

**Publicação e Integração de
Workflows Científicos na Web**

Gilberto Zonta Pastorello Jr.

Dissertação de Mestrado

Publicação e Integração de *Workflows* Científicos na Web

Gilberto Zonta Pastorello Jr.¹

Março de 2005

Banca Examinadora:

- Profa. Dra. Claudia Bauzer Medeiros (Orientadora)
- Prof. Dr. Mauro Biajiz
Departamento de Computação – Universidade Federal de São Carlos
- Prof. Dr. Jacques Wainer
- Prof. Dr. Edmundo R. M. Madeira (Suplente)

¹Este trabalho teve o apoio do CNPq e dos projetos WebMaps, Agroflow e MCT/Pronex.

Publicação e Integração de *Workflows* Científicos na Web

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Gilberto Zonta Pastorello Jr. e aprovada pela Banca Examinadora.

Campinas, 06 de abril de 2005.

Profa. Dra. Claudia Bauzer Medeiros
(Orientadora)

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

© Gilberto Zonta Pastorello Jr., 2005.
Todos os direitos reservados.

*Bem começado
é metade feito.*

Aritóteles, in *Politica*.

Resumo

Atividades científicas envolvem processos complexos e, freqüentemente, diversos especialistas. Além disso, são multidisciplinares e exigem trabalho cooperativo. Isso traz à tona uma série de problemas em Ciência da Computação para apoiar esse trabalho, indo desde a gestão de dados e processos até interfaces adequadas para aplicativos. Esta dissertação contribui na direção do provimento de soluções para alguns desses problemas. Seu foco é em aprimorar os mecanismos de documentação de processos e em possibilitar sua publicação e integração na Web. Isso facilita a especificação e execução de processos distribuídos na Web e o reuso dessas especificações. O trabalho desenvolvido se apóia em padrões da Web Semântica, visando interoperabilidade, e no uso de workflows científicos para modelar os processos e sua utilização na Web. As principais contribuições deste trabalho são: (i) um modelo de dados, tendo em vista padrões da Web Semântica, para representar workflows científicos e seu armazenamento em bancos de dados. O modelo induz uma metodologia de especificação de workflows que facilita seu reuso e integração; (ii) uma análise comparativa das propostas de padrões para representar workflows em XML; (iii) a proposta de uma arquitetura voltada para a Web para o gerenciamento de documentos (workflows, principalmente); e, (iv) a implementação parcial dessa proposta de arquitetura. O trabalho utiliza como domínio alvo a área de planejamento ambiental, para elucidar requisitos e validar a proposta.

Abstract

Scientific activities involve complex multidisciplinary processes and demand cooperative work. This entails a series of open problems in supporting this work ranging from data and process management to appropriate user interfaces for softwares. This work contributes in providing solutions to some of these problems. It focuses on improving the documentation mechanisms of processes and making it possible to publish and integrate them on the Web. This eases the specification and execution of distributed processes on the Web as well as the reuse of these specifications. The work was based on Semantic Web standards aiming at interoperability and the use of scientific workflows for modeling processes and using them on the Web. The main contributions of this work are: (i) a data model, which takes Semantic Web standards into consideration, for representing scientific workflows and storing them in a database. The model induces a workflow specification method that favors reuse and integration of these specifications; (ii) a comparative analysis of standards proposals for representing workflows in XML; (iii) the proposal of a Web-centered architecture for the management of documents (mainly workflows); and, (iv) the partial implementation of this architecture. The work uses as a motivation the area of environmental planning as a means to elucidate requirements and validate the proposal.

Sumário

	vii
Resumo	viii
Abstract	ix
1 Introdução	1
2 Revisão Bibliográfica	3
2.1 Planejamento Ambiental	3
2.2 Sistemas de Workflow	6
2.2.1 Conceitos Básicos	7
2.2.2 O Sistema WOODSS	10
2.3 Serviços Web e a Web Semântica	12
2.4 Padrões Para Representação de Workflows na Web	14
2.5 Reuso e Workflows	15
2.6 Conclusões	16
3 Modelo Proposto para Representação de Workflows	18
3.1 O Novo Modelo de Dados para Workflows no WOODSS	18
3.1.1 O modelo de dados original do WOODSS	19
3.1.2 O modelo de dados proposto	20
3.2 Representação de Workflows em XML	30
3.2.1 XPDL	30
3.2.2 WS-BPEL	34
3.3 Representação Visual de Workflows	38
3.3.1 UML	39
3.3.2 BPMN	44
3.4 Conclusões	50

