESSO Brasil** (nome fictício).

Assim tem-se um cenário típico de interoperabilidade distribuída entre os vários WFMS (*Workflow Management System*) das organizações participantes. A **Figura 6.1** apresenta a ilustração deste cenário para os WFMS participantes comunicando-se via Internet.

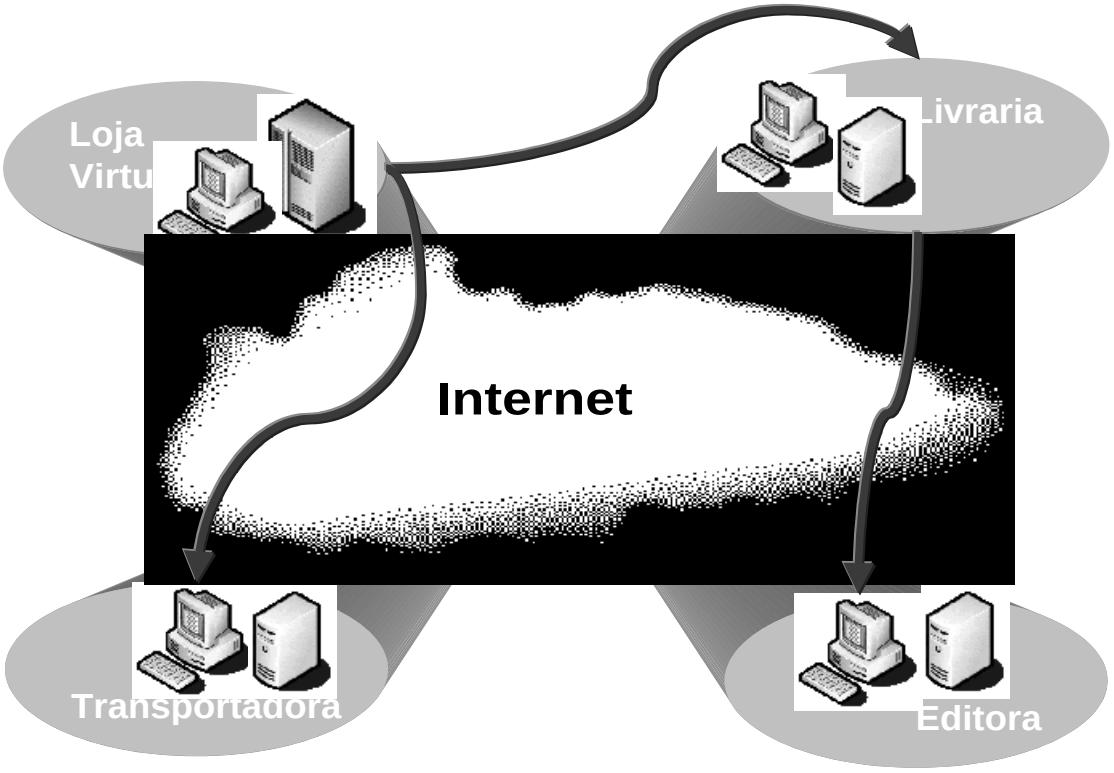

Figura 6.1 – Cenário com vários WFMS interoperando pela Internet

Este panorama descreve um cenário de interoperação hierárquica onde a loja virtual executa um processo primário (processo pai) que necessita de outros subprocessos (processos filhos) para sua conclusão. A **Figura 6.2**. apresenta uma visão global de todos os processos envolvidos com a compra do livro através da loja virtual.

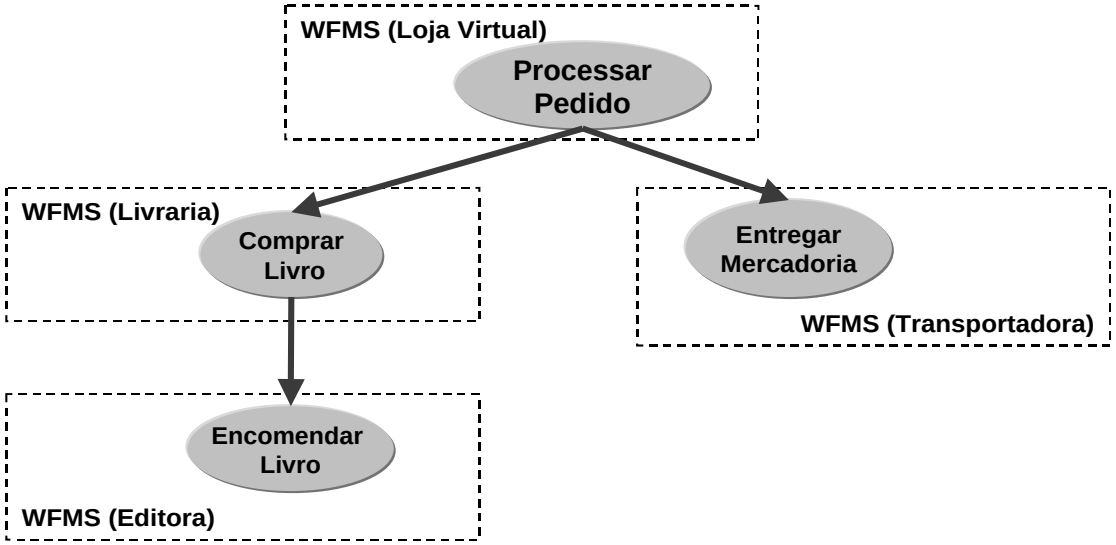

Figura 6.2 – Visão Global do cenário de interoperação Hierárquica dos processos

6.2.1 Processando um Pedido

Conforme descrito anteriormente o processamento de um pedido consiste em uma operação oferecida pela loja virtual onde o cliente confirma a compra do livro desejado passando as informações necessárias para que a compra se efetive. Esta operação dispara automaticamente um processo chamado “Processar Pedido” no WFMS da loja virtual. Este processo pode ser visto em detalhes na **Figura 6.3**, os retângulos da figura apresentam o conjunto de atividades que compõem o processo e os circulos apresentam os pontos de inicialização ou finalização do processo. Vale frisar que os retângulos mais escuros apresentam as atividades que invocam subprocessos distribuídos, e que um processo pode ser finalizado de duas maneiras distintas:

- **Fim**: o processo finalizou e o seu objetivo principal foi alcançado com sucesso;
- **Fim (Terminado)**: o processo foi finalizado mas o seu objetivo principal não foi alcançado.

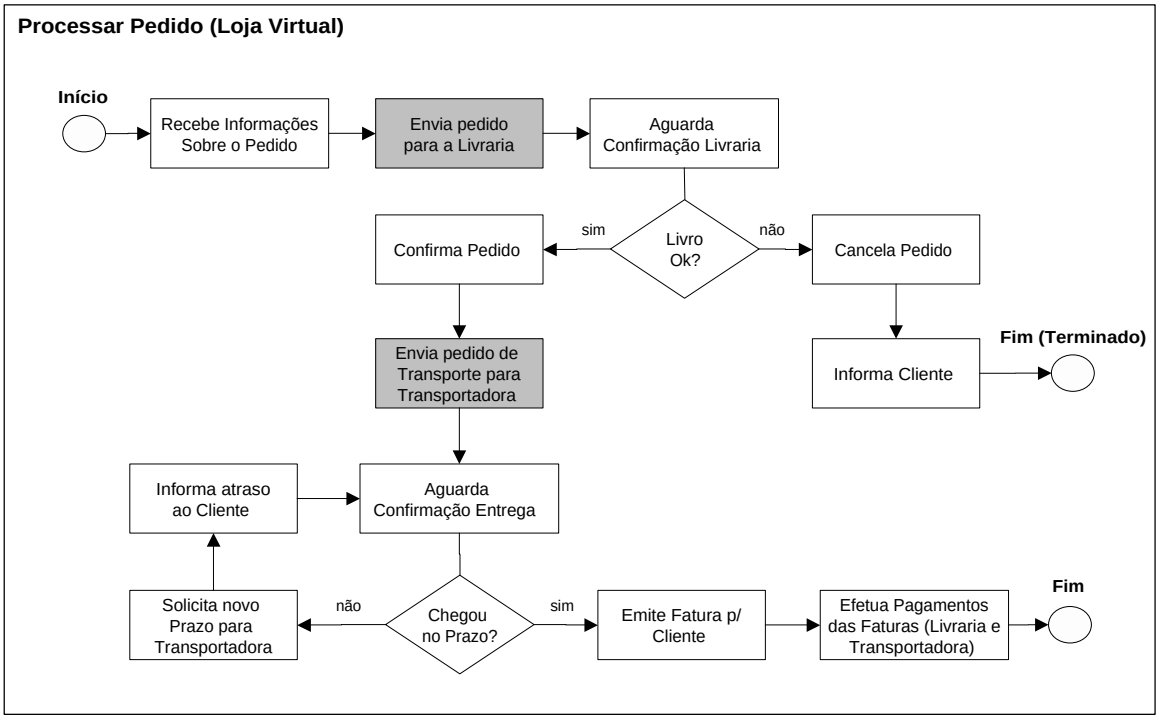

Figura 6.3 – Descrição do processo executado na Loja Virtual para compra de um livro

Conforme observado na **Figura 6.3** o processo da loja virtual NetBUY possui duas atividades que invocam dois subprocessos que serão executados em outros WFMS (*Workflow Management Sytem*). A atividade "Envia pedido para Livraria" invoca o processo "Comprar Livro" no WFMS da livraria MEGABOOKS. A atividade "Envia pedido de Transporte para Transportadora" invoca o processo "Entregar Mercadoria" no WFMS da transportadora EXPRESSO Brasil (vide **Figura 6.2** para melhor compreensão).

O processo "Comprar Livro" é apresentado na **Figura 6.4**. Este processo também possui uma atividade chamada "Encomenda Livro para Editora" que é responsável por invocar um subprocesso que será executado no WFMS da editora Brasil PRESS. Vale frisar que o processo "Comprar Livro" só executará o subprocesso no WFMS da editora senão houver o livro pedido em estoque.

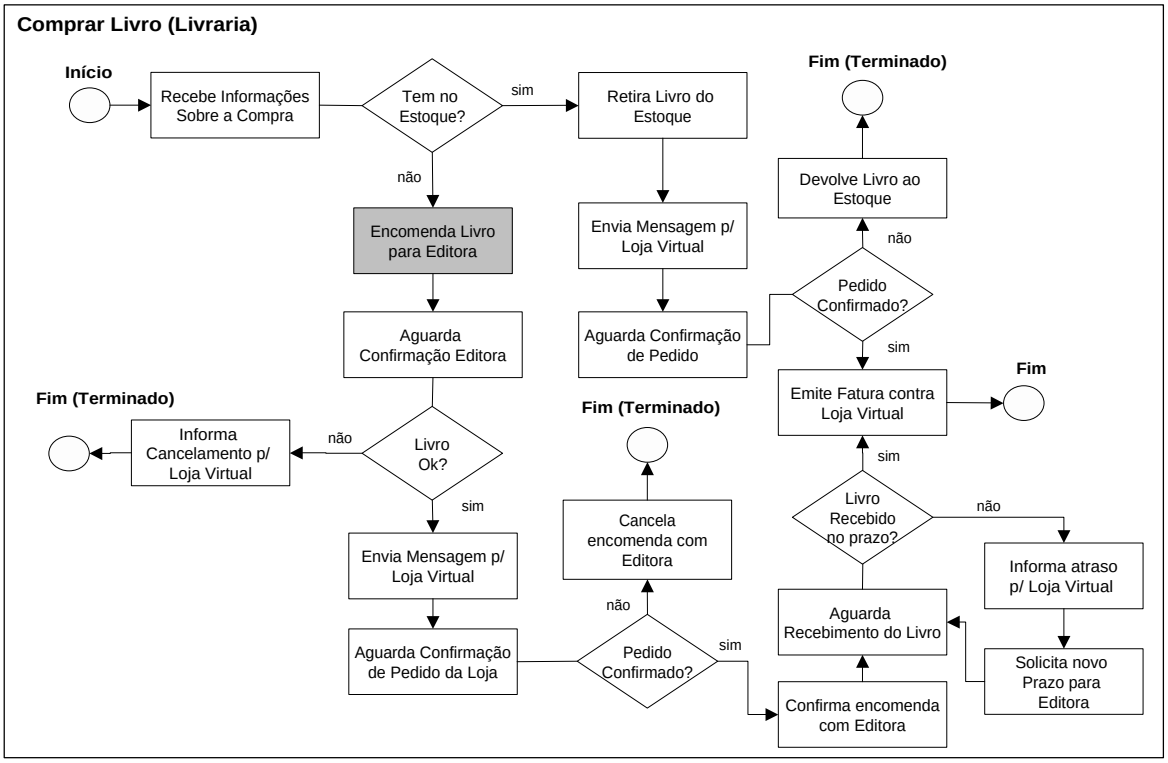

Figura 6.4 – Descrição do processo “Comprar Livro” executado no WFMS da livraria

Antes de detalhar o processo de Entrega de Mercadoria executado no WFMS da transportadora iremos apresentar o processo “Encomendar Livro” que pode ser invocado pelo processo “Comprar Livro” descrito na **Figura 6.4**.

A **Figura 6.5** descreve o processo de “Encomendar Livro” que é executado no WFMS da editora Brasil PRESS caso a livraria MEGABOOKS não tenha o livro pedido em estoque. Vale frisar que este processo é um processo local pois não depende de nenhum subprocesso distribuído para a sua finalização.

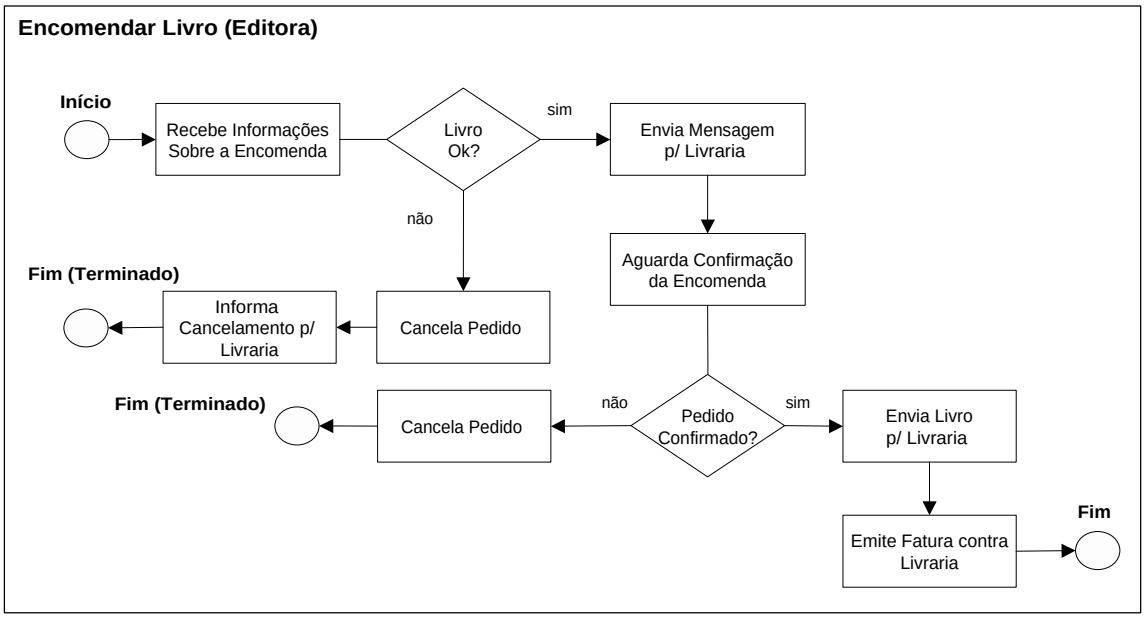

Figura 6.5 – Descrição do processo “Encomendar Livro” executado no WFMS da editora

Após o pedido do livro ser confirmado pelo WFMS da livraria MEGABOOKS o WFMS da loja virtual NetBUY continua a execução do processo “Processar Pedido”, onde há uma atividade de “Confirmação de Pedido” que envia uma mensagem para o WFMS da livraria confirmando o pedido e na sequência há a atividade “Envia pedido de Transporte para Transportadora” que invoca o processo “Entregar Mercadoria” no WFMS da transportadora EXPRESSO Brasil (vide **Figura 6.2**). A **Figura 6.6** apresenta a descrição do processo “Entregar Mercadoria” que é executado no WFMS da transportadora. A empresa transportadora possui uma agenda onde há o controle que informa quando os transportes serão realizados.

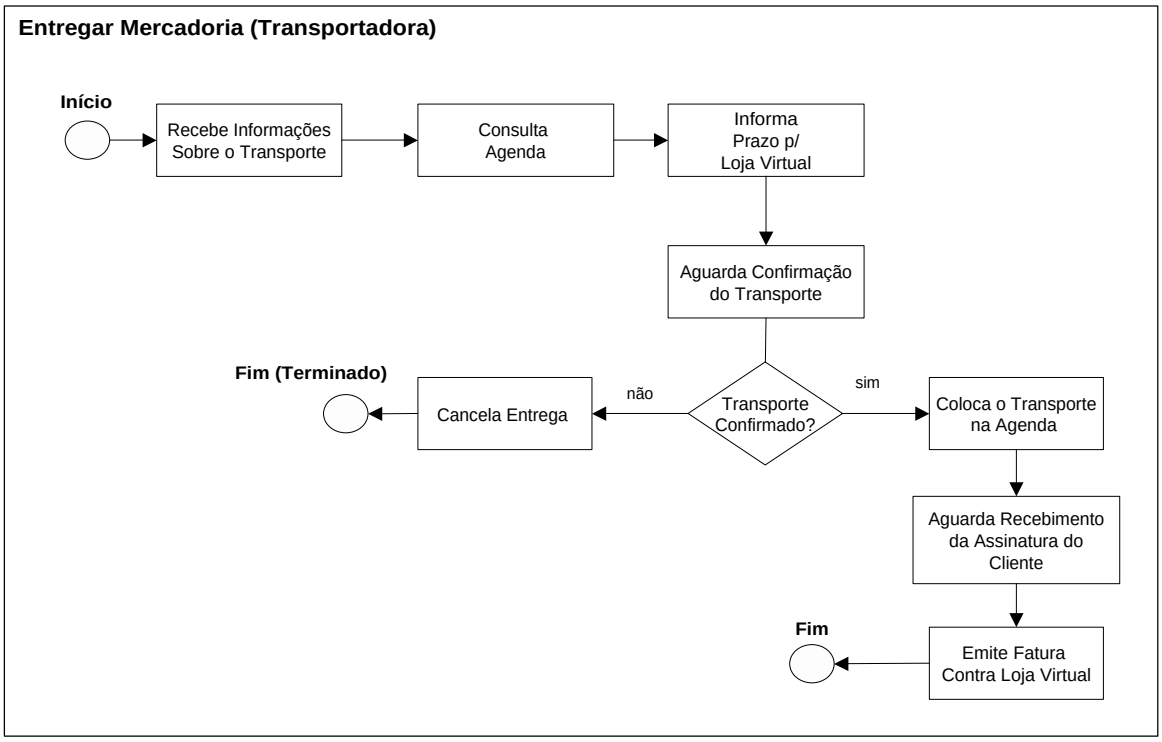

Figura 6.6 – Descrição do processo “Entregar Mercadoria” executado no WFMS da transportadora

Vale frisar que o processo “Entregar Mercadoria” é um processo local pois não invoca nenhum subprocesso distribuído durante sua execução.

6.2.2 Cancelando um Pedido

Conforme descrito anteriormente o cancelamento de um pedido consiste em uma operação oferecida pela loja virtual onde o cliente cancela o seu pedido de compra informando o número identificador do seu pedido. Assim o WFMS (*Workflow Management System*) da loja virtual cancela todo os processos envolvidos com a compra do referido cliente. Para tal o WFMS da NetBuy envia mensagens notificando o cancelamento do(s) subprocesso(s) que poderá(ão) estar sendo executado(s) no WFMS da MEGABOOKS e/ou no WFMS da EXPRESSO Brasil e depois altera o estado do seu processo “Processar Pedido" para cancelado. A notificação de cancelamento do subprocesso que poderá estar sendo executado no WFMS da editora Brasil PRESS é de exclusiva responsabilidade do WFMS da livraria MEGABOOKS, pois em um cenário hierárquico um processo pai é responsável somente pelos seus processos filhos, assim o processo “Encomendar Livro” é filho do processo “Comprar Livro” e este é o responsável por ele. Uma **mensagem de cancelamento** neste cenário pode ser de dois tipos:

- **Aborte**: onde o WFMS pai transmite uma mensagem de *ChangeProcessInstanceState* para mudar o estado do subprocesso filho para “Abortado” (*closed.aborted*) finalizando assim o subprocesso;
- **Compense**: o WFMS pai transmite uma mensagem de *ChangeProcessInstanceState* para mudar o estado do subprocesso filho para “Compensado” (*closed.terminated*). Esse tipo de mensagem é utilizado quando o(s) subprocesso(s) filho(s) já se encontra(m) no estado “Preparado” (*open.prepared*). Ver item 5.6 para maiores detalhes sobre os estados dos processos.