4	Arquitetura	52
4.1	Visão Geral	52
4.2	Visão Detalhada	56
4.2.1	Visão lógica	56
4.2.2	Visão física	58
4.2.3	Visão de processo	62
4.3	Conclusões	64
5	Aspectos de Implementação	65
5.1	Do Modelo Proposto ao Sistema Gerenciador de Banco de Dados Relacional	65
5.2	Traduzindo entre os Modelos Relacional e de WS-BPEL	68
5.3	Módulo Tradutor na Arquitetura Proposta	78
5.4	Conclusões	80
6	Conclusões e Extensões	81
6.1	Contribuições	81
6.2	Extensões	83
	Bibliografia	86
A	<i>Script</i> para Criação do Banco de Dados em PostgreSQL	92
B	Estruturas em XMLSchema WS-BPEL	97
B.1	XMLSchema para <i>partnerLinkTypes</i>	97
B.2	XMLSchema para WSDL	98
B.3	XMLSchema para WS-BPEL	103

Lista de Figuras

2.1	Cenários de desenvolvimento para o Vale do Itajaí (extraído de [Fra00]). . .	4
2.2	Adequabilidade de solo para <i>Coffea arabica</i> no Paraná (extraído de [FLP ⁺ 03]).	5
2.3	Modelo da WfMC para representação de workflows (extraído de [WfM95]).	8
2.4	Modelo da WfMC de arquitetura de um SGWf (extraído de [WfM95]). . .	9
2.5	Arquitetura do sistema WOODSS (extraída de [Roc03]).	11
2.6	Cópia de tela do sistema WOODSS.	11
2.7	Os padrões para a Web Semântica e Serviços Web	12
3.1	Modelo de Dados Original	19
3.2	Níveis de abstração na especificação de um workflow	21
3.3	Modelo de Dados Proposto	23
3.4	Ilustração do conceito de conectores e seu comportamento	26
3.5	Ilustração do conceito atributos de modos de controle	27
3.6	Funcionamento de elementos de <i>WorkflowConector</i>	28
3.7	Funcionamento do atributo validade aplicado a ciclos	29
3.8	Modelo de dados para a proposta XPDL	31
3.9	Modelo de dados para a proposta WS-BPEL: processo, atividades básicas e transições	35
3.10	Modelo de dados para a proposta WS-BPEL: atividades estruturadas . . .	37
3.11	Comparação entre o Modelo de Dados Proposto, XPDL e WS-BPEL . . .	51
4.1	Visão de funcionamento desejado em alto nível.	53
4.2	Proposta de arquitetura em alto nível para o sistema WOODSS – camada intermediária da Figura 4.1.	54
4.3	Principais módulos do sistema detalhados em pacotes	57
4.4	Distribuição física do sistema em nós de processamento	60
5.1	Tradução do modelo EER para o modelo relacional.	66
5.2	Presença do módulo tradutor na proposta de arquitetura.	79
5.3	Cópia de tela da Ferramenta de Criação do sistema WOODSS	80

Capítulo 1

Introdução

Atividades científicas usam processos complexos e dependem de diversos especialistas. Além disso, são multidisciplinares e exigem trabalho cooperativo. A utilização de ferramentas computacionais no apoio a essas atividades vem aumentando e com isso vem crescendo também a quantidade de dados e processos que se pode utilizar para a solução de um problema. Isso faz necessário criar mecanismos para se gerenciar esses dados e processos.

A abordagem cooperativa fica mais ampla no contexto da Web, que aumenta o número de participantes e as possibilidades de usos de dados e ferramentas. Workflows têm se mostrado um paradigma eficiente para o gerenciamento e execução de processos científicos. Isso é uma ajuda significativa no caso de execução distribuída. Além disso, ferramentas baseadas em workflows podem ser utilizadas para documentar tais processos. Tal documentação se torna cada vez mais necessária no contexto da Web, facilitando o reuso de ferramentas e soluções.

O objetivo deste trabalho é dar início a um processo de apoio a desenvolvimento de atividades científicas na Web. Para isso, dois fatores são considerados: a utilização do sistema WOODSS, um sistema de apoio à decisão espacial, desenvolvido no Laboratório de Sistemas de Informação da UNICAMP, adaptando o seu funcionamento em ambiente *desktop* para a Web; e a adoção de padrões e princípios da Web Semântica. Isso permite, como será visto no decorrer do texto, um apoio efetivo à especificação e documentação de experimentos científicos na Web, de forma distribuída e cooperativa. Com isso busca-se suprir uma necessidade de reuso de especificações de workflows. Dessa forma, a dissertação trata do conceito de workflow em dois sentidos. O primeiro, padrão, é o de modelo ou automatização de um processo; o segundo trata um workflow como um documento que descreve como um processo deve ser executado. Como se verá, uma das contribuições consiste em um modelo genérico para representar workflows, compatível com a Web, que permite reuso de workflows para execução (sentido de processo) e para especificação

(documentação).