6.2.3 Compensando um Processo

Um processo é compensado quando ele se encontra no estado “preparado” (*open.prepared*) e há uma ordem para que ele altere o seu estado para “compensado” (*closed.terminated*). Quando tal evento ocorre o workflow *engine* do WFMS responsável pelo processo executa o correspondente processo de compensação que é definido para cada processo, ou seja, cada processo definido no sistema deve ter seu referente processo de compensação que será executado para desfazer as modificações executadas pelo processo normal. Vale frisar que a responsabilidade pela definição e execução dos processos de compensação é respectivamente da Linguagem de Definição de Processos utilizada e do WFMS executor do processo. A arquitetura proposta neste trabalho (WS IF4T) é responsável por notificar ao WFMS que o processo deve ser abortado ou compensado, mas a compensação em si, como anteriormente citado é de responsabilidade do WFMS.

A **Figura 6.7** apresenta o processo de compensação do processo “Processar Pedido” da loja virtual. De maneira similar a um processo, os retângulos da figura representam as atividades, que nesse caso são de compensação.

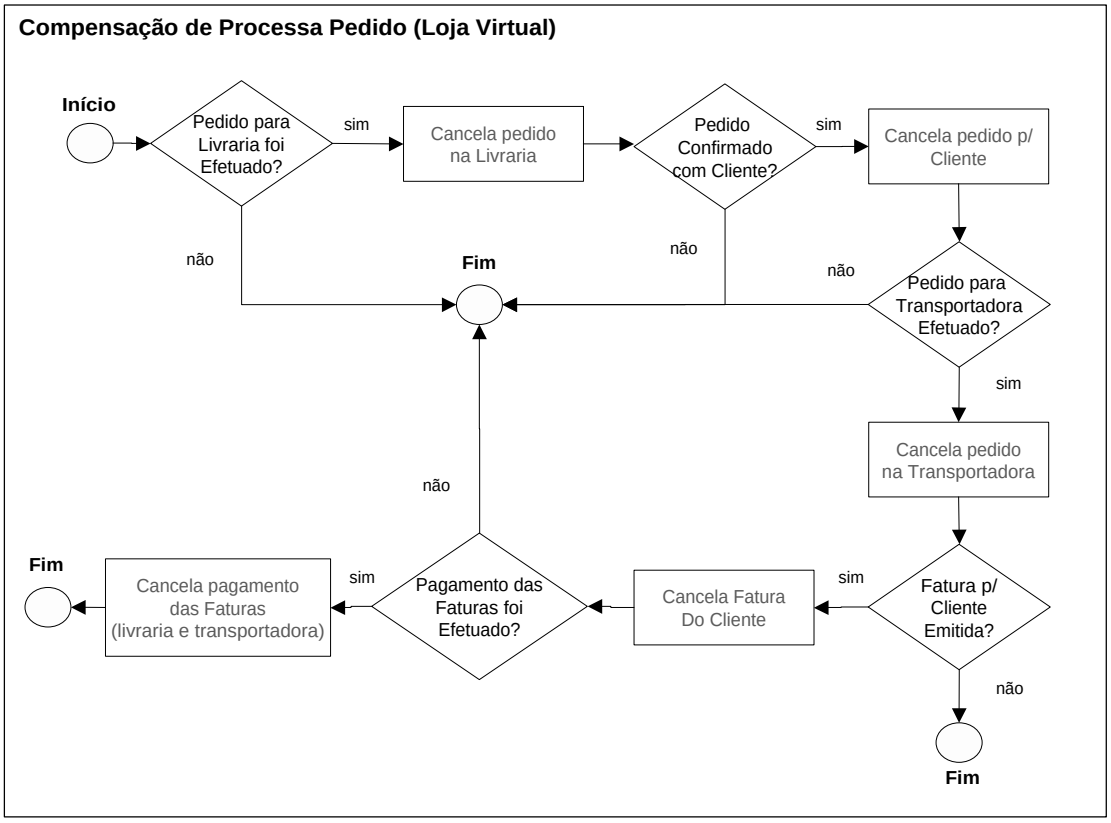

Figura 6.7 – Descrição do processo de Compensação do processo “Processar Pedido”

A seguir será apresentado todos os processos de compensação de todos os demais processos que compõem o cenário proposto neste estudo de caso. A **Figura 6.8** apresenta o processo de compensação de “Comprar Livro” que é executado no WFMS da livraria.

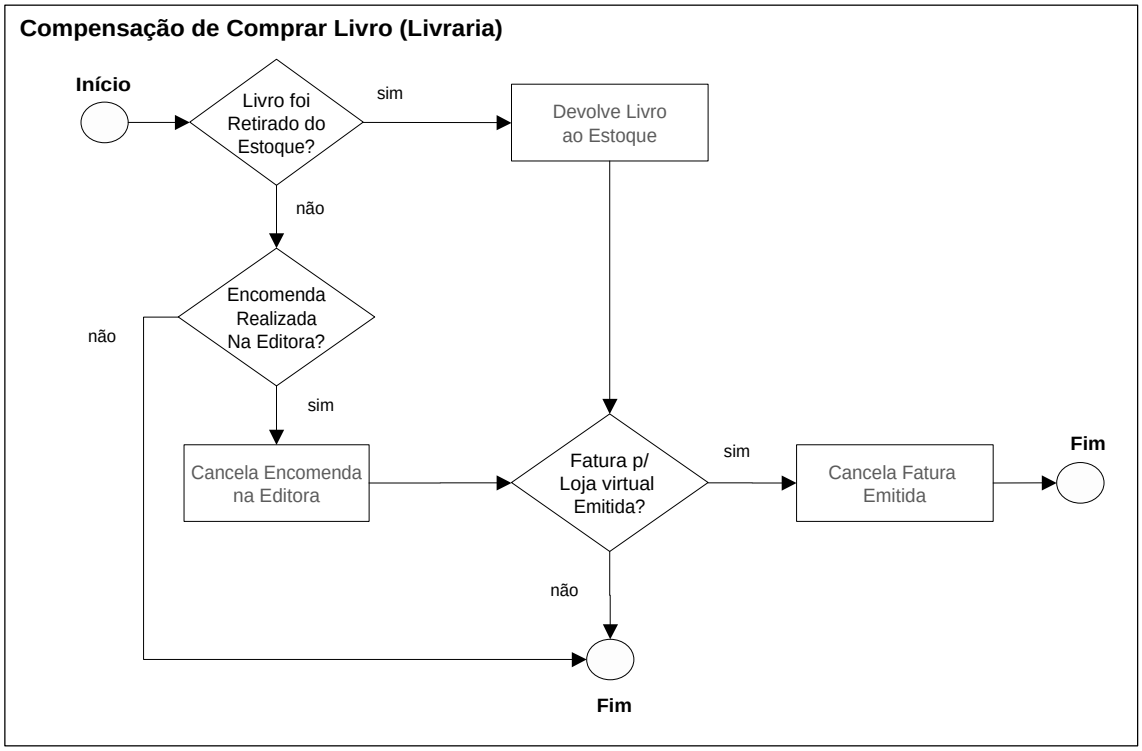

Figura 6.8 – Descrição do processo de Compensação do processo "Comprar Livro"

A **Figura 6.9** apresenta o processo de compensação de "Encomendar Livro" com as suas correspondentes atividades de compensação.

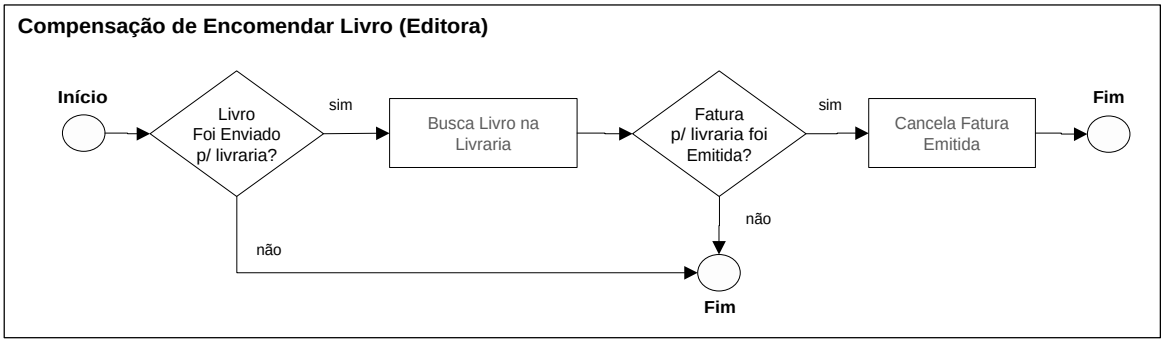

Figura 6.9 – Descrição do processo de Compensação do processo "Encomendar Livro"

A **Figura 6.10** apresenta o processo de compensação de “Entregar Mercadoria” com as suas correspondentes atividades de compensação.

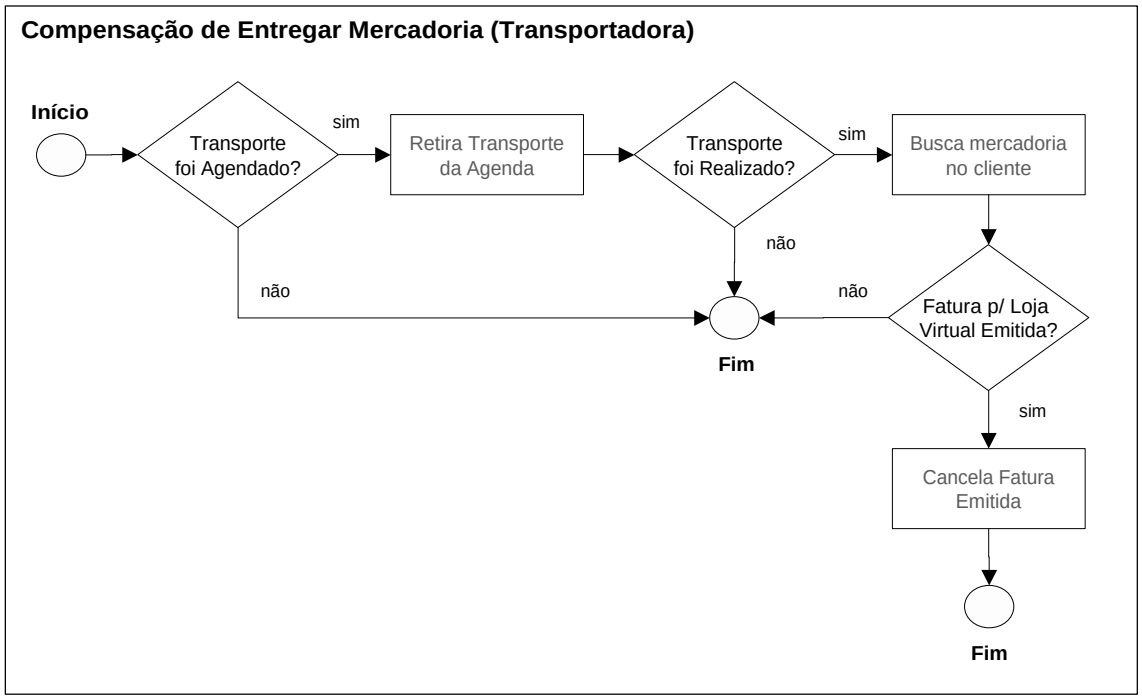

Figura 6.10 – Descrição do processo de Compensação do processo “Entregar Mercadoria”

6.3. Simulando o Cenário

Após se contextualizar o cenário escolhido será realizado uma simulação para validar o funcionamento e os algoritmos propostos na arquitetura WS IF4T. Nesta simulação serão obtidos todos os eventos de operações e troca de mensagens entre os WFMS integrantes do cenário.

A obtenção dos resultados será através de software(s) que simula(m) o funcionamento de um sistema gerenciador de workflow (WFMS) que utilizando a interface WS IF4T para a interoperação distribuída. O software foi implementado em *Borland Delphi* e os códigos fonte se encontram em anexo.

A **Figura 6.11** ilustra a interface gráfica do software que simula um WFMS com WS IF4T.

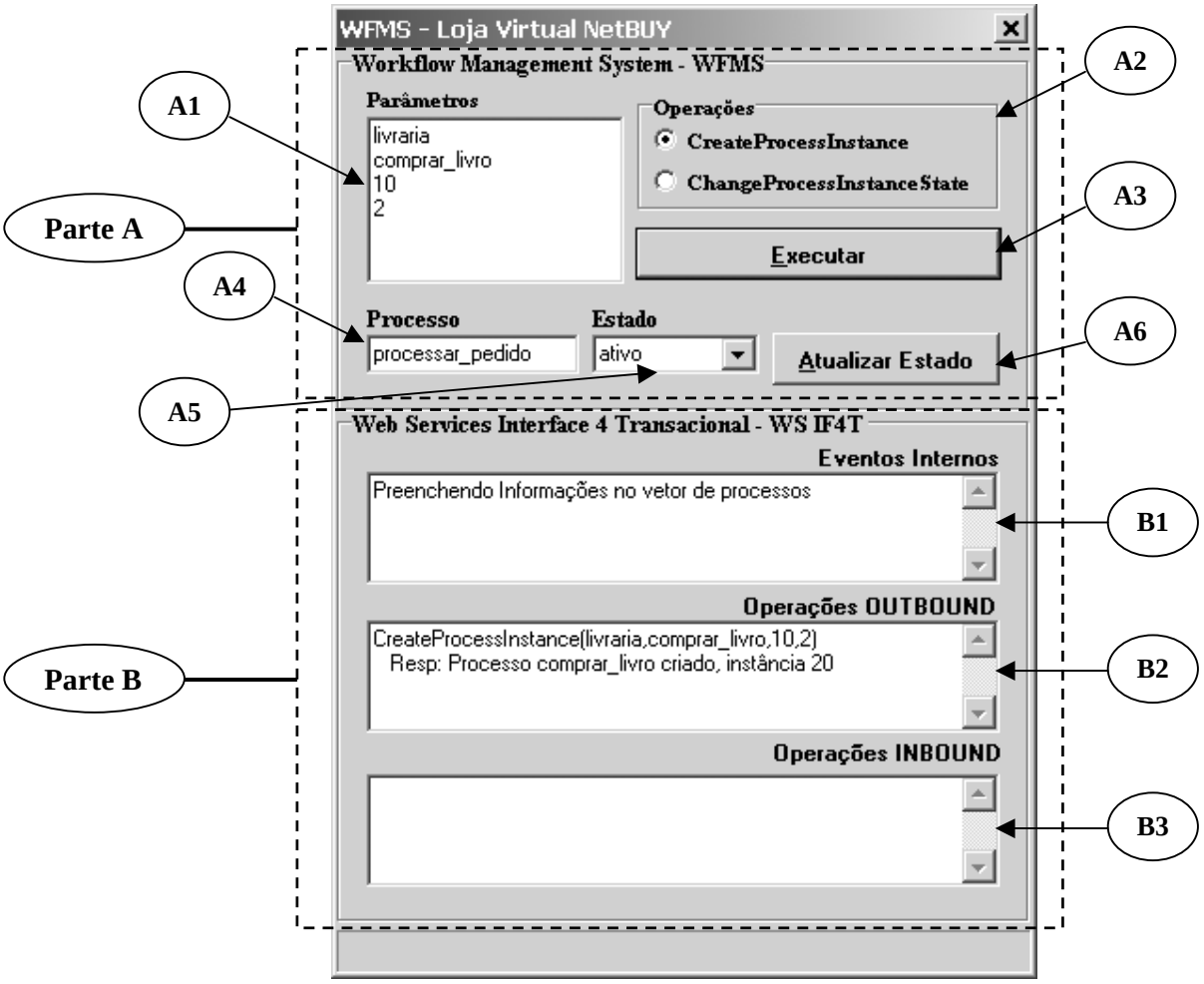

Figura 6.11 – Software utilizado para simulação de um WFMS utilizando WS IF4T

A **Figura 6.11** apresenta vários circulos com letras internas para que correspondem a cada um dos componentes da interface gráfica do software utilizado para simulação. A seguir explicaremos em detalhes a função de cada componente correspondente aos circulos ilustrados na figura:

- **A1:** caixa de texto onde digita-se os parâmetros de entrada para as operações *CreateProcessInstance* e *ChangeProcessInstanceState*. Por exemplo, para a operação *CreateProcessInstance* digita-se cada um dos quatro parâmetros nas quatro primeiras linhas (**livraria** = workflow engine do WFMS de destino, **comprar_livro** = nome da definição de processo a ser instânciada do WFMS de destino, **10** = número da instância do processo pai, nesse caso o número da instância do processo processar_pedido, e **2** = número identificador da atividade do processo pai que está invocando o subprocesso comprar_livro);

- **A2:** componente gráfico para escolher qual será a operação executada, *CreateProcessInstance* ou *ChangeProcessInstanceState*;
- **A3:** botão que realiza a execução das operações escolhidas em A2;
- **A4:** caixa de texto que informa qual o tipo de processo que está sendo executado no WFMS no momento;
- **A5:** componente *combo box* que informa/altera o estado atual do processo mostrado em A4. Esse componente recebe as atualizações de estado do WFMS pai/WS IF4T ou pode servir como meio para realizar alterações internas de estado determinadas pelo seu próprio WFMS;
- **A6:** botão que realiza a atualização do estado que foi alterado no componente A5 quando se tratar de uma alteração de estado do próprio WFMS;
- **B1:** caixa de texto que ilustra todos os eventos ocorridos internamentes no módulo transacional da WS IF4T;
- **B2:** caixa de texto que ilustra todas as operações enviadas da WS IF4T para o meio externo (operações OUTBOUND);
- **B3:** caixa de texto que ilustra todas as operações recebidas pela WS IF4T procedentes do meio externo (operações INBOUND).

A simulação é realizada através da interação de vários WFMS que representam cada uma das organizações descritas no cenário. A **Figura 6.12** ilustra o ambiente de interação entre os vários cenários.

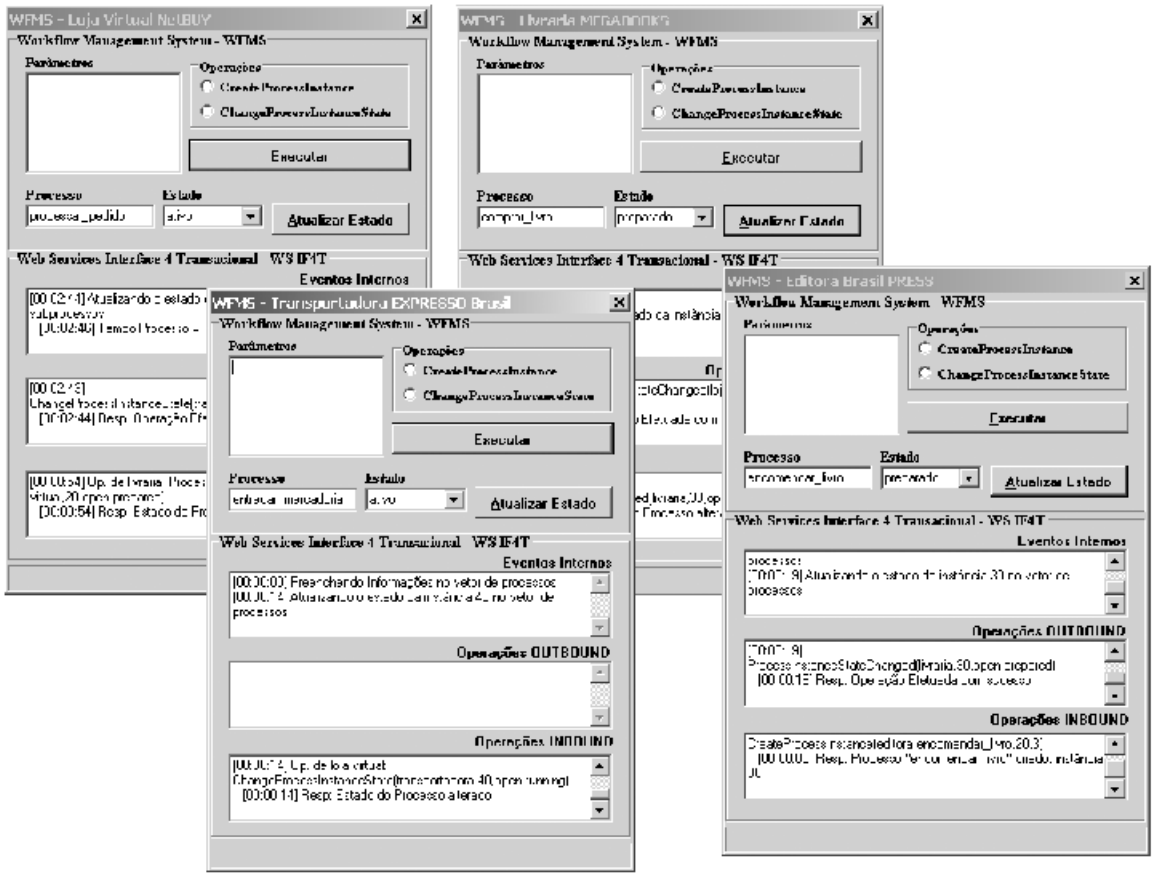

Figura 6.12 – Ambiente de Simulação dos processos

Conforme explicado anteriormente o algoritmo Executor do módulo transacional da WS IF4T tem os seus critérios de funcionamento baseados nos estados e tempos dos processos. Para tanto é necessário que seja mostrado quais serão os tempos dos processos simulados neste cenário. A **Figura 6.13** mostra os processos com os seus respectivos tempos e as suas respectivas instâncias que serão consideradas na simulação.

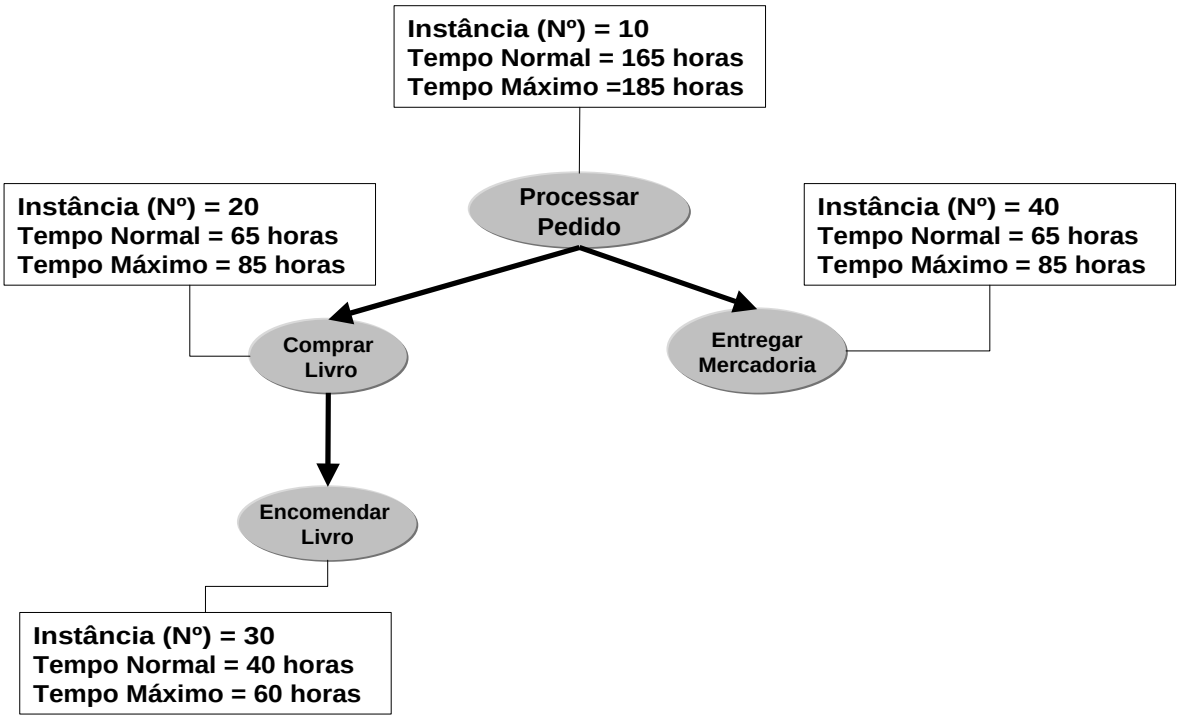

Figura 6.13 – Tempos e Instâncias dos processos simulados

Os tempos e as instâncias da **Figura 6.13** foram estabelecidos para a simulação. Vale frisar que o(s) tempo(s) de conclusão (Normal e Máximo) de um processo pai geralmente é maior que a somatória dos tempos de seus subprocessos filhos pois em tal processo deve-se levar em conta também o tempo gasto para se executar as atividades locais.

A **Tabela 6.1** ilustra as situações consideradas mais relevantes e os resultados que irão ser esperados durante a simulação deste cenário no ambiente de software que implementa a WS IF4T.

Hipótese	**Situação**	**Resultados Esperados**
1º **Execução Normal (A)**: processos foram concluídos com sucesso até o tempo Normal de "Processar Pedido".	• Os processos estarão todos "preparados" quando o algoritmo executor da WS IF4T da loja virtual estiver na etapa em que o tempo atual = tempo Normal.	Espera-se que os processos sejam automaticamente confirmados pelas suas respectivas WS IF4T e assim todos os processos serão confirmados e se encontrarão no estado "completo" (*closed.completed*).
2º **Execução Normal (B)**: processos foram concluídos com sucesso até o tempo Máximo de "Processar Pedido".	• Os processos estarão todos "preparados" quando o algoritmo executor da WS IF4T da loja virtual estiver na etapa em que o tempo atual = tempo Máximo.	Espera-se que os processos sejam automaticamente confirmados pelas suas respectivas WS IF4T e assim todos os processos serão confirmados e se encontrarão no estado "completo" (*closed.completed*).
3º **Execução Anormal (A)**: ocorreu algum erro durante a verificação de tempo atual de "Processar Pedido" = tempo Normal / 2.	• Os processos "Comprar Livro" e "Encomendar livro" se encontram ativos. O processo "Encomendar Livro" será propositalmente abortado antes de haver a verificação de tempo atual = tempo normal / 2 na WS IF4T de "Processar Pedido".	Espera-se que o processo "Comprar Livro" que se encontra ativo seja automaticamete cancelado e que haja a notificação e o cancelamento do processo "Processar Pedido". Assim todos os processos devem ser finalizados no estado "abortado" (*closed.aborted*).
4º **Execução Anormal (B)**: ocorreu um erro durante a verificação de tempo atual de "Processar Pedido" = Tempo Normal.	• Os processos "Comprar Livro" e "Encomendar livro" se encontram prontos e o processo "Entregar Mercadoria" será propositalmente abortado antes de haver a verificação de tempo atual = tempo normal na WS IF4T de "Processar Pedido".	A WS IF4T da Loja virtual aplicará o modelo Sagas que irá compensar todos os processos que anteriormente se encontravam prontos. Assim que o processo "Comprar Livro" for compensado a WS IF4T da livraria deverá compensar automaticamente o seu subprocesso "Encomendar Livro". Todos os processos com exeção de "Entregar Mercadoria" deverão ser finalizados com o seu estado "compensado" (*closed.terminated*).
5º **Execução Anormal (C)**: houve o cancelamento do "Processar Pedido" durante a verificação de Tempo atual = Tempo Máximo e todos os processos já se encontravam "prontos".	• Os processos "Comprar Livro" e "Encomendar livro" e "Entregar Mercadoria" já se encontram prontos, e o processo "Processar Pedido" será cancelado através da operação "Cancelar Pedido" disponível no web site da loja virtual.	A WS IF4T da Loja virtual deverá automaticamente propagar esse cancelamento para todos os seus subprocessos. Assim que o processo "Comprar Livro" for compensado a WS IF4T da livraria deverá compensar automaticamente o seu subprocesso "Encomendar Livro". Todos os processos deverão ser finalizados com o seu estado "compensado" (*closed.terminated*).

Tabela 6.1 – Hipóteses que serão simuladas no cenário do estudo de caso

6.3.1 Simulando Cenário - Execução Normal (A)

Nas tabelas (**Tabela 6.2** e **Tabela 6.3**) são apresentados os resultados obtidos em cada WFMS quando todos os processos são executados com sucesso no tempo Normal previsto. Os eventos em negrito correspondem aos executados pelo algoritmo Executor da WS IF4T.

WFMS – Loja Virtual NetBuy	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:04] Atualizando o estado da instância 20 no vetor de subprocessos **[00:00:04] O processo "processar_pedido" foi alocado no algoritmo executor** [00:00:48] Preenchendo Informações no vetor de processos [00:00:54] Atualizando o estado da instância 40 no vetor de subprocessos **[00:01:26] Tempo Processo = Tempo Normal / 2** [00:01:29] Atualizando o estado da instância 20 no vetor de subprocessos [00:01:38] Atualizando o estado da instância 20 no vetor de subprocessos [00:01:50] Atualizando o estado da instância 40 no vetor de subprocessos **[00:02:49] Tempo Processo = Tempo Normal** **[00:02:49] O Processo do Algoritmo Executor está sendo confirmado** [00:02:49] Atualizando o estado da instância 20 no vetor de subprocessos [00:02:50] Atualizando o estado da instância 40 no vetor de subprocessos **[00:02:50] O Algoritmo Executor foi Finalizado**
Operações OUTBOUND	[00:00:00] CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:04] ChangeProcessInstanceState(livraria,20,open.running) [00:00:04] Resp: Operação Efetuada com sucesso [00:00:47] CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:00:48] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:00:54] ChangeProcessInstanceState(transportadora,40,open.running) [00:00:54] Resp: Operação Efetuada com sucesso **[00:02:49] ChangeProcessInstanceState(livraria,20,closed.completed)** [00:02:49] Resp: Operação Efetuada com sucesso **[00:02:49] ChangeProcessInstanceState(transportadora,40,closed.completed)** [00:02:50] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:01:29] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:29] Resp: Estado do Processo alterado [00:01:38] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:01:38] Resp: Estado do Processo alterado [00:01:50] Op. de transportadora: ProcessInstanceStateChanged(loja virtual,40,open.prepared) [00:01:50] Resp: Estado do Processo alterado

Tabela 6.2 – Resultados obtidos com a Execução Normal (A) (parte 1)

WFMS – Livraria MEGABOOKS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:04] Atualizando o estado da instância 20 no vetor de processos [00:00:04] Atualizando o estado dos subprocessos distribuídos [00:00:16] Preenchendo Informações no vetor de processos [00:00:24] Atualizando o estado da instância 30 no vetor de subprocessos **[00:00:24] O processo "comprar_livro" foi alocado no algoritmo executor** **[00:00:56] Tempo Processo = Tempo Normal / 2** [00:01:04] Atualizando o estado da instância 30 no vetor de subprocessos **[00:01:29] Tempo Processo = Tempo Normal** **[00:01:29] O Processo do Algoritmo Executor está sendo confirmado** [00:01:29] Atualizando o estado da instância 20 no vetor de processos [**00:01:29] Aguardando confirmação do processo "processar_pedido"** [00:01:29] O Algoritmo Executor foi Finalizado [00:01:39] Atualizando o estado da instância 20 no vetor de processos [00:02:49] Atualizando o estado da instância 30 no vetor de subprocessos **[00:02:49] Atualizando o estado dos subprocessos distribuídos**
Operações OUTBOUND	[00:00:16] CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:16] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:23] ChangeProcessInstanceState(editora,30,open.running) [00:00:24] Resp: Operação Efetuada com sucesso [00:01:29] ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:29] Resp: Operação Efetuada com sucesso [00:01:38] ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:01:39] Resp: Operação Efetuada com sucesso **[00:02:49] ChangeProcessInstanceState(editora,30,closed.completed)** [00:02:49] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:04] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,open.running) [00:00:04] Resp: Estado do Processo alterado [00:01:04] Op. de editora: ProcessInstanceStateChanged(livraria,30,open.prepared) [00:01:04] Resp: Estado do Processo alterado **[00:02:49] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,closed.completed)** [00:02:49] Resp: Estado do Processo alterado

WFMS – Editora Brasil PRESS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:07] Atualizando o estado da instância 30 no vetor de processos [00:00:47] Atualizando o estado da instância 30 no vetor de processos [00:02:32] Atualizando o estado da instância 30 no vetor de processos
Operações OUTBOUND	[00:00:46] ProcessInstanceStateChanged(livraria,30,open.prepared) [00:00:47] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de livraria: CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:00] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:07] Op. de livraria: ChangeProcessInstanceState(editora,30,open.running) [00:00:07] Resp: Estado do Processo alterado **[00:02:32] Op. de livraria: ChangeProcessInstanceState(editora,30,closed.completed)** [00:02:32] Resp: Estado do Processo alterado

WFMS – Transportadora EXPRESSO Brasil	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 40 no vetor de processos [00:01:02] Atualizando o estado da instância 40 no vetor de processos [00:02:01] Atualizando o estado da instância 40 no vetor de processos
Operações OUTBOUND	[00:01:01] ProcessInstanceStateChanged(loja virtual,40,open.prepared) [00:01:02] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:00:00] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:00:06] Op. de loja virtual: ChangeProcessInstanceState(transportadora,40,open.running) [00:00:06] Resp: Estado do Processo alterado **[00:02:01]Op. de loja virtual: ChangeProcessInstanceState(transportadora,40,closed.completed)** [00:02:01] Resp: Estado do Processo alterado

Tabela 6.3 – Resultados obtidos com a Execução Normal (A) (parte 2)

6.3.2 Simulando Cenário – Execução Normal (B)

Nas tabelas (**Tabela 6.4** e **Tabela 6.5**) são apresentados os resultados obtidos em cada WFMS quando todos os processos são executados com sucesso no tempo Máximo previsto. Os eventos em negrito correspondem aos executados pelo algoritmo Executor da WS IF4T.

WFMS – Loja Virtual NetBuy	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 20 no vetor de subprocessos **[00:00:06] O processo "processar_pedido" foi alocado no algoritmo executor** [00:00:44] Preenchendo Informações no vetor de processos [00:00:53] Atualizando o estado da instância 40 no vetor de subprocessos **[00:01:28] Tempo Processo = Tempo Normal / 2** [00:01:50] Atualizando o estado da instância 20 no vetor de subprocessos **[00:02:51] Tempo Processo = Tempo Normal** [00:02:56] Atualizando o estado da instância 20 no vetor de subprocessos [00:02:59] Atualizando o estado da instância 40 no vetor de subprocessos **[00:03:11] Tempo Processo = Tempo Máximo** [00:03:11] O Processo do Algoritmo Executor está sendo confirmado [00:03:11] Atualizando o estado da instância 20 no vetor de subprocessos [00:03:11] Atualizando o estado da instância 40 no vetor de subprocessos **[00:03:11] O Algoritmo Executor foi Finalizado**
Operações OUTBOUND	[00:00:00] CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:05] ChangeProcessInstanceState(livraria,20,open.running) [00:00:06] Resp: Operação Efetuada com sucesso [00:00:43] CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:00:44] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:00:53] ChangeProcessInstanceState(transportadora,40,open.running) [00:00:53] Resp: Operação Efetuada com sucesso **[00:03:11] ChangeProcessInstanceState(livraria,20,closed.completed)** [00:03:11] Resp: Operação Efetuada com sucesso **[00:03:11] ChangeProcessInstanceState(transportadora,40,closed.completed)** [00:03:11] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:01:50] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:50] Resp: Estado do Processo alterado [00:02:56] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:02:56] Resp: Estado do Processo alterado [00:02:59] Op. de transportadora: ProcessInstanceStateChanged(loja virtual,40,open.prepared) [00:02:59] Resp: Estado do Processo alterado

Tabela 6.4 – Resultados obtidos com a Execução Normal (B) (parte 1)

WFMS – Livraria MEGABOOKS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 20 no vetor de processos [00:00:06] Atualizando o estado dos subprocessos distribuídos [00:00:18] Preenchendo Informações no vetor de processos [00:00:24] Atualizando o estado da instância 30 no vetor de subprocessos **[00:00:24] O processo "comprar_livro" foi alocado no algoritmo executor** **[00:00:56] Tempo Processo = Tempo Normal / 2** **[00:01:29] Tempo Processo = Tempo Normal** [00:01:34] Atualizando o estado da instância 30 no vetor de subprocessos **[00:01:49] Tempo Processo = Tempo Máximo** [00:01:49] O Processo do Algoritmo Executor está sendo confirmado [00:01:50] Atualizando o estado da instância 20 no vetor de processos [00:01:50] Aguardando confirmação do processo "processar_pedido" [00:01:50] O Algoritmo Executor foi Finalizado [00:02:56] Atualizando o estado da instância 20 no vetor de processos [00:03:11] Atualizando o estado da instância 20 no vetor de processos [00:03:11] Atualizando o estado da instância 30 no vetor de subprocessos [00:03:11] Atualizando o estado dos subprocessos distribuídos
Operações OUTBOUND	[00:00:17] CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:18] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:24] ChangeProcessInstanceState(editora,30,open.running) [00:00:24] Resp: Operação Efetuada com sucesso [00:01:49] ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:50] Resp: Operação Efetuada com sucesso [00:02:56] ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:02:56] Resp: Operação Efetuada com sucesso **[00:03:11] ChangeProcessInstanceState(editora,30,closed.completed)** [00:03:11] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:06] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,open.running) [00:00:06] Resp: Estado do Processo alterado [00:01:34] Op. de editora: ProcessInstanceStateChanged(livraria,30,open.prepared) [00:01:34] Resp: Estado do Processo alterado **[00:03:11] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,closed.completed)** [00:03:11] Resp: Estado do Processo alterado
WFMS – Editora Brasil PRESS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 30 no vetor de processos [00:01:16] Atualizando o estado da instância 30 no vetor de processos [00:02:53] Atualizando o estado da instância 30 no vetor de processos
Operações OUTBOUND	[00:01:16] ProcessInstanceStateChanged(livraria,30,open.prepared) [00:01:16] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de livraria: CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:00] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:06] Op. de livraria: ChangeProcessInstanceState(editora,30,open.running) [00:00:06] Resp: Estado do Processo alterado **[00:02:53] Op. de livraria: ChangeProcessInstanceState(editora,30,closed.completed)** [00:02:53] Resp: Estado do Processo alterado
WFMS – Transportadora EXPRESSO Brasil	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:08] Atualizando o estado da instância 40 no vetor de processos [00:02:15] Atualizando o estado da instância 40 no vetor de processos [00:02:27] Atualizando o estado da instância 40 no vetor de processos
Operações OUTBOUND	[00:02:14] ProcessInstanceStateChanged(loja virtual,40,open.prepared) [00:02:15] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:00:00] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:00:08] Op. de loja virtual: ChangeProcessInstanceState(transportadora,40,open.running) [00:00:08] Resp: Estado do Processo alterado **[00:02:27]Op. de loja virtual: ChangeProcessInstanceState(transportadora,40,closed.completed)** [00:02:27] Resp: Estado do Processo alterado

Tabela 6.5 – Resultados obtidos com a Execução Normal (B) (parte 2)

6.3.3 Simulando Cenário - Execução Anormal (A)

Nas tabelas (**Tabela 6.6** e **Tabela 6.7**) são apresentados os resultados obtidos para cada WFMS quando ocorreu algum erro durante a verificação de tempo atual de "Processar Pedido" = Tempo Normal / 2.