Outros aspectos relacionados com o trabalho incluem os esforços sobre a Web Semântica e serviços Web. A iniciativa da Web Semântica visa aumentar o grau de automatização de tarefas na Web oferecendo uma infra-estrutura para descrever e utilizar automaticamente recursos disponíveis. Serviços Web aparecem nesse ponto como um meio de implementação dessas idéias, ou seja, uma forma de fazer funcionar a automatização de tarefas. Isso vem a ajudar nos quesitos de gerenciamento de recursos.

A especificação de composição de serviços Web, segundo algumas iniciativas, faz uso de workflows. Como essas propostas são para o ambiente Web, as especificações são feitas em documentos XML. Além disso, existem algumas propostas de padrões para representar workflows na Web. Entretanto não é claro qual será efetivamente adotada como padrão na Web, já que a escolha depende, além de uma comparação técnica de representatividade e usabilidade, de interesses comerciais envolvidos.

O trabalho de estender o WOODSS para a Web exigiu não apenas modificar o sistema, mas definir a partir de um estudo comparativo, os padrões e propostas (ainda não aceitas definitivamente como padrões) da Web a se adotar. O foco do trabalho foi nas tarefas de documentação e suporte à cooperação. Partes do trabalho foram contempladas em um artigo que será publicado pelo *Journal of Data Semantics*.

As principais contribuições desta dissertação são:

- Um modelo de dados para representar workflows em diferentes níveis de abstração e capaz de expressar estruturas complexas. Este modelo induz os projetistas de experimentos científicos a uma metodologia incremental de especificação de workflows;
- Análise comparativa das propostas de padrão para representar workflows em XML e adaptação de WS-BPEL para suportar o modelo proposto;
- Proposta de arquitetura para gerenciamento de documentos e execução de workflows na Web;
- Implementação parcial da arquitetura proposta.

O restante do texto está organizado da seguinte forma. O Capítulo 2 apresenta uma revisão dos conceitos relacionados ao trabalho. O Capítulo 3 apresenta a proposta do novo modelo de dados para representar workflows e discute o método de especificação induzido pelo modelo. O Capítulo também discute as duas principais propostas de representação de workflows na Web, bem como propostas para notação gráfica de workflows. O Capítulo 4 apresenta a proposta de arquitetura, descrevendo-a em pontos de vista lógico, físico e de processos. O Capítulo 5 traz uma discussão sobre aspectos de implementação. Por fim, o Capítulo 6 traz as conclusões do trabalho, listando as contribuições e possibilidades para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta os conceitos relacionados com este trabalho através da revisão da bibliografia correspondente. A seção 2.1 apresenta a área que serviu como motivação para este trabalho, planejamento ambiental. Em seguida, a seção 2.2 apresenta sistemas de workflows, que compõem a abordagem utilizada para auxiliar a solução de problemas em planejamento ambiental. A seção 2.3 apresenta serviços Web, que são importantes para este trabalho sob dois aspectos: (i) a extensão do sistema para áreas de aplicação mais genéricas na Web (disponibilizadas como serviços Web); e (ii) a utilização de workflows nas propostas de padrões de composição de serviços na Web em XML. As representações de workflows na Web são tratadas separadamente na seção 2.4.

2.1 Planejamento Ambiental

Planejamento ambiental, neste trabalho, é interpretado como o desenvolvimento de qualquer plano que envolva alterações em alguma região geográfica. Existem outras definições para o termo que consideram idéias como exploração e preservação de recursos existentes [SMRY99] ou valorização e conservação da base natural de um território [Fra00]. Outras definições estabelecem ainda restrições nas possíveis áreas de atuação de quem realiza um planejamento ambiental como limitá-las a controle de poluição, avaliação de impacto ambiental e planejamento de uso do solo [Ort84]. Entretanto, uma definição mais genérica é usada neste trabalho para evitar restrições no escopo de possíveis estudos de caso.

Desenvolver atividades de planejamento ambiental é, freqüentemente, uma tarefa complexa que pode depender do uso de conhecimento advindo de muitos especialistas de diferentes áreas.

Para o desenvolvimento de atividades de planejamento ambiental é necessário fazer referência às coordenadas geográficas da área coberta. Atualmente tais atividades são fortemente centradas no uso de Sistemas de Informação Geográfica [SCP97, SMRY99].

Sistemas de Informação Geográfica (SIGs) são definidos como sistemas automatizados usados para armazenar, analisar e manipular dados geo-referenciados, ou seja, dados que representam objetos e fenômenos para os quais a localização geográfica é uma característica inerente à informação e indispensável para analisá-la [CCH⁺96].