WFMS – Loja Virtual NetBuy	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 20 no vetor de subprocessos [00:00:06] O processo "processar_pedido" foi alocado no algoritmo executor [00:00:55] Atualizando o estado da instância 20 no vetor de subprocessos **[00:01:28] Tempo Processo = Tempo Normal / 2** **[00:01:28] Os Subprocessos do Processo foram abortados ou compensados** **[00:01:28] O processo do Algoritmo Executor foi abortado** **[00:01:28] O Algoritmo Executor foi Finalizado**
Operações OUTBOUND	[00:00:00] CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:06] ChangeProcessInstanceState(livraria,20,open.running) [00:00:06] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:55] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,closed.aborted) [00:00:55] Resp: Estado do Processo alterado

Tabela 6.6 – Resultados obtidos com a Execução Anormal (A) (parte 1)

WFMS – Livraria MEGABOOKS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 20 no vetor de processos [00:00:06] Atualizando o estado dos subprocessos distribuídos [00:00:15] Preenchendo Informações no vetor de processos [00:00:23] Atualizando o estado da instância 30 no vetor de subprocessos **[00:00:23] O processo "comprar_livro" foi alocado no algoritmo executor** [00:00:43] Atualizando o estado da instância 30 no vetor de subprocessos [00:00:55] Tempo Processo = Tempo Normal / 2 [00:00:55] Os Subprocessos do Processo foram abortados ou compensados [00:00:55] Atualizando o estado da instância 20 no vetor de processos **[00:00:55] O processo do Algoritmo Executor foi abortado** **[00:00:55] O Algoritmo Executor foi Finalizado**
Operações OUTBOUND	[00:00:14] CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:15] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:23] ChangeProcessInstanceState(editora,30,open.running) [00:00:23] Resp: Operação Efetuada com sucesso [00:00:55] ProcessInstanceStateChanged(loja virtual,20,closed.aborted) [00:00:55] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:06] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,open.running) [00:00:06] Resp: Estado do Processo alterado **[00:00:43] Op. de editora: ProcessInstanceStateChanged(livraria,30,closed.aborted)** [00:00:43] Resp: Estado do Processo alterado

WFMS – Editora Brasil PRESS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:07] Atualizando o estado da instância 30 no vetor de processos [00:00:28] Atualizando o estado da instância 30 no vetor de processos
Operações OUTBOUND	[00:00:28] ProcessInstanceStateChanged(livraria,30,closed.aborted) [00:00:28] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de livraria: CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:00] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:07] Op. de livraria: ChangeProcessInstanceState(editora,30,open.running) [00:00:07] Resp: Estado do Processo alterado

WFMS – Transportadora EXPRESSO Brasil	
Eventos Internos	NÃO FOI ATIVADO
Operações OUTBOUND	NÃO FOI ATIVADO
Operações INBOUND	NÃO FOI ATIVADO

Tabela 6.7 – Resultados obtidos com a Execução Anormal (A) (parte 2)

6.3.4 Simulando Cenário - Execução Anormal (B)

Nas tabelas (**Tabela 6.8** e **Tabela 6.9**) são apresentados os resultados obtidos em cada WFMS quando ocorreu algum erro durante a verificação de tempo atual de "Processar Pedido" = Tempo Normal.

WFMS – Loja Virtual NetBuy	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:08] Atualizando o estado da instância 20 no vetor de subprocessos **[00:00:08] O processo "processar_pedido" foi alocado no algoritmo executor** [00:01:27] Atualizando o estado da instância 20 no vetor de subprocessos **[00:01:30] Tempo Processo = Tempo Normal / 2** [00:01:39] Atualizando o estado da instância 20 no vetor de subprocessos [00:01:44] Preenchendo Informações no vetor de processos [00:01:51] Atualizando o estado da instância 40 no vetor de subprocessos [00:01:58] Atualizando o estado da instância 40 no vetor de subprocessos **[00:02:53] Tempo Processo = Tempo Normal** [00:02:53] Atualizando o estado da instância 20 no vetor de subprocessos **[00:02:53] Os Subprocessos do Processo foram abortados ou compensados** **[00:02:53] O processo do Algoritmo Executor foi abortado** **[00:02:53] O Algoritmo Executor foi Finalizado**
Operações OUTBOUND	[00:00:00] CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:07] ChangeProcessInstanceState(livraria,20,open.running) [00:00:08] Resp: Operação Efetuada com sucesso [00:01:43] CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:01:44] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:01:50] ChangeProcessInstanceState(transportadora,40,open.running) [00:01:51] Resp: Operação Efetuada com sucesso **[00:02:53] ChangeProcessInstanceState(livraria,20,closed.terminated)** [00:02:53] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:01:27] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:27] Resp: Estado do Processo alterado [00:01:39] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:01:39] Resp: Estado do Processo alterado [00:01:58] Op. de transportadora: ProcessInstanceStateChanged(loja virtual,40,closed.aborted) [00:01:58] Resp: Estado do Processo alterado

Tabela 6.8 – Resultados obtidos com a Execução Anormal (B) (parte 1)

WFMS – Livraria MEGABOOKS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:08] Atualizando o estado da instância 20 no vetor de processos [00:00:08] Atualizando o estado dos subprocessos distribuídos [00:00:16] Preenchendo Informações no vetor de processos [00:00:22] Atualizando o estado da instância 30 no vetor de subprocessos **[00:00:22] O processo "comprar_livro" foi alocado no algoritmo executor** **[00:00:54] Tempo Processo = Tempo Normal / 2** [00:01:01] Atualizando o estado da instância 30 no vetor de subprocessos **[00:01:27] Tempo Processo = Tempo Normal** [00:01:27] O Processo do Algoritmo Executor está sendo confirmado [00:01:27] Atualizando o estado da instância 20 no vetor de processos **[00:01:27] Aguardando confirmação do processo "processar_pedido"** **[00:01:27] O Algoritmo Executor foi Finalizado** [00:01:39] Atualizando o estado da instância 20 no vetor de processos [00:02:53] Atualizando o estado da instância 20 no vetor de processos [00:02:53] Atualizando o estado da instância 30 no vetor de subprocessos [00:02:53] Atualizando o estado dos subprocessos distribuídos
Operações OUTBOUND	[00:00:15] CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:16] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:22] ChangeProcessInstanceState(editora,30,open.running) [00:00:22] Resp: Operação Efetuada com sucesso [00:01:27] ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:27] Resp: Operação Efetuada com sucesso [00:01:39] ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:01:39] Resp: Operação Efetuada com sucesso **[00:02:53] ChangeProcessInstanceState(editora,30,closed.terminated)** [00:02:53] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:08] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,open.running) [00:00:08] Resp: Estado do Processo alterado [00:01:01] Op. de editora: ProcessInstanceStateChanged(livraria,30,open.prepared) [00:01:01] Resp: Estado do Processo alterado **[00:02:53] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,closed.terminated)** [00:02:53] Resp: Estado do Processo alterado

WFMS – Editora Brasil PRESS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 30 no vetor de processos [00:00:44] Atualizando o estado da instância 30 no vetor de processos [00:02:37] Atualizando o estado da instância 30 no vetor de processos
Operações OUTBOUND	[00:00:44] ProcessInstanceStateChanged(livraria,30,open.prepared) [00:00:44] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de livraria: CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:00] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:06] Op. de livraria: ChangeProcessInstanceState(editora,30,open.running) [00:00:06] Resp: Estado do Processo alterado **[00:02:37] Op. de livraria: ChangeProcessInstanceState(editora,30,closed.terminated)** [00:02:37] Resp: Estado do Processo alterado

WFMS – Transportadora EXPRESSO Brasil	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 40 no vetor de processos [00:00:14] Atualizando o estado da instância 40 no vetor de processos
Operações OUTBOUND	**[00:00:13] ProcessInstanceStateChanged(loja virtual,40,closed.aborted)** [00:00:14] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:00:00] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:00:06] Op. de loja virtual: ChangeProcessInstanceState(transportadora,40,open.running) [00:00:06] Resp: Estado do Processo alterado

Tabela 6.9 – Resultados obtidos com a Execução Anormal (B) (parte 2)

6.3.5 Simulando Cenário - Execução Anormal (C)

Nas tabelas (**Tabela 6.10** e **Tabela 6.11**) são apresentados os resultados obtidos para cada WFMS quando ocorreu algum erro durante a verificação de tempo atual de "Processar Pedido" = Tempo Máximo.

WFMS – Loja Virtual NetBuy	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 20 no vetor de subprocessos **[00:00:06] O processo "processar_pedido" foi alocado no algoritmo executor** [00:01:27] Atualizando o estado da instância 20 no vetor de subprocessos [00:01:28] Tempo Processo = Tempo Normal / 2 [00:01:31] Atualizando o estado da instância 20 no vetor de subprocessos [00:01:43] Preenchendo Informações no vetor de processos [00:01:48] Atualizando o estado da instância 40 no vetor de subprocessos **[00:02:51] Tempo Processo = Tempo Normal** [00:02:55] Atualizando o estado da instância 40 no vetor de subprocessos **[00:03:11] Tempo Processo = Tempo Máximo** [00:03:14] Atualizando o estado da instância 40 no vetor de subprocessos [00:03:14] Atualizando o estado da instância 20 no vetor de subprocessos [00:03:14] Os Subprocessos do Processo foram abortados ou compensados **[00:03:17] O Algoritmo Executor foi Finalizado**
Operações OUTBOUND	[00:00:00] CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:05] ChangeProcessInstanceState(livraria,20,open.running) [00:00:06] Resp: Operação Efetuada com sucesso [00:01:41] CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:01:43] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:01:48] ChangeProcessInstanceState(transportadora,40,open.running) [00:01:48] Resp: Operação Efetuada com sucesso **[00:03:14] ChangeProcessInstanceState(transportadora,40,closed.terminated)** [00:03:14] Resp: Operação Efetuada com sucesso **[00:03:14] ChangeProcessInstanceState(livraria,20,closed.terminated)** [00:03:14] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:01:27] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:27] Resp: Estado do Processo alterado [00:01:31] Op. de livraria: ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:01:31] Resp: Estado do Processo alterado [00:02:55] Op. de transportadora: ProcessInstanceStateChanged(loja virtual,40,open.prepared) [00:02:55] Resp: Estado do Processo alterado

Tabela 6.10 – Resultados obtidos com a Execução Anormal (C) (parte 1)

WFMS – Livraria MEGABOOKS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 20 no vetor de processos [00:00:06] Atualizando o estado dos subprocessos distribuídos [00:00:16] Preenchendo Informações no vetor de processos [00:00:22] Atualizando o estado da instância 30 no vetor de subprocessos **[00:00:22] O processo "comprar_livro" foi alocado no algoritmo executor** **[00:00:54] Tempo Processo = Tempo Normal / 2** [00:00:58] Atualizando o estado da instância 30 no vetor de subprocessos **[00:01:27] Tempo Processo = Tempo Normal** [00:01:27] O Processo do Algoritmo Executor está sendo confirmado [00:01:27] Atualizando o estado da instância 20 no vetor de processos **[00:01:27] Aguardando confirmação do processo "processar_pedido"** **[00:01:27] O Algoritmo Executor foi Finalizado** [00:01:31] Atualizando o estado da instância 20 no vetor de processos [00:03:14] Atualizando o estado da instância 20 no vetor de processos [00:03:14] Atualizando o estado da instância 30 no vetor de subprocessos [00:03:14] Atualizando o estado dos subprocessos distribuídos
Operações OUTBOUND	[00:00:15] CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:16] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:22] ChangeProcessInstanceState(editora,30,open.running) [00:00:22] Resp: Operação Efetuada com sucesso [00:01:27] ProcessInstanceStateChanged(loja virtual,20,open_prepared) [00:01:27] Resp: Operação Efetuada com sucesso [00:01:31] ProcessInstanceStateChanged(loja virtual,20,open.prepared) [00:01:31] Resp: Operação Efetuada com sucesso **[00:03:14] ChangeProcessInstanceState(editora,30,closed.terminated)** [00:03:14] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(livraria,comprar_livro,10,2) [00:00:00] Resp: Processo "comprar_livro" criado, instância 20 [00:00:06] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,open.running) [00:00:06] Resp: Estado do Processo alterado [00:00:58] Op. de editora: ProcessInstanceStateChanged(livraria,30,open.prepared) [00:00:58] Resp: Estado do Processo alterado **[00:03:14] Op. de loja virtual: ChangeProcessInstanceState(livraria,20,closed.terminated)** [00:03:14] Resp: Estado do Processo alterado
WFMS – Editora Brasil PRESS	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:06] Atualizando o estado da instância 30 no vetor de processos [00:00:42] Atualizando o estado da instância 30 no vetor de processos [00:02:58] Atualizando o estado da instância 30 no vetor de processos
Operações OUTBOUND	[00:00:42] ProcessInstanceStateChanged(livraria,30,open.prepared) [00:00:42] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de livraria: CreateProcessInstance(editora,encomendar_livro,20,2) [00:00:00] Resp: Processo "encomendar_livro" criado, instância 30 [00:00:06] Op. de livraria: ChangeProcessInstanceState(editora,30,open.running) [00:00:06] Resp: Estado do Processo alterado **[00:02:58] Op. de livraria: ChangeProcessInstanceState(editora,30,closed.terminated)** [00:02:58] Resp: Estado do Processo alterado
WFMS – Transportadora EXPRESSO Brasil	
Eventos Internos	[00:00:00] Preenchendo Informações no vetor de processos [00:00:04] Atualizando o estado da instância 40 no vetor de processos [00:01:11] Atualizando o estado da instância 40 no vetor de processos [00:01:30] Atualizando o estado da instância 40 no vetor de processos
Operações OUTBOUND	[00:01:10] ProcessInstanceStateChanged(loja virtual,40,open.prepared) [00:01:11] Resp: Operação Efetuada com sucesso
Operações INBOUND	[00:00:00] Op. de loja virtual: CreateProcessInstance(transportadora,entregar_mercadoria,10,5) [00:00:00] Resp: Processo "entregar_mercadoria" criado, instância 40 [00:00:04] Op. de loja virtual: ChangeProcessInstanceState(transportadora,40,open.running) [00:00:04] Resp: Estado do Processo alterado **[00:01:30]Op. de loja virtual:ChangeProcessInstanceState(transportadora,40,closed.terminated)** [00:01:30] Resp: Estado do Processo alterado

Tabela 6.11 – Resultados obtidos com a Execução Anormal (C) (parte 2)

6.3.6 Conclusão

Pode-se observar que em todas as situações simuladas obteve-se os resultados esperados de acordo com Tabela 6.1 confirmando assim o funcionamento dos algoritmos de controle propostos na WS IF4T.

A WS IF4T mostrou-se uma opção viável para coordenar a execução de processos distribuídos, garantindo que todos os processos retornem sempre a um estado concistente e que o processo como um todo seja tratado de forma atômica, ou seja, ou o processo é inteiramente executado com sucesso ou as ações de compensação/aborte são efetuadas para garantir que os efeitos do processo sejam desfeitos/anulados.

6.3.7 Resumo

O capítulo em questão apresentou um cenário para a aplicação e validação da arquitetura proposta que oferece um sistema de workflow transacional distribuído utilizando uma interface de interoperabilidade implementada em web services.

Nos itens 6.1 até 6.2.3 apresentou-se o cenário escolhido para o estudo de caso.

Nos itens 6.3 até 6.3.5 realizou-se simulações para obter-se os resultados e validar o funcionamento da WS IF4T.

Capítulo 7

CONCLUSÃO

A WS IF4T apresentou-se como uma proposta de fácil adaptação para sistemas de workflow participantes de uma cadeia de valores. Isso se deve ao fato da WS IF4T ser modular e necessitar apenas de uma API para ser integrada ao sistema já existente (ver itens 5.2 e 5.10.7). Tal característica faz da WS IF4T uma solução viável para tornar um sistema de workflow comum em um sistema de workflow transacional, o que é uma característica bem relevante, pois há uma demanda crescente por aplicações que sejam capazes de implementar um processo de negócio como uma transação.

A WS IF4T utilizou os padrões de interoperação definidos na interface 4 abstrata que foi elaborada pelo WFMC. A interface 4 abstrata serviu de base para todas as interfaces WFMC de interoperabilidade (Wf-XML, XML-http, etc). Portanto a adaptação da WS IF4T em sistemas que utilizam as interfaces de interoperabilidade WFMC não exigirá modificações drásticas pois todas se fundamentaram na mesma origem (interface 4 abstrata).

Os algoritmos internos e a especificação da interface web services da WS IF4T atenderam as especificações desejadas, pois se implementou um simulador que utilizou uma interface de comunicação web services e uma lógica interna herdada dos algoritmos propostos na WS IF4T, o que possibilitou a validação do funcionamento da WS IF4T interoperando em um cenário Hierárquico (*Nested Subprocesses*) e realizando o controle transacional segundo a lógica do modelo transacional adotado nos algoritmos (modelo SAGAS). Isto foi comprovado através dos resultados obtidos (ver iten 6.3) em todas as hipóteses simuladas na Tabela 6.1. Com isso pode-se afirmar que todos objetivos propostos neste trabalho (ver item **1.2**) foram alcançados.

7.1. Contribuições

A interface WS IF4T preenche algumas lacunas apresentadas pelos modelos de workflow transacionais existentes. A WS IF4T apresenta uma interface de interoperação padronizada compatível com os padrões estabelecidos pelo principal orgão normatizador de workflow que é o WFMC.

A proposta deste trabalho também oferece uma solução de fácil escalabilidade e de baixo custo de adaptação em sistemas de workflow já existentes visto que não interfere diretamente no funcionamento e na estrutura e lógica interna dos mesmos.

Outro fator preponderante é que a interface WS IF4T faz uso dos recursos e flexibilidade oferecidos pelos web services para interoperação de aplicações via internet. Comprovando que os web services oferecem todos os recursos necessários para a implementação de interfaces de comunicação para sistemas de workflow via Internet, o que serve de motivação para novos sistemas de workflow utilizar esta tecnologia. A **Tabela 7.1** trás um comparativo das soluções de workflow transacional pesquisadas com a WS IF4T.

	Wide Project	**Modelo DTWS**	**EXOTICA Project**	**General Model**	**CrossFlow**	**WS IF4T**
Interface de Interoperabilidade padronizada						X
Interface de Interoperação compatível com WFMC						X
Independência de Fornecedor				X	X	X
Interoperação via Web	X	X	X		X	X
Utilização de Web Services						X
Escalabilidade	X	X	X		X	X
Baixo custo de adaptação					X	X

Tabela 7.1 – Comparativo entre as soluções de Workflow Transacional

7.2. Trabalhos Futuros

Para a utilização e adaptação da WS IF4T em sistemas comerciais de workflow faz-se necessário desenvolver uma serie de recursos e funcionalidades que são propostos como trabalhos futuros. Em relação as atuais funcionalidades/características da interface WS IF4T, poderíamos ter como melhoramentos:

i. Implementação de um conversor para adequar os principais padrões de mercado (Wf-XML, HTTP-Binding,etc) para a interface Abstrata;
ii. Extensões nas linguagens de definição de processos (XPDL, WPDL,etc) para que elas suportem atividades de compensação e as definições de tempos (Tempo Normal e Tempo Máximo);
iii. Elaboração de API´s que sejam capazes de extrair informações sobre os processos a partir de suas linguagens de definições (XPDL,WPDL, etc);
iv. Desenvolver uma tecnologia capaz de prover um ambiente seguro e estável para autenticação de web services;
v. Adaptação da WS IF4T para suporte do SAGAS estendido através da introdução de *safe-points*.
vi. Implantar a interface para interoperar em cenários reais, realizando testes para obter dados mais consistentes de sua performance e funcionalidade.

REFERÊNCIAS BIBLIOGRÁFICAS

[ALL96] ALLEN, ROB. Workflow Management Coalition: *Workflow, An Introduction*. Winchester Hampshire, 1996, p. 26-31.

[ALON96] ALONSO, G. & AGRAWAL, D. & KAMATH, M. *Advanced Transaction Models in Workflow Contexts*. IEEE, 1996, p. 574-581.

[AND99] M.J.ANDERSON. Workflow Management Coalition: *Workflow Standard – Interoperability Abstract Specification*, version 2.0b. Winchester Hampshire, 1999, p. 2-34.

[CAS99] F. CASATI, P. GREFEN, B. PERNICI, G. POZZI, G. SÁNCHEZ. WIDE: *Workflow model and architecture*. Politécnico di Milano, 1999, p.1-35.

[DAVE90] DAVEPORT, T.H. & SHORT, J.E. The New Industrial Engineering : *Information Technology and Business Process Redesign*. Sloan Management Review, 1990, p. 11-27.

[DONG99] DONG-SOO, HAN & JAE-YONG, SHIM. *Design and Implementation of a Distributed Transactional Workflow System*. IEEE, 1999, p. 431-434.

[DUB03] DUBRAY, JEAN-JACQUES. *BPEL4WS*. Education Business Process Modelling (http://www.ebpml.org/bpel4ws.htm) Acessado dia 16/06/2003.

[DUI01] DUIVESTEIN, SANDER. *Web Services and Workflow*. Web Services Architect (www.webservicesarchitect.com), Setembro 2001. Acessado dia 10/05/2003.

[ELM90] ELMAGARMID A. & LEU Y & W. LITWIN and M. RUSINKIEWICZ. *A Multidatabase Transaction Model for Interbase*. Proceedings of the 16th International Conference on Very Large Database, Brisbane, Australia, 1990, p.507-518.

[FAI98] FAIÇAL, FARHAT. *Programação Orientada a Objetos usando Delphi.* Editora Érica, 2ª edição, São Paulo, 1998, p.427-442.

[GMS87] GARCIA-MOLINA, HECTOR. & SALEM, KENNETH. SAGAS in 1987 SIGMOD International Conference on Management of Data. ACM Press, San Francisco 1987, p.249-259.

[HOFF00] HOFFNER, YIGAL & LUDWIG, HEIKO & GÜLCÜ, CEKI & GREFEN, PAUL. *An architeture for Cross-Organizational Business Processes*. IEEE, 2000, p. 1-5.

[HOLL95] DAVID, HOLLINGSWORTH. Workflow Management Coalition: *The Workflow Reference Model*. Winchester Hampshire, 1995, p. 6-56.

[HOLL99] DAVID, HOLLINGSWORTH. Workflow Management Coalition: *Terminology & Glossary*, version 3.0. Winchester Hampshire, 1999, p. 5-60.

[JAC01] JACQUES, SAINT-BLANCAT. *Crossflow: Final Report*. IBM France, La Gaude, 2001, p.1-12.

[KLI00] KLINGEMANN, JUSTUS. *Crossflow: Flexible Change Control*. GMD, 2000, p. 1-2.

[KUO96] KUO, DEAN & LAWLEY, MICHAEL & LIU, CHENGFEI & ORLOWSKA, MARIA. *A General Model for Nested Transactional Workflows*. Div. Of Information Technology CSIRO GPO Canberra, 1996, p. 1-16.

[LEY01] LEYMANN, FRANK. *Web Services Flow Language (WSFL1.0)*. IBM Software Group, 2001, p. 7-14.

[LIU01] LIU, JIANXUN & ZHANG, SHENSHENG & CAO, JIAN. *An Inter-Enterprise Workflow Management System for B2B E-Commerce and Supply Chain: A Case Study*. IEEE, 2001, p. 2921-2925.

[LOCK95] P. C. LOCKEMANN, & H.D. Walter. *Object-Oriented Protocol Hierarchies for Distributed Workflow Systems*. Universität Karlsruhe, 1995, p. 4-10.

[MAL03] MALLET, ALEX. *Aulas de Banco de Dados (parte2).* Universidade do Vale do Itajaí (*www.sj.univali.br/prof/Alex%20Sandro%20Teixeira%20Mallet/*), Agosto 2002. Acessado dia 10/08/2003.

[MOS85] MOSS, J. E. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, Cambridge, 1985, p.10-25.

[MTR02] MIKALSEN, TOMAS & TAI, STEFAN & ROUVELLOU, ISABELE. *Transactional Attitudes: Reliable Composition of Autonomous Web Services*. Workshop on Dependable Middleware-based Systems (WDMS 2002), part of the International Conference on Dependable Systems and Networks (DSN 2002), Washington D.C., Junho 2002, p.1-9.

[NEW02] NEWCOMER, ERIC. *Understanding Web Services.* Addison-Wesley, Boston, 2002, p.1-31.

[PRE02] PREUNER, GÜNTER & SCHREFL, MICHAEL. *Integration of Web Services into Workflows through a Multi-Level Schema Architecture*. 4th IEEE international Workshop on advanced issues of E-Commerce and Web-based Information Sytems (WECWIS), 2002, p.1-7.

[RUS92] RUSINKIEWICZ, MAREK & MANSOOR, ANSARI & LINDA, NESS. *Using Flexible Transactions to Support Multi-System Telecommunication Applications*. Proceedings of the 18th Conference on Very Large Database, Vancouver, Canada, 1992, p.65-76.

[RUS95] RUSINKIEWICZ, MAREK, & SHETH, AMIT. *Specification and Execution of Transactional Workflows*. In Modern database systems: The object model, interoperability, and beyond. ACM Press, 1995, p.592-617.

[STAR97] STARK, HEATER. *Understanding Workflow* in Workflow Handbook, Published in Association with the Workflow Management Coalition, p. 5-25, John Wiley & Sons Ltd, NewYork, USA, 1997.

[VAR00] Vários autores. *Workflow Standard: XML-HTTP Binding*. Workflow Management Coalition, Fevereiro 2000, p. 4-20.

[VAR03] Vários autores. *Business Process Execution Language for Web Services*, version 1.1. Technical Document, 31 de março 2003 (http://www-106.ibm.com/developerworks/library/ws-bpel/). Acessado dia 15/04/03.

[VON99] VONK, JOCHEM & DERKS, WIJNAND & GREFEN, PAUL & KOETSIER, MARJANCA. *Crossflow: LoC Model*. Crossflow Consortium, Setembro 1999, p. 2-14.

[WEIK92] WEIKUM, G. & SCHEK, H. *Transaction Models for Advanced Database Applications*. Ed. Morgan-kaufmann, Los Altos CA, 1992, p.2-11.

[WIET01] WIETRZY, VLAD & MAKOTO, TAKIZAWA. *A Secure Tramsaction Enviorement for Workflows in Distributed Systems*. IEEE, 2001, p. 198-205.

[WIL00] WIL VAN DER AASLT. *Loosely coupled interorganizational workflows: modeling and analyzing workflow crossing organizational boundaries*. Information & Management 37, 2000, p. 67-75.

[WOR02] WORKGROUP 1. Workflow Management Coalition Standard: *Workflow Process Definition Interface – XML Process Definition Language*. Winchester Hampshire, 1999, p. 4-8.

[WOR99] WORKGROUP 1. Workflow Management Coalition Specification, Interface 1: *Process Definition Interchange Process Model*. Winchester Hampshire, 1999, p. 4-6.

[ZNB94] ZHANG, A. & NODINE, M. & BHARGAVA, B. *Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems.* In 1994 SIGMOD International Conference on Management of Data, ACM Press, San Francisco 1994, p.67-78.

ANEXO A

Fontes do Simulador

Neste anexo é apresentado todos os códigos do simulador da WS IF4T que foi implementado em Borland *Delphi*. No *Delphi* cada parte do código é associada a uma *unit* em específico. Neste anexo os códigos fontes serão apresentados de acordo com suas *units*.

unit uDefinitions;

```
interface

const MAX_SUBPROCESS = 10;
///// Definição dos Estados de um Processo
type Wf_Process_State =(
                open_notrunning, //Processo Instânciado mas aguardando inicialização
                open_running, //Processo em execução
                open_notrunning_suspended, //Processo suspenso temporariamente
                open_prepared, //Processo Concluído e aguardando um commit ou roll-back Final
                closed_completed, //Processo Concluído com sucesso e commit Final confirmado
                closed_terminated, //Processo Concluído sem sucesso, porém sem erro, ex:teve q ser compensado
                closed_aborted //Pocesso Concluído sem sucesso, ocorreu um erro
                );

////////////////////////////////////////////////////////////////////////////////////////
//// Definição de um Processo de Workflow no Módulo Transacional
////////////////////////////////////////////////////////////////////////////////////////
type
  TProcessoWf = record
    url_WfEngine       : String; //URL do Wf Engine que o processo pertence
    id_Definicao       : Integer; //Nº da definição de processo instanciada
    id_Instancia       : Integer;//Nº da Instância
    str_Estado         : String;//Estado atual da instância
    TempoConclusao     : Integer; //Qual é o Tempo estimado para a conclusão do processo
    TempoMaxConclusao  : Integer; //Qual é o Tempo máximo tolerável para a conclusão do processo
    TempoAtivacao      : TDateTime; //Qual foi o horário de ativação do processo
end;


////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Definição da estrutura que armazena informações de parentesco entre os processos
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
type
  TParentescoInfo = record
    url_WfEngine          : String; //URL do Workflow Engine do Processo Pai
    id_DefinicaoProcesso  : Integer; //Nº da definição do Processo Pai
    id_InstanciaProcesso  : Integer; //Nº da instância do Processo Pai
    id_AtividadeInvocadora : Integer; //Nº da Atividade da Instância que invocou o subprocesso
end;

////////////////////////////////////////////////////////////////////////////////////////
///// Definição do tipo de variável para montar um vetor de processos
///// (VetorProcessos: array of TProcessos;)
////////////////////////////////////////////////////////////////////////////////////////
type
  TProcessos = record
    Processo           :TProcessoWf; //Informações sobre o processo
    Local              :Boolean; //Verdadeiro se o processo não tiver subprocessos
    SubProcessos       :array[0..MAX_SUBPROCESS] of TProcessoWf;//vetor dos processos filhos
    TemPai             :Boolean; //Se o processo tiver pai (TemPai = TRUE)
    InfoPai            :TParentescoInfo; //Informações sobre o pai do processo (se tiver)
end;

Type   TSubProcessos = array[0..MAX_SUBPROCESS] of String;

implementation

end.
```

unit WFMS;

```
interface

uses
  SysUtils, Classes, Forms, StdCtrls, Controls, ComCtrls, ExtCtrls, Buttons,
  Dialogs, DB, WS_IF4T_Impl;

type
  TfrmWFMS = class(TForm)
    GroupBox1: TGroupBox;
    memoIN: TMemo;
    Label1: TLabel;
    memoOUT: TMemo;
    Label2: TLabel;
    GroupBox2: TGroupBox;
    rgSelecao: TRadioGroup;
    memoPARAM: TMemo;
    StatusBar1: TStatusBar;
    Label3: TLabel;
    btnExecutar: TBitBtn;
    memoOP: TMemo;
    Label4: TLabel;
    cmbEstado: TComboBox;
    Label5: TLabel;
    Label6: TLabel;
    edtProcesso: TEdit;
    btnAtualizar: TBitBtn;
    procedure btnExecutarClick(Sender: TObject);
    procedure NotifyWFMS(str: String );
    procedure CreateProcessInstance(Sender: TObject);
    procedure ChangeProcessInstanceState(Sender: TObject);
    procedure ProcessInstanceStateChanged(Sender: TObject);
    procedure btnAtualizarClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    function ValidateWfEngine(strWEngine: string):Integer;
    function ValidateDefProcess(strProcess: string):Integer;
    function ValidateInstProcess(iProcess: Integer):Boolean;
    function GetInstProcess(strProcess: string):Integer;
    procedure SetProcess( strProcess :String );
    procedure SetState( strState:String );
    function ConvertToUiState( strState: string ) :string;
    function ConvertToState( strState: string ) :string;
  public
    { Public declarations }
  private
    //Variáveis que aramazenam informações sobre o processo do WFMS
    m_strProcesso, m_strEstado, m_strWEngPai :String;

  published
    property ProcessoWFMS : String read m_strProcesso write SetProcess;
    property EstadoProcessoWFMS: String read m_strEstado write SetState;
    property PaiProcessoWFMS : String read m_strWEngPai write m_strWEngPai;

  end;

var
  frmWFMS: TfrmWFMS;
  WS_IF4T: TWS_IF4T;
```

```
const NOT_FOUND = -1;

implementation

uses SockApp, Variants, uDataModule, uMessages ;
{$R *.dfm}
procedure TfrmWFMS.FormCreate(Sender: TObject);
begin
  //Inicializando o processo padrão no WFMS primário
  if frmWFMS.Caption = 'WFMS - Loja Virtual NetBUY' then
  begin
    //Processo primário que chamará todos os outros subprocessos
    frmWFMS.ProcessoWFMS:='processar_pedido';
    //O estado do processo já se encontra ativo
    frmWFMS.EstadoProcessoWFMS:='open.running';
    //O WFMS que é pai do processo primário
    frmWFMS.PaiProcessoWFMS:='loja virtual';
  end;
end;


procedure TfrmWFMS.SetProcess( strProcess :String );
begin
 if Trim( strProcess ) <> '' then
 begin
   edtProcesso.Clear;
   edtProcesso.Text := strProcess;
   m_strProcesso := strProcess
 end;
end;

procedure TfrmWFMS.SetState( strState :String );
begin
  if Trim( strState ) <> '' then
  begin
    if strState = 'open.running' then cmbEstado.Text:= ConvertToUiState('open.running')
    else if strState = 'open.notrunning' then cmbEstado.Text:=ConvertToUiState('open.notrunning')
    else if strState = 'open.notrunning.suspended' then
cmbEstado.Text:=ConvertToUiState('open.notrunning.suspended')
    else if strState = 'closed.completed' then cmbEstado.Text:= ConvertToUiState('closed.completed')
    else if strState = 'closed.terminated' then cmbEstado.Text:= ConvertToUiState('closed.terminated')
    else if strState = 'closed.aborted'  then cmbEstado.Text:= ConvertToUiState('closed.aborted');

    if strState <> m_strEstado then m_strEstado := strState
  end;
end;

function TfrmWFMS.ValidateWfEngine(strWEngine: string):Integer;
var
  ResultLookup : Variant;
begin
  ResultLookup:= DM.tblNomeWEngine.Lookup('WorkFlow_Engine', strWEngine,'id_WorkFlow_Engine');
  if VarType( ResultLookup ) = varNull then
    Result:= NOT_FOUND
  else
    Result:= ResultLookup;
end;

function TfrmWFMS.ValidateDefProcess(strProcess: string):Integer;
var
  ResultLookup : Variant;
begin
  ResultLookup:= DM.tblNomeProcesso.Lookup('Nome_Processo', strProcess,'id_Def_Processo');
```

```
  if VarType( ResultLookup ) = varNull then
    Result:= NOT_FOUND
  else
    Result:= ResultLookup;
end;
function TfrmWFMS.ConvertToUiState( strState: string ) :string;
begin
     if strState = 'open.running' then Result:= 'ativo'
     else if strState = 'open.notrunning' then Result:= 'inativo'
     else if strState = 'open.notrunning.suspended' then Result:='suspenso'
     else if strState = 'open.prepared' then Result:= 'preparado'
     else if strState = 'closed.completed' then Result:= 'completo'
     else if strState = 'closed.terminated' then Result:= 'compensado'
     else if strState = 'closed.aborted'  then Result:= 'abortado'
     else Result:= '';
end;
function TfrmWFMS.ConvertToState( strState: string ) :string;
begin
     if strState = 'ativo' then Result:= 'open.running'
     else if strState = 'inativo' then Result:= 'open.notrunning'
     else if strState = 'suspenso' then Result:='open.notrunning.suspended'
     else if strState = 'preparado' then Result:= 'open.prepared'
     else if strState = 'completo' then Result:= 'closed.completed'
     else if strState = 'compensado' then Result:= 'closed.terminated'
     else if strState = 'abortado'  then Result:= 'closed.aborted'
     else Result:= '';
end;

function TfrmWFMS.ValidateInstProcess( iProcess: Integer ):Boolean;
begin
  Result:= DM.tblNomeProcesso.Locate('inst_Processo', iProcess,[loCaseInsensitive]);
end;

function TfrmWFMS.GetInstProcess(strProcess: string):Integer;
var
  ResultLookup : Variant;
begin
  ResultLookup:= DM.tblNomeProcesso.Lookup('Nome_Processo', strProcess,'inst_Processo');
  if VarType( ResultLookup ) = varNull then
    Result:= NOT_FOUND
  else
    Result:= ResultLookup;
end;

procedure TfrmWFMS.NotifyWFMS(str: String );
begin
   ShowMessage(Format(NOTIFY_WS_IF4T, [str]));
end;

procedure TfrmWFMS.btnExecutarClick(Sender: TObject);
begin
  case rgSelecao.ItemIndex  of
      0: CreateProcessInstance(Sender);
      1: ChangeProcessInstanceState(Sender);
  end;
  memoPARAM.Clear;
  memoPARAM.SetFocus;
  rgSelecao.ItemIndex:= -1;
end;

procedure TfrmWFMS.btnAtualizarClick(Sender: TObject);
begin
  if ConvertToUiState( m_strEstado )<> cmbEstado.Text then
```

```
  begin
    m_strEstado := ConvertToState( cmbEstado.Text );
    ProcessInstanceStateChanged(Sender);
  end;
end;

procedure TfrmWFMS.CreateProcessInstance(Sender: TObject);
var
  iWEng, iProcessDef, iInstProcessPai, iAtivInvoc :Integer;
  strWEng, strProcessDef :String;
  bError : Boolean;
begin
  bError:= FALSE;
  //Atribuindo os Valores Default
  iInstProcessPai := 1;
  iAtivInvoc := 1;

  //Procurando o id Equivalente ao texto digitado no WfEng
  strWEng := UpperCase( memoPARAM.Lines[0]);
  iWEng := ValidateWfEngine( strWEng );
  if iWEng = NOT_FOUND then
  begin
    bError:= True;
    ShowMessage( NOT_FOUND_WENG );
  end;

  //Procurando o id Equivalente ao texto digitado no ProcessDef
  strProcessDef := UpperCase( memoPARAM.Lines[1]);
  iProcessDef := ValidateDefProcess( strProcessDef );
  if iProcessDef = NOT_FOUND then
  begin
    bError:= TRUE;
    ShowMessage( NOT_FOUND_PROC );
  end;

  //Atribuindo a instância do Processo Pai
  if memoPARAM.Lines[2] <> '' then
    iInstProcessPai := StrToInt( memoPARAM.Lines[2]);

  //Atribuindo que a atividade invocadora
  if memoPARAM.Lines[3] <> '' then
    iAtivInvoc := StrToInt( memoPARAM.Lines[3]);;

  //Senão ocorreu nenhum erro chama a operação da IF4 abstrata da WS IF4T
  if not bError then
    WS_IF4T.CreateProcessInstanceABS(iWEng, iProcessDef, iInstProcessPai, iAtivInvoc);
end;

procedure TfrmWFMS.ChangeProcessInstanceState(Sender: TObject);
var
  iWEng, iInstProcess:Integer;
  strWEng,strState :String;
  bError : Boolean;
begin
  bError :=FALSE;
  //Procurando o id Equivalente ao texto digitado no WfEng
  strWEng := UpperCase( memoPARAM.Lines[0]);
  iWEng := ValidateWfEngine( strWEng );
  if iWEng = NOT_FOUND then
  begin
    bError:= TRUE;
    ShowMessage( NOT_FOUND_WENG );
  end;
```

```
  //Validando a instância Equivalente ao texto digitado na InstProcesso
  iInstProcess:=StrToInt(memoPARAM.Lines[1]);
  if not ValidateInstProcess(iInstProcess) then
  begin
    bError:= TRUE;
    ShowMessage( NOT_FOUND_INST );
  end;

  //Adaptando o estado digitado para ser enviado
  strState := Trim( LowerCase(memoPARAM.Lines[2]) );
  if strState = 'ative' then strState:= 'open.running'
  else if strState = 'suspenda' then strState:= 'open.notrunning.suspended'
  else if strState = 'complete' then strState:= 'closed.completed'
  else if strState = 'compense' then strState:= 'closed.terminated'
  else if strState = 'aborte'   then strState:= 'closed.aborted'
  else
  begin
       bError:= TRUE;
       ShowMessage( NOT_FOUND_STATE );
  end;

  //Senão ocorreu nenhum erro chama a operação da IF4 abstrata da WS IF4T
  if not bError then
     WS_IF4T.ChangeProcessInstanceStateABS( iWEng, iInstProcess, strState );
end;

procedure TfrmWFMS.ProcessInstanceStateChanged(Sender: TObject);
var
  iWEng, iInstProcess:Integer;
  bError : Boolean;
begin
  bError :=FALSE;
  //Procurando o id Equivalente ao WEngine Pai
  iWEng := ValidateWfEngine( m_strWEngPai );
  if iWEng = NOT_FOUND then
     bError:= TRUE;

  //Validando a instância Equivalente ao Processo do WFMS
  iInstProcess:= GetInstProcess( m_strProcesso );
  if not iInstProcess = NOT_FOUND then
     bError:= TRUE;

   //Senão ocorreu nenhum erro chama a operação da IF4 abstrata da WS IF4T
  if not bError then
     WS_IF4T.ProcessInstanceStateChangedABS( iWEng, iInstProcess, m_strEstado )
end;

initialization
  TWebAppSockObjectFactory.Create('wsif4t');
end.
```

{ Invokable interface IWebService_IF4 }
unit WS_IF4T_Intf;

```
interface

uses InvokeRegistry, Types, XSBuiltIns, SoapHTTPClient;

  type
    TAtributo = class(TRemotable)
  private
    Att1, Att2 : Integer;
  published
    property Attributo1 : Integer read Att1 write Att1;
    property Attributo2 : Integer read Att2 write Att2;
  end;

type

  { Invokable interfaces must derive from IInvokable }
  IWS_IF4T = interface(IInvokable)
  ['{B4F8B2CA-4E32-4D48-B791-17A74A5B7835}']
  function CreateProcessInstance(url_Engine_Origem,url_Engine_Destino: String; id_Processo_Pai,
id_Atividade_Invocadora,id_Def_Processo_Filho : Integer): Integer; stdcall;
  function SetProcessInstanceAttributes(url_Engine_Origem,url_Engine_Destino: String; id_Processo_Pai,
id_Atividade_Invocadora,id_Instancia_Processo : Integer; typAtributos :TAtributo ): Boolean; stdcall;
  function ChangeProcessInstanceState(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean; stdcall;
  //function GetProcessInstanceAttributes(url_Engine_Origem,url_Engine_Destino: String; id_Processo_Pai,
id_Atividade_Invocadora,id_Instancia_Processo : Integer; typAtributos :TAtributo ): TAtributo; stdcall;
  function GetProcessInstanceState(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer): String; stdcall;
  function ProcessInstanceStateChanged(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean; stdcall;
    { Methods of Invokable interface must not use the default }
    { calling convention; stdcall is recommended }
end;

function GetIWS_IF4T(UseWSDL: Boolean=System.False; Addr: string=''; HTTPRIO: THTTPRIO = nil):
IWS_IF4T;

implementation

function GetIWS_IF4T(UseWSDL: Boolean; Addr: string; HTTPRIO: THTTPRIO): IWS_IF4T;
const
  defWSDL = 'http://localhost:1024/wfms_a.wsif4t/wsdl/IWS_IF4T';
  defURL  = 'http://localhost:1024/wfms_a.wsif4t/soap/IWS_IF4T';
  defSvc  = 'IWS_IF4Tservice';
  defPrt  = 'IWS_IF4TPort';
var
  RIO: THTTPRIO;
begin
  Result := nil;
  if (Addr = '') then
  begin
    if UseWSDL then
      Addr := defWSDL
    else
      Addr := defURL;
  end;
  if HTTPRIO = nil then
    RIO := THTTPRIO.Create(nil)
  else
```

```
    RIO := HTTPRIO;
  try
    Result := (RIO as IWS_IF4T);
    if UseWSDL then
    begin
      RIO.WSDLLocation := Addr;
      RIO.Service := defSvc;
      RIO.Port := defPrt;
    end else
      RIO.URL := Addr;
  finally
    if (Result = nil) and (HTTPRIO = nil) then
      RIO.Free;
  end;
end;


initialization
  { Invokable interfaces must be registered }
  InvRegistry.RegisterInterface(TypeInfo(IWS_IF4T));
  InvRegistry.RegisterDefaultSOAPAction(TypeInfo(IWS_IF4T),                    'urn:WS_IF4T_Intf-
IWS_IF4T#%operationName%');
  RemTypeRegistry.RegisterXSClass(TAtributo);
end.
```

{Invokable implementation File for TWebService_IF4 which implements IWebService_IF4}

unit WS_IF4T_Impl;

```
interface

uses InvokeRegistry, Types, XSBuiltIns, WS_IF4T_Intf,uDefinitions;

  ////////////////////////////////////////////////////////
  //// Classe que implementa a WS IF4T
  ////////////////////////////////////////////////////////
  type
  { TWebService_IF4 }
  TWS_IF4T = class(TInvokableClass, IWS_IF4T)
  public
     //Declaração das Funções da Interface 4 Abstrata
     function CreateProcessInstanceABS(iWEng,iProcessDef,iInstProcessPai,iAtivInvoc:Integer):Integer;
     function ChangeProcessInstanceStateABS(iWEng,iInstProcesso:Integer;strEstado:String):Boolean;
     function ProcessInstanceStateChangedABS(iWEng,iInstProcesso:Integer;strEstado:String):Boolean;
     //Declaração das Funções da Interface 4 Web Services
     function CreateProcessInstance(url_Engine_Origem,url_Engine_Destino: String; id_Processo_Pai,
id_Atividade_Invocadora,id_Def_Processo_Filho : Integer): Integer; stdcall;
     function ChangeProcessInstanceState(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean; stdcall;
     function ProcessInstanceStateChanged(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean; stdcall;
     function GetProcessInstanceState(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer): String; stdcall;
     function SetProcessInstanceAttributes(url_Engine_Origem,url_Engine_Destino: String; id_Processo_Pai,
id_Atividade_Invocadora,id_Instancia_Processo : Integer; typAtributos :TAtributo ): Boolean; stdcall;
     function GetProcessInstanceAttributes(url_Engine_Origem,url_Engine_Destino: String; id_Processo_Pai,
id_Atividade_Invocadora,id_Instancia_Processo : Integer; typAtributos :TAtributo ): TAtributo; stdcall;
     //Declaração dos procedimentos do algoritmo executor
     procedure AtiveAlgoritmoExecutor( iIndexProcess :Integer );
     procedure AlgoritmoExecutorMetadeTempo(Sender: TObject);
     procedure AlgoritmoExecutorTempoNormal(Sender: TObject);
     procedure AlgoritmoExecutorTempoMaximo(Sender: TObject);
  private
     //Declaração das funções de uso interno
     function IsOUTBOUND( url_Engine_Origem :String ):Boolean;
     function AgoraInicial():String;
     function GetURLEngine( id_Engine :Integer ):String;
     function GetIdEngine( url_Engine :String ):Integer;
     function GetEngineName( id_Engine :Integer ):String;
     function GetProcessInstance( id_Def_Processo :Integer ):Integer;
     function GetProcessName( id_Def_Processo :Integer ):String;
     function GetIdDefProcesso( str_Processo : String ) :Integer;
     function GetTime(  id_Def_Processo :Integer; strResultField :String ):Integer;
     function FindSubprocess( id_DefSubprocesso : Integer ) : Integer;
     function NumeroSubProcessos( iIndexProcess :Integer  ) : Integer;
     //Declaração das Funções da API
     function GET_PROCESS_DEFINITION( id_Inst_Processo :Integer ):Integer;
     function GET_SUB_PROCESS( id_Processo_Pai: Integer ) :TSubprocessos;
     //Declaração das Funções INBOUND e OUTBOUND
     function CreateProcessInstanceINBOUND(url_Engine_Origem,url_Engine_Destino: String;
id_Processo_Pai, id_Atividade_Invocadora,id_Def_Processo_Filho : Integer): Integer;
     function CreateProcessInstanceOUTBOUND(url_Engine_Origem,url_Engine_Destino: String;
id_Processo_Pai, id_Atividade_Invocadora,id_Def_Processo_Filho : Integer): Integer;
```

```
    function ChangeProcessInstanceStateINBOUND(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean;
    function ChangeProcessInstanceStateOUTBOUND(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean;
    function ProcessInstanceStateChangedINBOUND(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean;
    function ProcessInstanceStateChangedOUTBOUND(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer; st_Estado: String ): Boolean;
    //Declaração dos procedimentos utilizados no algoritmo executor
    procedure FINALIZE_ALGORITMO();
    procedure ABORTE_PROCESSO( iIndexProcess :Integer );
    procedure ABORTE_SUBPROCESSOS_ANTERIORES( iIndexProcess,iSubprocessAborted:Integer);
    procedure CONFIRME_PROCESSO( iIndexProcess :Integer );
  end;

  ////////////////////////////////////////////////////
  //// Definição das Constantes Internas
  ////////////////////////////////////////////////////
  const SELF_URL   = 'http://localhost:1024/wfms_a.wsif4t/wsdl/IWS_IF4T';
  const OPERATION_ERROR = 0;
  const WF_INVALID_ENGINE = -1;
  const WF_INVALID_PROCESS_DEFINITION = -2;
  const WF_ENGINE_ERROR = -3;
  const WF_INVALID_PROCESS_INSTANCE = -4;


//Variáveis Globais
var
  //Vetor de Processos
  Vetor: array[0..10] of TProcessos;
  iProcesso :Integer;

implementation


uses Variants,uDataModule,SysUtils, WFMS, uMessages,uWebModule;


////////////////////////////////////////////////////////
//// Implementação das Funções Internas
////////////////////////////////////////////////////////

function TWS_IF4T.IsOUTBOUND( url_Engine_Origem :String ):Boolean;
begin
  Result:=FALSE;
  //Mensagens OutBound
  if url_Engine_Origem = SELF_URL then
  begin
    Result:= TRUE;
  end
  //Mensagens InBound
  else if url_Engine_Origem <> SELF_URL then
  begin
    Result:= FALSE;
  end;
end;

function TWS_IF4T.AgoraInicial():String;
var
  Tempo :TDateTime;
begin
  if Vetor[0].Processo.TempoAtivacao = 0 then
    Tempo := Vetor[0].Processo.TempoAtivacao
```

```
  else
    Tempo := Now() - Vetor[0].Processo.TempoAtivacao;

  Result:= TimeToStr(Tempo);
end;
function TWS_IF4T.GetURLEngine( id_Engine :Integer ):String;
var
  ResultLookup : Variant;
begin
  //Procurando o URL equivalente ao id do Workflow Engine
  ResultLookup:= DM.tblConversao.Lookup( 'id_WorkFlow_Engine',
                                id_Engine,
                                'url_WorkFlow_Engine'
                               );
  if VarType( ResultLookup ) = varNull then
    //Senão encontrou retorne um código de erro
    Result:= IntToStr( WF_INVALID_ENGINE )
  else
    //Se encontrou retorne a instância
    Result:= ResultLookup;
end;
function TWS_IF4T.GetIdEngine( url_Engine :String ):Integer;
var
  ResultLookup : Variant;
begin
  //Procurando o URL equivalente ao id do Workflow Engine
  ResultLookup:= DM.tblConversao.Lookup( 'url_WorkFlow_Engine',
                                url_Engine,
                                'id_WorkFlow_Engine'
                               );
  if VarType( ResultLookup ) = varNull then
    //Senão encontrou retorne um código de erro
    Result:= WF_INVALID_ENGINE
  else
    //Se encontrou retorne a instância
    Result:= ResultLookup;
end;



function TWS_IF4T.GetEngineName( id_Engine :Integer ):String;
var
  ResultLookup : Variant;
begin
  //Procurando o URL equivalente ao id do Workflow Engine
  ResultLookup:= DM.tblNomeWEngine.Lookup( 'id_WorkFlow_Engine',
                                  id_Engine,
                                  'WorkFlow_Engine'
                                 );
  if VarType( ResultLookup ) = varNull then
    //Senão encontrou retorne um código de erro
    Result:= IntToStr( WF_INVALID_ENGINE )
  else
    //Se encontrou retorne a instância
    Result:= ResultLookup;
end;

function TWS_IF4T.GetProcessInstance( id_Def_Processo :Integer ):Integer;
var
  ResultLookup : Variant;
begin
  //Procurando o URL equivalente ao id do Workflow Engine
  ResultLookup:= DM.tblNomeProcesso.Lookup( 'id_Def_Processo',
```

```
                                    id_Def_Processo,
                                    'inst_Processo'
                                   );
    if VarType( ResultLookup ) = varNull then
      //Senão encontrou retorne um código de erro
      Result:= WF_INVALID_PROCESS_DEFINITION
   else
      //Se encontrou retorne a instância
      Result:= ResultLookup;
end;

function TWS_IF4T.GetProcessName( id_Def_Processo :Integer ):String;
var
   ResultLookup : Variant;
begin
   //Procurando o URL equivalente ao id do Workflow Engine
   ResultLookup:= DM.tblNomeProcesso.Lookup( 'id_Def_Processo',
                                    id_Def_Processo,
                                    'Nome_Processo'
                                   );
    if VarType( ResultLookup ) = varNull then
      //Senão encontrou retorne um código de erro
      Result:= IntToStr( WF_INVALID_PROCESS_DEFINITION )
   else
      //Se encontrou retorne a instância
      Result:= ResultLookup;
end;


function TWS_IF4T.GetTime(  id_Def_Processo :Integer; strResultField :String ):Integer;
var
   ResultLookup : Variant;
begin
    //Procurando a Definição de processo equivalente ao id da inst Processo
   ResultLookup:= DM.tblProcessos.Lookup( 'id_Def_Processo',
                                    id_Def_Processo,
                                    strResultField
                                     );
    if VarType( ResultLookup ) = varNull then
      //Senão encontrou
      Result:= 0
    else
      Result:= ResultLookup;
end;

function TWS_IF4T.GET_PROCESS_DEFINITION( id_Inst_Processo :Integer ):Integer;
var
   ResultLookup : Variant;
begin
   //Procurando a Definição de processo equivalente ao id da inst Processo
   ResultLookup:= DM.tblNomeProcesso.Lookup( 'inst_Processo',
                                    id_Inst_Processo,
                                    'id_Def_Processo'
                                   );
    if VarType( ResultLookup ) = varNull then
      //Senão encontrou retorne um código de erro
      Result:= WF_INVALID_PROCESS_INSTANCE
   else
      //Se encontrou retorne a instância
      Result:= ResultLookup;
end;

function TWS_IF4T.GetIdDefProcesso( str_Processo : String ) :Integer;
```

```
var
  ResultLookup : Variant;
begin
  //Procurando a Definição de processo equivalente ao id da inst Processo
  ResultLookup:= DM.tblNomeProcesso.Lookup( 'Nome_Processo',
                                    str_Processo,
                                    'id_Def_Processo'
                                   );
   if VarType( ResultLookup ) = varNull then
     //Senão encontrou retorne um código de erro
     Result:= 0
  else
     //Se encontrou retorne a instância
     Result:= ResultLookup;
end;


function TWS_IF4T.GET_SUB_PROCESS( id_Processo_Pai: Integer ) :TSubprocessos;
var
  ResultLookup : Variant;
  SubProcessos : TSubprocessos;
  iPos, iCountSubproc: Integer;
  strResultado : String;
begin
  for iPos:= 0 to MAX_SUBPROCESS do
       Subprocessos[iPos]:= '';

  //Procurando a Definição de processo equivalente ao id da inst Processo
  ResultLookup:= DM.tblProcessos.Lookup( 'id_Def_Processo',
                                    id_Processo_Pai,
                                    'id_Subprocessos'
                                     );
   if VarType( ResultLookup ) = varNull then
   begin
     //Senão encontrou o array Vazio
     Result:= Subprocessos;
   end
  else
  begin
     iCountSubproc := 0;
     strResultado := String(ResultLookup);
     //Contando quantos subprocessos tem a string
     for iPos :=1 to (Length( strResultado )) do
     begin
       if strResultado[iPos] = ',' then
         iCountSubproc := iCountSubproc +1
       else
         Subprocessos[iCountSubproc]:=Subprocessos[iCountSubproc] + strResultado[iPos];
     end;
     //Retorne o array com os subprocessos
     Result:= Subprocessos;
  end;

end;

function TWS_IF4T.FindSubprocess( id_DefSubprocesso : Integer ) : Integer;
var
  iPos, iResultado : Integer;
begin
  iResultado := WF_INVALID_PROCESS_DEFINITION;
  for iPos := 0 to MAX_SUBPROCESS do
  begin
     if Vetor[0].SubProcessos[iPos].id_Definicao = id_DefSubprocesso then
```

```
      iResultado:= iPos;
  end;
  Result:= iResultado;
end;

function TWS_IF4T.NumeroSubProcessos( iIndexProcess :Integer  ) : Integer;
var
  iPos, iSubprocess :Integer;
begin
  iSubprocess := 0;
  for iPos := 0 to MAX_SUBPROCESS do
  begin
    if Vetor[iIndexProcess].SubProcessos[iPos].id_Definicao <> 0 then
      iSubprocess:= iSubprocess+1;
  end;
  Result:= iSubprocess;
end;



////////////////////////////////////////////////////////////////////////////////
//// Implementação das Operações da Interface 4 Abstrata
////////////////////////////////////////////////////////////////////////////////

function TWS_IF4T.CreateProcessInstanceABS( iWEng,
                                    iProcessDef,
                                    iInstProcessPai,
                                    iAtivInvoc :Integer
                                   ): Integer;
var
  url_Destino  : String;
begin
  url_Destino := GetURLEngine( iWEng );

  if url_Destino <> IntToStr( WF_INVALID_ENGINE ) then
    //Convertendo de Interface 4 Abstrata para Interface 4 Web Services
    Result:= CreateProcessInstance( SELF_URL,
                                url_Destino,
                                iInstProcessPai,
                                iAtivInvoc,
                                iProcessDef
                               )
  else
    Result:= WF_INVALID_ENGINE ;
end;

function TWS_IF4T.ChangeProcessInstanceStateABS(
                      iWEng,
                      iInstProcesso:Integer;
                      strEstado: String
                      ): Boolean;
var
  url_Destino  : String;
begin
  url_Destino := GetURLEngine( iWEng );

  if url_Destino <> IntToStr( WF_INVALID_ENGINE ) then
    //Convertendo de Interface 4 Abstrata para Interface 4 Web Services
    Result:= ChangeProcessInstanceState( SELF_URL,
                                    url_Destino,
                                    iInstProcesso,
                                    strEstado
                                   )
```

```
  else
    Result:= FALSE;
end;

function TWS_IF4T.ProcessInstanceStateChangedABS(
                iWEng,
                iInstProcesso:Integer;
                strEstado: String
                ): Boolean;
var
  url_Destino  : String;
begin
  url_Destino := GetURLEngine( iWEng );

  //Senão tiver pai para notificar atualize somente o estado do processo no vetor
  if url_Destino = SELF_URL then
    Vetor[0].Processo.str_Estado := strEstado;

  if (url_Destino <> SELF_URL)  and
    (url_Destino <> IntToStr( WF_INVALID_ENGINE ) ) then
    //Convertendo de Interface 4 Abstrata para Interface 4 Web Services
    Result:= ProcessInstanceStateChanged( SELF_URL,
                                  url_Destino,
                                  iInstProcesso,
                                  strEstado
                                 )
  else
    Result:= FALSE;
end;
////////////////////////////////////////////////////////////////////////////////////
//// Implementação das Operações da Interface 4 Web Services
////////////////////////////////////////////////////////////////////////////////////

function TWS_IF4T.CreateProcessInstance( url_Engine_Origem,
                                url_Engine_Destino: String;
                                id_Processo_Pai,
                                id_Atividade_Invocadora,
                                id_Def_Processo_Filho : Integer
                               ): Integer; stdcall;
begin
  //Determinando o sentido da operação (OUTBOUND ou INBOUND)
  if IsOUTBOUND( url_Engine_Origem ) then
    Result:=    CreateProcessInstanceOUTBOUND(url_Engine_Origem,url_Engine_Destino,    id_Processo_Pai,
id_Atividade_Invocadora,id_Def_Processo_Filho)
  else
    Result:=     CreateProcessInstanceINBOUND(url_Engine_Origem,url_Engine_Destino,     id_Processo_Pai,
id_Atividade_Invocadora,id_Def_Processo_Filho);
end;

function TWS_IF4T.ChangeProcessInstanceState(
                    url_Engine_Origem,
                    url_Engine_Destino: String;
                    id_Instancia_Processo:Integer;
                    st_Estado: String
                    ): Boolean; stdcall;
begin
   //Determinando o sentido da operação (OUTBOUND ou INBOUND)
  if IsOUTBOUND( url_Engine_Origem ) then
    Result:=                    ChangeProcessInstanceStateOUTBOUND(url_Engine_Origem,url_Engine_Destino,
id_Instancia_Processo, st_Estado)
  else
    Result:=                      ChangeProcessInstanceStateINBOUND(url_Engine_Origem,url_Engine_Destino,
id_Instancia_Processo, st_Estado);
```

```
end;

function TWS_IF4T.ProcessInstanceStateChanged(
                url_Engine_Origem,
                url_Engine_Destino: String;
                id_Instancia_Processo:Integer;
                st_Estado: String
                ): Boolean; stdcall;
begin
   //Determinando o sentido da operação (OUTBOUND ou INBOUND)
  if IsOUTBOUND( url_Engine_Origem ) then
     Result:=                ProcessInstanceStateChangedOUTBOUND(url_Engine_Origem,url_Engine_Destino,
id_Instancia_Processo, st_Estado)
  else
     Result:=                    ProcessInstanceStateChangedINBOUND(url_Engine_Origem,url_Engine_Destino,
id_Instancia_Processo, st_Estado);
end;

function          TWS_IF4T.SetProcessInstanceAttributes(url_Engine_Origem,url_Engine_Destino:          String;
id_Processo_Pai, id_Atividade_Invocadora,id_Instancia_Processo : Integer; typAtributos :TAtributo ): Boolean;
stdcall;
begin
   //Utilizado para testes
   Result:=True;
end;
function          TWS_IF4T.GetProcessInstanceAttributes(url_Engine_Origem,url_Engine_Destino:          String;
id_Processo_Pai,  id_Atividade_Invocadora,id_Instancia_Processo  :  Integer;  typAtributos  :TAtributo  ):
TAtributo; stdcall;
{var
  valor: TAtributo;}
begin
    //GetNormalTimeProcess ( typAtributos.Attributo1 )
    //GetMaxTimeProcess ( typAtributos.Attributo1 )
  {valor.Attributo1 := 10;
  valor.Attributo2 := 10;}
  Result:= typAtributos;
end;

function                              TWS_IF4T.GetProcessInstanceState(url_Engine_Origem,url_Engine_Destino:
String;id_Instancia_Processo:Integer): String; stdcall;
begin
   Result:='open.running';
end;


//////////////////////////////////////////////////////////////////////
//// Implementação das lógicas de Gerenciamento
//// INBOUND e OUTBOUND
//////////////////////////////////////////////////////////////////////

function TWS_IF4T.CreateProcessInstanceOUTBOUND( url_Engine_Origem: String;
                url_Engine_Destino: String;
                id_Processo_Pai: Integer;
                id_Atividade_Invocadora: Integer;
                id_Def_Processo_Filho : Integer
                ): Integer;
var
  comunicador: IWS_IF4T;
  iInstCriada, iWEng, iPos: Integer;
  SubProcessos : TSubprocessos;
begin
  iWEng:= GetIdEngine( url_Engine_Destino );
  try
```

```
    //Monstrando os Eventos
    frmWFMS.memoOUT.Lines.Add(
        Format( CREATE_PROC_MSG_OUT,
              [AgoraInicial(), GetEngineName(iWEng), GetProcessName(id_Def_Processo_Filho),
              id_Processo_Pai, id_Atividade_Invocadora]
            ) );

    //Setando o componente para se comunicar com o workflow Engine de destino
    WebMod.HTTPRIO.WSDLLocation := url_Engine_Destino;
    //Recebendo o TypeCast do Componente
    comunicador:= WebMod.HTTPRIO as IWS_IF4T;
    //Chamando a operação CreateProcessInstance no WF de destino
    iInstCriada:= comunicador.CreateProcessInstance(url_Engine_Origem,
                              url_Engine_Destino,
                              id_Processo_Pai,
                              id_Atividade_Invocadora,
                              id_Def_Processo_Filho);

    //Escrevendo Resposta no Memo
    frmWFMS.memoOUT.Lines.Add(
        Format( PROCESS_CREATED,
              [AgoraInicial(),GetProcessName(id_Def_Processo_Filho ),iInstCriada]
            ) );

    //Se resposta for válida
    if iInstCriada > 0 then
    begin
      //Preenchendo o Vetor de Processos em memória
      if Vetor[0].Processo.url_WfEngine = '' then
      begin
        Vetor[0].Processo.url_WfEngine := url_Engine_Origem;
        Vetor[0].Processo.id_Definicao := GET_PROCESS_DEFINITION( id_Processo_Pai );
        Vetor[0].Processo.id_Instancia := id_Processo_Pai;
        Vetor[0].Processo.str_Estado   := frmWFMS.EstadoProcessoWFMS;
        Vetor[0].Processo.TempoAtivacao := Now();
        Vetor[0].Processo.TempoConclusao := GetTime(   GET_PROCESS_DEFINITION( id_Processo_Pai ),
'Tempo' );
        Vetor[0].Processo.TempoMaxConclusao := GetTime(  GET_PROCESS_DEFINITION( id_Processo_Pai
), 'TempoMaximo' );

        //Obtendo todos os subprocessos do processo Pai
        SubProcessos := GET_SUB_PROCESS(GET_PROCESS_DEFINITION( id_Processo_Pai ));
        //Preenchedo os subprocessos
        for iPos := 0  to MAX_SUBPROCESS do
        Vetor[0].SubProcessos[iPos].id_Definicao:= GetIdDefProcesso(SubProcessos[iPos]);
      end;
      //Localizando o índice do Subprocesso
      iPos := FindSubprocess( id_Def_Processo_Filho );
      //Preenchendo o url do subprocesso
      Vetor[0].SubProcessos[iPos].url_WfEngine := url_Engine_Destino;
      //Preenchendo o valor da instância
      Vetor[0].SubProcessos[iPos].id_Instancia := iInstCriada;
      //Preenchendo o estado padrão de um subprocesso quando ele é criado
      Vetor[0].SubProcessos[iPos].str_Estado := 'open.notrunning';
      //Escrevendo no Memo a operação de preenchimento de vetor
      frmWFMS.memoOP.Lines.Add( Format(FILL_VECTOR_MSG,[AgoraInicial()]));
    end;
    //Retornando o valor da operação
    Result:= iInstCriada;
  except
    //Pegando-se o objeto de exceção
    on E: Exception do
    begin
```

```
        frmWFMS.NotifyWFMS( Format('Tipo: %s',[ E.ClassName]) );
        //Escrevendo no Memo
        frmWFMS.memoOUT.Lines.Add( ERR_TRANS_MSG );
        //Retornando código de Erro
        Result := OPERATION_ERROR;
      end;
  end;//end try-except
end;

function TWS_IF4T.CreateProcessInstanceINBOUND(
                    url_Engine_Origem: String;
                    url_Engine_Destino: String;
                    id_Processo_Pai: Integer;
                    id_Atividade_Invocadora: Integer;
                    id_Def_Processo_Filho: Integer
                    ): Integer;
var
  iResultado,iWEng, iWEng2, iPos :Integer;
  SubProcessos : TSubprocessos;
begin
  iWEng:= GetIdEngine( url_Engine_Origem  );
  iWEng2:= GetIdEngine( url_Engine_Destino );

  //Monstrando os Eventos
  frmWFMS.memoIN.Lines.Add(
      Format( CREATE_PROC_MSG_IN,
           [AgoraInicial(),GetEngineName(iWEng), GetEngineName(iWEng2),
            GetProcessName(id_Def_Processo_Filho),
            id_Processo_Pai,id_Atividade_Invocadora]
          ) );

  //Retornando a Instância do Processo Criado
  iResultado:= GetProcessInstance( id_Def_Processo_Filho );

  if iResultado = WF_INVALID_PROCESS_DEFINITION  then
    //Monstrando os Eventos
    frmWFMS.memoIN.Lines.Add( ERR_INVALID_PROCESS_DEFINITION )
  else
  begin
    //Monstrando os Eventos
    frmWFMS.memoIN.Lines.Add(
        Format( PROCESS_CREATED,
             [AgoraInicial(),GetProcessName(id_Def_Processo_Filho ),iResultado]
            ) );
    //Preenchendo o Vetor de Processos em memória
    if Vetor[0].Processo.url_WfEngine = '' then
    begin
      //Preenchendo informações de parentesco
      Vetor[0].TemPai := TRUE;
      Vetor[0].InfoPai.url_WfEngine := url_Engine_Origem;
      Vetor[0].InfoPai.id_DefinicaoProcesso := GET_PROCESS_DEFINITION( id_Processo_Pai );
      Vetor[0].InfoPai.id_InstanciaProcesso := id_Processo_Pai;
      Vetor[0].InfoPai.id_AtividadeInvocadora := id_Atividade_Invocadora;
      //Setando o Pai no WFMS
      frmWFMS.PaiProcessoWFMS := GetEngineName( GetIdEngine( url_Engine_Origem ) );
      //Preenchendo informações no vetor de processos
      Vetor[0].Processo.url_WfEngine := url_Engine_Destino;
      Vetor[0].Processo.id_Definicao := id_Def_Processo_Filho;
      Vetor[0].Processo.id_Instancia := iResultado;
      //Estado default quando o processo é criado
      Vetor[0].Processo.str_Estado   := 'open.notrunning';
      Vetor[0].Processo.TempoAtivacao := Now();
```

```
      Vetor[0].Processo.TempoConclusao := GetTime( GET_PROCESS_DEFINITION(
Vetor[0].Processo.id_Instancia ), 'Tempo' );
      Vetor[0].Processo.TempoMaxConclusao := GetTime( GET_PROCESS_DEFINITION(
Vetor[0].Processo.id_Instancia ), 'TempoMaximo' );

      //Atribuindo os Valores do Processo no WFMS
      frmWFMS.ProcessoWFMS := GetProcessName( Vetor[0].Processo.id_Definicao );
      frmWFMS.EstadoProcessoWFMS := Vetor[0].Processo.str_Estado;
      //Obtendo todos os subprocessos do processo
      SubProcessos := GET_SUB_PROCESS( id_Def_Processo_Filho );
      if SubProcessos[0] = 'local' then
        Vetor[0].Local := TRUE
      else
      begin
        //Preenchedo os subprocessos
        for iPos := 0 to MAX_SUBPROCESS do
          Vetor[0].SubProcessos[iPos].id_Definicao:= GetIdDefProcesso(SubProcessos[iPos]);
      end;
      //Escrevendo no Memo a operação de preenchimento de vetor
      frmWFMS.memoOP.Lines.Add( Format(FILL_VECTOR_MSG,[AgoraInicial()]));
    end;//end-if
  end;
  Result:= iResultado;
end;


function TWS_IF4T.ChangeProcessInstanceStateOUTBOUND(
                    url_Engine_Origem,
                    url_Engine_Destino: String;
                    id_Instancia_Processo:Integer;
                    st_Estado: String
                    ): Boolean;
var
  comunicador: IWS_IF4T;
  iWEng, id_Def_Subprocesso, iPos: Integer;
  bResultado: Boolean;
begin
  iWEng:= GetIdEngine( url_Engine_Destino );
  try
    //Monstrando os Eventos
    frmWFMS.memoOUT.Lines.Add(
        Format( CHANGE_PROC_MSG_OUT,
            [AgoraInicial(),GetEngineName(iWEng),id_Instancia_Processo,st_Estado]
          ) );

    //Setando o componente para se comunicar com o workflow Engine de destino
    WebMod.HTTPRIO.WSDLLocation := url_Engine_Destino;
    //Recebendo o TypeCast do Componente
    comunicador:= WebMod.HTTPRIO as IWS_IF4T;
    //Chamando a operação CreateProcessInstance no WF de destino
    bResultado:= comunicador.ChangeProcessInstanceState(
                        url_Engine_Origem,
                        url_Engine_Destino,
                        id_Instancia_Processo,
                        st_Estado
                              );
    //Se resultado for válido
    if bResultado then
    begin
      id_Def_Subprocesso := GET_PROCESS_DEFINITION(id_Instancia_Processo);
      //Localizando o índice do Subprocesso
      iPos := FindSubprocess( id_Def_Subprocesso );
      //Atribuindo o estado ao subprocesso
```

```
      Vetor[0].SubProcessos[iPos].str_Estado := st_Estado;
      //Mostrando a operação
      frmWFMS.memoOP.Lines.Add(          Format(          ATUALIZE_STATE_MSG_OUT,
[AgoraInicial(),id_Instancia_Processo] ));

      //Condição para inicializar o algoritmo executor
      if (st_Estado = 'open.running') and (iPos = 0) then
      begin
        //Aloque processo atual no algoritmo executor
        AtiveAlgoritmoExecutor(0);
      end;
    end;
    //Escrevendo Resposta no Memo
    frmWFMS.memoOUT.Lines.Add(Format(OK_TRANS_MSG, [AgoraInicial()]));
    //Retornando o valor da operação
    Result:= bResultado;
  except
    //Pegando-se o objeto de exceção
    on E: Exception do
    begin
      frmWFMS.NotifyWFMS( Format('Tipo: %s',[ E.ClassName]) );
      //Escrevendo no Memo
      frmWFMS.memoOUT.Lines.Add( ERR_TRANS_MSG );
      //Retornando código de Erro
      Result := FALSE;
    end;
  end;//end try-except
end;

function TWS_IF4T.ChangeProcessInstanceStateINBOUND(
                    url_Engine_Origem,
                    url_Engine_Destino: String;
                    id_Instancia_Processo:Integer;
                    st_Estado: String
                    ): Boolean;
var
  iWEng, iWEng2, iCount :Integer;
begin
  iWEng:= GetIdEngine( url_Engine_Origem  );
  iWEng2:= GetIdEngine( url_Engine_Destino );

  //Monstrando os Eventos
  frmWFMS.memoIN.Lines.Add(
     Format( CHANGE_PROC_MSG_IN,
          [AgoraInicial(),GetEngineName(iWEng), GetEngineName(iWEng2),
           id_Instancia_Processo, st_Estado]
         ) );


  //Preenchendo o Vetor em memória
  //Atribuindo a mudança de estado ao Processo
  Vetor[0].Processo.str_Estado := st_Estado;

  //Mostrando a operação
  frmWFMS.memoOP.Lines.Add(          Format(          ATUALIZE_STATE_MSG_IN,
[AgoraInicial(),id_Instancia_Processo] ));

  //Atualizando o estado do processo no WFMS
  frmWFMS.EstadoProcessoWFMS := Vetor[0].Processo.str_Estado;

  //Se o processo for distribuído
  if not Vetor[0].Local then
  begin
```

```
    if (st_Estado = 'closed.aborted') or (st_Estado = 'closed.terminated')
    or (st_Estado = 'closed.completed') then
    begin
      //Atualize o estado de todos subprocessos
      for iCount := 0 to MAX_SUBPROCESS do
      begin
        if Vetor[0].SubProcessos[iCount].url_WfEngine <> '' then
          ChangeProcessInstanceStateOUTBOUND(
            Vetor[0].Processo.url_WfEngine,
            Vetor[0].SubProcessos[iCount].url_WfEngine,
            Vetor[0].SubProcessos[iCount].id_Instancia,
            st_Estado );
      end;
    end;
    if (st_Estado = 'open.notrunning.suspended') then
    begin
      //Suspenda todos processos ativos
      for iCount := 0 to MAX_SUBPROCESS do
      begin
        if (Vetor[0].SubProcessos[iCount].url_WfEngine <> '') and
          (Vetor[0].SubProcessos[iCount].str_Estado = 'open.running') then
          ChangeProcessInstanceStateOUTBOUND(
            Vetor[0].Processo.url_WfEngine,
            Vetor[0].SubProcessos[iCount].url_WfEngine,
            Vetor[0].SubProcessos[iCount].id_Instancia,
            st_Estado );
      end;
    end;
    if (st_Estado = 'open.running') then
    begin
      //Ative todos os processos suspensos
      for iCount := 0 to MAX_SUBPROCESS do
      begin
        if (Vetor[0].SubProcessos[iCount].url_WfEngine <> '') and
          (Vetor[0].SubProcessos[iCount].str_Estado = 'open.notrunning.suspended') then
          ChangeProcessInstanceStateOUTBOUND(
            Vetor[0].Processo.url_WfEngine,
            Vetor[0].SubProcessos[iCount].url_WfEngine,
            Vetor[0].SubProcessos[iCount].id_Instancia,
            st_Estado );
      end;
    end;
    //Mostrando Mensagens
    frmWFMS.memoOP.Lines.Add(Format( ATUALIZE_SUBPROC_MSG,[AgoraInicial()]));
  end;//end-if

  //Monstrando os Eventos
  frmWFMS.memoIN.Lines.Add( Format( CHANGE_DONE, [AgoraInicial()]) );

   //Operação recebida com sucesso
   Result:=TRUE;
end;

function TWS_IF4T.ProcessInstanceStateChangedOUTBOUND(
                      url_Engine_Origem,
                      url_Engine_Destino: String;
                      id_Instancia_Processo:Integer;
                      st_Estado: String
                      ): Boolean;
var
  comunicador: IWS_IF4T;
  iWEng, iCount: Integer;
  bResultado: Boolean;
```

```
begin
  iWEng:= GetIdEngine( url_Engine_Destino );
  try
    //Monstrando os Eventos
    frmWFMS.memoOUT.Lines.Add(
        Format( PROC_CHANGE_MSG_OUT,
             [AgoraInicial(),GetEngineName(iWEng),id_Instancia_Processo,st_Estado]
           ) );

    //Setando o componente para se comunicar com o workflow Engine de destino
    WebMod.HTTPRIO.WSDLLocation := url_Engine_Destino;
    //Recebendo o TypeCast do Componente
    comunicador:= WebMod.HTTPRIO as IWS_IF4T;
    //Chamando a operação CreateProcessInstance no WF de destino
    bResultado:= comunicador.ProcessInstanceStateChanged(
                          url_Engine_Origem,
                          url_Engine_Destino,
                          id_Instancia_Processo,
                          st_Estado
                                );


    //Se resultado for válido
    if bResultado then
    begin
      //Preenchendo o Vetor em memória
      //Atribuindo a mudança de estado ao Processo
      Vetor[0].Processo.str_Estado := st_Estado;

      //Mostrando a operação
      frmWFMS.memoOP.Lines.Add(          Format(          ATUALIZE_STATE_MSG_IN,
[AgoraInicial(),id_Instancia_Processo] ));

      //Se o processo for distribuído
      if not Vetor[0].Local then
      begin
        if (st_Estado = 'open.running') then
        begin
          //Atualize o estado de todos subprocessos
          for iCount := 0 to MAX_SUBPROCESS do
          begin
            if (Vetor[0].SubProcessos[iCount].url_WfEngine <> '')  and
              //Verificando se subprocesso já não está abortado, terminado ou suspenso
              (Vetor[0].SubProcessos[iCount].str_Estado = 'open.notrunning.suspended') then
            begin
              ChangeProcessInstanceStateOUTBOUND(
                Vetor[0].Processo.url_WfEngine,
                Vetor[0].SubProcessos[iCount].url_WfEngine,
                Vetor[0].SubProcessos[iCount].id_Instancia,
                st_Estado );
            end;
          end;//end-for
        end;
        if (st_Estado = 'closed.aborted') or (st_Estado = 'closed.terminated')
        or (st_Estado = 'open.notrunning.suspended') then
        begin
          //Atualize o estado de todos subprocessos
          for iCount := 0 to MAX_SUBPROCESS do
          begin
            if (Vetor[0].SubProcessos[iCount].url_WfEngine <> '')  and
              //Verificando se subprocesso já não está abortado, terminado ou suspenso
              (Vetor[0].SubProcessos[iCount].str_Estado <> 'closed.aborted') and
              (Vetor[0].SubProcessos[iCount].str_Estado <> 'closed.terminated') and
```

```
              (Vetor[0].SubProcessos[iCount].str_Estado <> 'open.notrunning.suspended') then
            begin
              ChangeProcessInstanceStateOUTBOUND(
                Vetor[0].Processo.url_WfEngine,
                Vetor[0].SubProcessos[iCount].url_WfEngine,
                Vetor[0].SubProcessos[iCount].id_Instancia,
                st_Estado );
            end;
          end;//end-for
        end;//end-if
      end;//end-if
    end;//end-if

    //Escrevendo Resposta no Memo
    frmWFMS.memoOUT.Lines.Add(Format(OK_TRANS_MSG, [AgoraInicial()]));
    //Retornando o valor da operação
    Result:= bResultado;
  except
    //Pegando-se o objeto de exceção
    on E: Exception do
    begin
      frmWFMS.NotifyWFMS( Format('Tipo: %s',[ E.ClassName]) );
      //Escrevendo no Memo
      frmWFMS.memoOUT.Lines.Add( ERR_TRANS_MSG );
      //Retornando código de Erro
      Result := FALSE;
    end;
  end;//end try-except
end;

function TWS_IF4T.ProcessInstanceStateChangedINBOUND(
                      url_Engine_Origem,
                      url_Engine_Destino: String;
                      id_Instancia_Processo:Integer;
                      st_Estado: String
                      ): Boolean;
var
  iWEng, iWEng2,id_Def_Subprocesso,iPos :Integer;
begin
  iWEng:= GetIdEngine( url_Engine_Origem  );
  iWEng2:= GetIdEngine( url_Engine_Destino );

  //Monstrando os Eventos
  frmWFMS.memoIN.Lines.Add(
      Format( PROC_CHANGE_MSG_IN,
           [AgoraInicial(),GetEngineName(iWEng), GetEngineName(iWEng2),
            id_Instancia_Processo, st_Estado]
          ) );

  //Preenchendo o Vetor em memória
  id_Def_Subprocesso := GET_PROCESS_DEFINITION(id_Instancia_Processo);
  //Localizando o índice do Subprocesso
  iPos := FindSubprocess( id_Def_Subprocesso );
  //Atribuindo o  estado ao subprocesso
  Vetor[0].SubProcessos[iPos].str_Estado := st_Estado;
  //Mostrando a operação
  frmWFMS.memoOP.Lines.Add(                Format(                ATUALIZE_STATE_MSG_OUT,
[AgoraInicial(),id_Instancia_Processo] ));

  //Monstrando os Eventos
  frmWFMS.memoIN.Lines.Add( Format( CHANGE_DONE, [AgoraInicial()]) );

   //Operação recebida com sucesso
```

```
   Result:=TRUE;
end;


procedure TWS_IF4T.AtiveAlgoritmoExecutor( iIndexProcess :Integer );
begin
  //Atribuindo o processo que será controlado no algoritmo
  iProcesso := iIndexProcess;

  //Atribuindo os Tempos do Processo aos Timers
  uWebModule.WebMod.MetadeTempo.Interval := ( Round( (Vetor[iIndexProcess].Processo.TempoConclusao
* 0.5))*1000 );
  uWebModule.WebMod.TempoNormal.Interval := ( Vetor[iIndexProcess].Processo.TempoConclusao * 1000 );
  uWebModule.WebMod.TempoMaximo.Interval  :=  (  Vetor[iIndexProcess].Processo.TempoMaxConclusao  *
1000 );

  //Ativando os Timers
  uWebModule.WebMod.MetadeTempo.Enabled := True;
  uWebModule.WebMod.TempoNormal.Enabled := True;
  uWebModule.WebMod.TempoMaximo.Enabled := True;

  //Mostrando mensagem de ativação
  frmWFMS.memoOP.Lines.Add(
    Format( INIT_ALGORITMO_MSG,
          [AgoraInicial(),GetProcessName( Vetor[iIndexProcess].Processo.id_Definicao)]
                ) );
end;

procedure TWS_IF4T.AlgoritmoExecutorMetadeTempo(Sender: TObject);
var
  bAlgumAbortou :Boolean;
  iCount : Integer;
begin
  //Mostrando Mensagem
  frmWFMS.memoOP.Lines.Add( Format( ALGORITMO_TEMPOMEIO_MSG,[AgoraInicial()]));

  bAlgumAbortou := FALSE;
  //Consultando os estados de todos os subprocessos
  for iCount :=0 to (NumeroSubProcessos( iProcesso )-1) do
  begin
    //Se algum Abortou
    if Vetor[iProcesso].SubProcessos[iCount].str_Estado = 'closed.aborted' then
    begin
      //Abortado Subprocessos Anteriores
      ABORTE_SUBPROCESSOS_ANTERIORES( iProcesso, iCount );
      bAlgumAbortou := TRUE;
    end;
  end;

  if bAlgumAbortou or
   (Vetor[iProcesso].Processo.str_Estado = 'closed.aborted') then
  begin
    ABORTE_PROCESSO( iProcesso );
    FINALIZE_ALGORITMO();
  end;

  //Se o Processo estiver abortado
  if (Vetor[iProcesso].Processo.str_Estado = 'closed.aborted') then
  begin
    //Segundo Modelo Sagas
    for iCount:=(NumeroSubProcessos( iProcesso )) downto 0 do
      //Abortado Subprocessos Anteriores
      ABORTE_SUBPROCESSOS_ANTERIORES( iProcesso, iCount );
```

```
    if Vetor[iProcesso].TemPai then
    begin
       ProcessInstanceStateChangedOUTBOUND(
         Vetor[iProcesso].Processo.url_WfEngine,
         Vetor[iProcesso].InfoPai.url_WfEngine,
         Vetor[iProcesso].Processo.id_Instancia,
         'closed.aborted'  );
    end;
    FINALIZE_ALGORITMO();
  end;
  //Finalizando esta fase do algoritmo
  uWebModule.WebMod.MetadeTempo.Enabled := False;
end;

procedure TWS_IF4T.AlgoritmoExecutorTempoNormal(Sender: TObject);
var
  bAlgumAbortou :Boolean;
  iCount, iPreparados :Integer;
begin
  //Mostrando Mensagem
  frmWFMS.memoOP.Lines.Add( Format(ALGORITMO_TEMPONORMAl_MSG,[AgoraInicial()]));

  bAlgumAbortou := False;
  iPreparados := 0;
  //Consultando os estados de todos os subprocessos
  for iCount :=0 to (NumeroSubProcessos( iProcesso )-1) do
  begin
    //Se algum Abortou
    if Vetor[iProcesso].SubProcessos[iCount].str_Estado = 'closed.aborted' then
    begin
      //Abortado Subprocessos Anteriores
      ABORTE_SUBPROCESSOS_ANTERIORES( iProcesso, iCount );
      bAlgumAbortou := TRUE;
    end;
    if Vetor[iProcesso].SubProcessos[iCount].str_Estado = 'open.prepared' then
    begin
      iPreparados := iPreparados + 1;
    end;
  end;//end-for

  if bAlgumAbortou then
  begin
    ABORTE_PROCESSO( iProcesso );
    FINALIZE_ALGORITMO();
  end;

  //Se o Processo estiver abortado
  if (Vetor[iProcesso].Processo.str_Estado = 'closed.aborted') then
  begin
    //Segundo Modelo Sagas
    for iCount:=(NumeroSubProcessos( iProcesso )) downto 0 do
      //Abortado Subprocessos Anteriores
      ABORTE_SUBPROCESSOS_ANTERIORES( iProcesso, iCount );
    if Vetor[iProcesso].TemPai then
    begin
       ProcessInstanceStateChangedOUTBOUND(
         Vetor[iProcesso].Processo.url_WfEngine,
         Vetor[iProcesso].InfoPai.url_WfEngine,
         Vetor[iProcesso].Processo.id_Instancia,
         'closed.aborted'  );
    end;
    FINALIZE_ALGORITMO();
  end;
```

```
  //Se todos os subprocessos estão preparados e processo ainda estiver ativo
  if ( iPreparados = NumeroSubProcessos( iProcesso ) )and
    ( Vetor[iProcesso].Processo.str_Estado = 'open.running' ) then
  begin
    CONFIRME_PROCESSO( iProcesso );
    FINALIZE_ALGORITMO();
  end;

  //Finalizando esta fase do algoritmo
  uWebModule.WebMod.TempoNormal.Enabled := False;
end;

procedure TWS_IF4T.AlgoritmoExecutorTempoMaximo(Sender: TObject);
var
  bAlgumAbortou :Boolean;
  iCount, iPreparados :Integer;

begin
  //Mostrando Mensagem
  frmWFMS.memoOP.Lines.Add( Format( ALGORITMO_TEMPOMAX_MSG,[AgoraInicial()]));

  //Se o processo estiver suspenso, finalize algoritmo e não faça nada
  if ( Vetor[iProcesso].Processo.str_Estado = 'open.notrunning.suspended' ) then
    //Encerrando o algoritmo
    FINALIZE_ALGORITMO()
  else
  begin
    //Execute as validações
    bAlgumAbortou := False;
    iPreparados := 0;
    //Consultando os estados de todos os subprocessos
    for iCount :=0 to (NumeroSubProcessos( iProcesso )-1) do
    begin
      //Se algum Abortou
      if Vetor[iProcesso].SubProcessos[iCount].str_Estado = 'closed.aborted' then
      begin
        //Abortado Subprocessos Anteriores
        ABORTE_SUBPROCESSOS_ANTERIORES( iProcesso, iCount );
        bAlgumAbortou := TRUE;
      end;
      if Vetor[iProcesso].SubProcessos[iCount].str_Estado = 'open.prepared' then
      begin
        iPreparados := iPreparados + 1;
      end;
    end;//end-for

    if bAlgumAbortou then
    begin
      ABORTE_PROCESSO( iProcesso );
      FINALIZE_ALGORITMO();
    end;

      //Se o Processo estiver abortado
    if (Vetor[iProcesso].Processo.str_Estado = 'closed.aborted') then
    begin
      //Segundo Modelo Sagas
      for iCount:=(NumeroSubProcessos( iProcesso )) downto 0 do
        //Abortado Subprocessos Anteriores
        ABORTE_SUBPROCESSOS_ANTERIORES( iProcesso, iCount );
      if Vetor[iProcesso].TemPai then
      begin
        ProcessInstanceStateChangedOUTBOUND(
```

```
          Vetor[iProcesso].Processo.url_WfEngine,
          Vetor[iProcesso].InfoPai.url_WfEngine,
          Vetor[iProcesso].Processo.id_Instancia,
          'closed.aborted'  );
        end;
        FINALIZE_ALGORITMO();
      end;


      //Se todos os subprocessos estão preparados e processo ainda estiver ativo
      if ( iPreparados = NumeroSubProcessos( iProcesso ) )and
        ( Vetor[iProcesso].Processo.str_Estado = 'open.running' ) then
      begin
        CONFIRME_PROCESSO( iProcesso );
        FINALIZE_ALGORITMO();
      end;

      //Finalizando esta fase do algoritmo
      uWebModule.WebMod.TempoMaximo.Enabled := False;
    end;//end-else
end;

procedure TWS_IF4T.FINALIZE_ALGORITMO();
begin
  if uWebModule.WebMod.MetadeTempo.Enabled then
    uWebModule.WebMod.MetadeTempo.Enabled := False;

  if uWebModule.WebMod.TempoNormal.Enabled then
    uWebModule.WebMod.TempoNormal.Enabled := False;

  if uWebModule.WebMod.TempoMaximo.Enabled then
    uWebModule.WebMod.TempoMaximo.Enabled := False;
  //Mostrando Mensagem
  frmWFMS.memoOP.Lines.Add( Format(FINAL_ALGORITMO_MSG,[AgoraInicial()]));
end;

procedure TWS_IF4T.CONFIRME_PROCESSO( iIndexProcess :Integer );
var
  iCount :Integer;
begin
  //Mostrando mensagem
  frmWFMS.memoOP.Lines.Add( Format(CONFIRME_PROCESSO_MSG,[AgoraInicial()]) );
  //Se o processo tem Pai, notifique e aguarde confirmação
  if Vetor[iIndexProcess].TemPai then
  begin
    ProcessInstanceStateChangedOUTBOUND(
      Vetor[iIndexProcess].Processo.url_WfEngine,
      Vetor[iIndexProcess].InfoPai.url_WfEngine,
      Vetor[iIndexProcess].Processo.id_Instancia,
      'open_prepared'  );
    frmWFMS.memoOP.Lines.Add(                                                    Format(
WAIT_CONFIRMATION_MSG,[AgoraInicial(),GetProcessName(
Vetor[iIndexProcess].InfoPai.id_DefinicaoProcesso) ]) );
  end
  else
  begin
    //Confirmando todos os Subprocessos
    for iCount := 0  to ( NumeroSubProcessos( iIndexProcess )-1 ) do
    begin
      ChangeProcessInstanceStateOUTBOUND(
        Vetor[iIndexProcess].Processo.url_WfEngine,
        Vetor[iIndexProcess].SubProcessos[iCount].url_WfEngine,
        Vetor[iIndexProcess].SubProcessos[iCount].id_Instancia,
```

```
          'closed.completed' );
      end;
     //Confirmando o processo
     Vetor[iIndexProcess].Processo.str_Estado := 'closed.completed';
     //Notificando ao WFMS
     frmWFMS.EstadoProcessoWFMS := 'closed.completed';
  end;
end;

procedure TWS_IF4T.ABORTE_PROCESSO( iIndexProcess :Integer );
begin
  //Abortando o processo
  Vetor[iIndexProcess].Processo.str_Estado := 'closed.aborted';
  //Notificando ao WFMS
  frmWFMS.EstadoProcessoWFMS := 'closed.aborted';
  //Se o processo tem Pai, notifique
  if Vetor[iIndexProcess].TemPai then
  begin
       ProcessInstanceStateChangedOUTBOUND(
          Vetor[iIndexProcess].Processo.url_WfEngine,
          Vetor[iIndexProcess].InfoPai.url_WfEngine,
          Vetor[iIndexProcess].Processo.id_Instancia,
          'closed.aborted'  );
  end;
  //Mostrando Mensagem
  frmWFMS.memoOP.Lines.Add( Format(ABORT_PROCESSO_MSG,[AgoraInicial()]) );

end;

procedure TWS_IF4T.ABORTE_SUBPROCESSOS_ANTERIORES( iIndexProcess ,
iSubprocessAborted:Integer);
var
  iCount :Integer;
begin
  //Abortando subprocessos anteriores
  for iCount := (iSubprocessAborted -1) downto 0 do
  begin
    //Se o processo já estava preparado, compense
    if Vetor[iIndexProcess].SubProcessos[iCount].str_Estado = 'open.prepared' then
         ChangeProcessInstanceStateOUTBOUND(
           Vetor[iIndexProcess].Processo.url_WfEngine,
           Vetor[iIndexProcess].SubProcessos[iCount].url_WfEngine,
           Vetor[iIndexProcess].SubProcessos[iCount].id_Instancia,
           'closed.terminated' );

    //Para Evitar que um subprocesso seja abortado duas vezes
    if (Vetor[iIndexProcess].SubProcessos[iCount].str_Estado <> 'closed.aborted') and
      (Vetor[iIndexProcess].SubProcessos[iCount].str_Estado <> 'closed.terminated')
    then
         ChangeProcessInstanceStateOUTBOUND(
           Vetor[iIndexProcess].Processo.url_WfEngine,
           Vetor[iIndexProcess].SubProcessos[iCount].url_WfEngine,
           Vetor[iIndexProcess].SubProcessos[iCount].id_Instancia,
           'closed.aborted' );
  end;
  //Verificando senão havia nenhum processo posterior ao abortado que
  //estava ativo e com isso tem que ser tbém abortado
  for iCount := (NumeroSubProcessos( iIndexProcess ) -1) downto (iSubprocessAborted +1) do
  begin
    if Vetor[iIndexProcess].SubProcessos[iCount].str_Estado = 'open.running' then
         ChangeProcessInstanceStateOUTBOUND(
           Vetor[iIndexProcess].Processo.url_WfEngine,
           Vetor[iIndexProcess].SubProcessos[iCount].url_WfEngine,
```

```
            Vetor[iIndexProcess].SubProcessos[iCount].id_Instancia,
            'closed.aborted' );
  end;
  //Mostrando mensagem
  if  FrmWFMS.memoOP.Lines[(FrmWFMS.memoOP.Lines.Count)-1]  <>  ABORT_SUBPROCESSO_MSG
then
    frmWFMS.memoOP.Lines.Add( Format( ABORT_SUBPROCESSO_MSG, [AgoraInicial()]));

end;
    initialization
 { Invokable classes must be registered }
 InvRegistry.RegisterInvokableClass(TWS_IF4T);

end.
```

unit uMessages;

```
interface

////////////////////////////////////////////////////////
//// Definição das Mensagens do WFMS
////////////////////////////////////////////////////////
const NOT_FOUND_WENG = 'O Workflow Engine digitado não foi encontrado no BD.';
const NOT_FOUND_PROC = 'O processo escolhido não foi encontrado no BD.';
const NOT_FOUND_INST = 'Essa instância de processo não foi encontrada no sistema.';
const NOT_FOUND_STATE= 'Você deve digitar um dos estados para o processo(ative, suspenda, complete,
compense ou aborte)';
const NOTIFY_WS_IF4T = 'Notificação WS IF4T: %s';

////////////////////////////////////////////////////////////
//// Definição das Mensagens da WS IF4T
////////////////////////////////////////////////////////////
const ERR_INVALID_PROCESS_DEFINITION = '   Erro: WF_INVALID_PROCESS_DEFINITION';
const ERR_ENGINE_MSG ='   Erro: WF_ENGINE_ERROR';
const ERR_TRANS_MSG ='   Erro: Ocorreu algum erro na comunicação';

const CREATE_PROC_MSG_OUT = '[%s] CreateProcessInstance(%s,%s,%d,%d)';
const CHANGE_PROC_MSG_OUT = '[%s] ChangeProcessInstanceState(%s,%d,%s)';
const PROC_CHANGE_MSG_OUT = '[%s] ProcessInstanceStateChanged(%s,%d,%s)';

const CREATE_PROC_MSG_IN = '[%s] Op. de %s: CreateProcessInstance(%s,%s,%d,%d)';
const CHANGE_PROC_MSG_IN = '[%s] Op. de %s: ChangeProcessInstanceState(%s,%d,%s)';
const PROC_CHANGE_MSG_IN = '[%s] Op. de %s: ProcessInstanceStateChanged(%s,%d,%s)';

const PROCESS_CREATED ='[%s] Resp: Processo "%s" criado, instância %d';
const CHANGE_DONE     ='[%s] Resp: Estado do Processo alterado';

const OK_TRANS_MSG =   [%s] Resp: Operação Efetuada com sucesso';
const FILL_VECTOR_MSG= '[%s] Preenchendo Informações no vetor de processos';
const ATUALIZE_STATE_MSG_OUT = '[%s] Atualizando o estado da instância %d no vetor de subprocessos';
const ATUALIZE_STATE_MSG_IN = '[%s] Atualizando o estado da instância %d no vetor de processos';
const ATUALIZE_SUBPROC_MSG ='[%s] Atualizando o estado dos subprocessos distribuídos';

const INIT_ALGORITMO_MSG = '[%s] O processo "%s" foi alocado no algoritmo executor';
const ABORT_PROCESSO_MSG = '[%s] O processo do Algoritmo Executor foi abortado';
const  ABORT_SUBPROCESSO_MSG  =  '[%s]  Os  Subprocessos   do  Processo  foram  abortados  ou
compensados';

const WAIT_CONFIRMATION_MSG ='[%s] Aguardando confirmação do processo "%s"';
const CONFIRME_PROCESSO_MSG = '[%s] O Processo do Algoritmo Executor está sendo confirmado';
const FINAL_ALGORITMO_MSG = '[%s] O Algoritmo Executor foi Finalizado';
```

```
const ALGORITMO_TEMPOMEIO_MSG = '[%s] Tempo Processo = Tempo Normal / 2';
const ALGORITMO_TEMPONORMAl_MSG = '[%s] Tempo Processo = Tempo Normal';
const ALGORITMO_TEMPOMAX_MSG = '[%s] Tempo Processo = Tempo Máximo';

implementation
end.
```

{ SOAP Web Module - WAD }

unit uWebModule;

```
interface

uses
  SysUtils, Classes, HTTPApp, InvokeRegistry, WSDLIntf, TypInfo,
  WebServExp, WSDLBind, XMLSchema, WSDLPub, SOAPPasInv, SOAPHTTPPasInv,
  SOAPHTTPDisp, WebBrokerSOAP, Rio, SOAPHTTPClient, ExtCtrls,WS_IF4T_Impl;

type
  TWebMod = class(TWebModule)
    HTTPSoapDispatcher: THTTPSoapDispatcher;
    HTTPSoapPascalInvoker: THTTPSoapPascalInvoker;
    WSDLHTMLPublish: TWSDLHTMLPublish;
    HTTPRIO: THTTPRIO;
    MetadeTempo: TTimer;
    TempoNormal: TTimer;
    TempoMaximo: TTimer;
    procedure WebModule2DefaultHandlerAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure MetadeTempoTimer(Sender: TObject);
    procedure TempoNormalTimer(Sender: TObject);
    procedure TempoMaximoTimer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebMod: TWebMod;

var
  WS_IF4T: TWS_IF4T;

implementation

uses WebReq;

{$R *.dfm}

procedure TWebMod.WebModule2DefaultHandlerAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  WSDLHTMLPublish.ServiceInfo(Sender, Request, Response, Handled);
end;

procedure TWebMod.MetadeTempoTimer(Sender: TObject);
begin
   WS_IF4T.AlgoritmoExecutorMetadeTempo(Sender);
end;
```

```
procedure TWebMod.TempoNormalTimer(Sender: TObject);
begin
   WS_IF4T.AlgoritmoExecutorTempoNormal(Sender);
end;

procedure TWebMod.TempoMaximoTimer(Sender: TObject);
begin
  WS_IF4T.AlgoritmoExecutorTempoMaximo(Sender);
end;

initialization
 WebRequestHandler.WebModuleClass := TWebMod;

end.
```

unit uDataModule;

```
interface

uses
 SysUtils, Classes, DB, ADODB;

type
 TDM = class(TDataModule)
   dsConversao: TDataSource;
   tblConversao: TADOTable;
   tblNomeProcesso: TADOTable;
   ADOConnection1: TADOConnection;
   tblNomeWEngine: TADOTable;
   tblProcessos: TADOTable;
   dsNomeProcesso: TDataSource;
   dsNomeWEngine: TDataSource;
   dsProcessos: TDataSource;
   tblNomeWEngineid_WorkFlow_Engine: TIntegerField;
   tblNomeWEngineWorkFlow_Engine: TWideStringField;
   tblProcessosid_Processo: TWideStringField;
   tblProcessosid_Subprocessos: TWideStringField;
   tblProcessosTempo: TSmallintField;
   tblProcessosTempoMaximo: TSmallintField;
   tblNomeProcessoid_Def_Processo: TIntegerField;
   tblNomeProcessoNome_Processo: TWideStringField;
   tblNomeProcessoinst_Processo: TIntegerField;
   tblConversaoid_WorkFlow_Engine: TIntegerField;
   tblConversaourl_WorkFlow_Engine: TWideStringField;
   tblProcessosid_Def_Processo: TIntegerField;
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 DM: TDM;

implementation

{$R *.dfm}
end.
```