Dois exemplos serão usados para ilustrar aspectos diversos envolvidos em atividades de planejamento ambiental. O primeiro está relacionado com o desenvolvimento da região do Vale do Itajaí no estado de Santa Catarina [Fra00] e o segundo com planejamento de uso de solo para a cultura de café no estado do Paraná [FLP⁺03, CCW⁺01].

Os principais aspectos de desenvolvimento considerados para o planejamento de desenvolvimento da região do Vale do Itajaí em [Fra00] foram: (i) preservação ambiental; (ii) desenvolvimento tecnológico; e (iii) a exploração racional das potencialidades de turismo ecológico da região. A metodologia de desenvolvimento do plano de ação envolveu a criação de três grupos de especialistas, que contavam com o apoio de membros da comunidade e de associações locais. Cada grupo ficou encarregado de um dos aspectos mencionados. Foram vários os tipos de dados utilizados, como informações sobre a colonização do território advinda de movimentos migratórios, recursos de transporte instalados, recursos naturais e paisagísticos, temperatura, índices de chuva, altimetria, tipos de solo, recursos hídricos, áreas de alagamento, vegetação e uso de solo, indústria e urbanização, além de outros dados providos pelos especialistas e pelos membros da comunidade. O modelo gerado para o trabalho é ilustrado na Figura 2.1, na qual as atividades desenvolvidas pelos grupos são representadas por *Grupo1*, *Grupo2* e *Grupo3*.

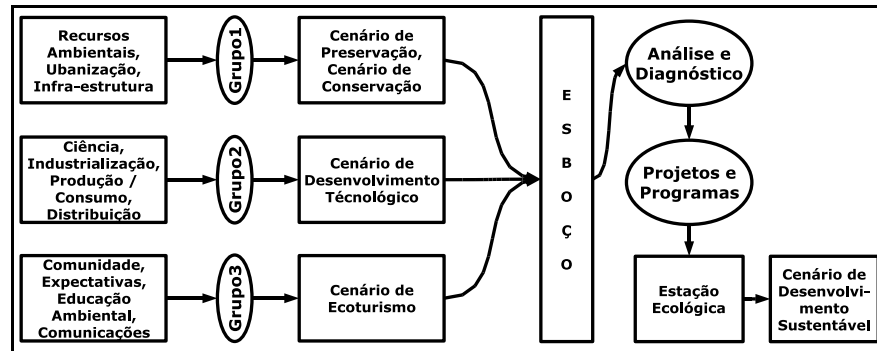


Figura 2.1: Cenários de desenvolvimento para o Vale do Itajaí (extraído de [Fra00]).

Como o modelo apresentado está em um nível de detalhe bastante restrito, não é possível identificar as atividades mais básicas dentro de cada elipse de transformação dos dados.

O resultado do desenvolvimento dessas atividades utilizando conjuntos de dados adequados é um mapeamento das áreas geográficas com maior potencial para cada cenário gerado. Além disso, um mapeamento similar poderia ter sido obtido para uma outra

região alterando-se as fontes de dados e alguns parâmetros na re-execução das atividades.

Uma área de planejamento ambiental que também é de interesse é o *zoneamento agrícola*, que é definido como um processo científico para determinar a adequabilidade de solos, em uma região geográfica, para certas culturas [FLP⁺03]. Zoneamento agrícola é uma atividade de planejamento ambiental que tenta identificar, para cada porção de uma região a melhor cultura de plantio. Outros resultados desse tipo de atividade incluem a determinação dos melhores períodos do ano para a realização de atividades de cultivo, como plantio, colheita, poda, etc.

Como a maioria das atividades de planejamento ambiental, o zoneamento agrícola utiliza o conhecimento de muitos tipos de especialistas para construir um modelo de avaliação de uso do solo. Uma grande parte dos dados necessários é proveniente de sensores remotos – estações de meteorologia, satélites, etc. – usados para coletar dados sobre fenômenos físicos e biológicos. Os dados disponibilizados por esses dispositivos freqüentemente são heterogêneos e sujeitos a variação de qualidade [MA99, Lim03].

A Figura 2.2 ilustra as diferentes etapas necessárias para realizar um estudo de zoneamento agrícola, a partir de especificações de especialistas. Os modelos matemáticos e as tarefas executadas estão representados dentro das elipses. Cada uma destas etapas é potencialmente executada por diferentes especialistas em vários locais. Vale notar que, exceto pelas características biológicas da cultura em questão, todos os outros dados usados no processo são geograficamente referenciados. Portanto, SIG precisam ser usados em todos sub-processos.

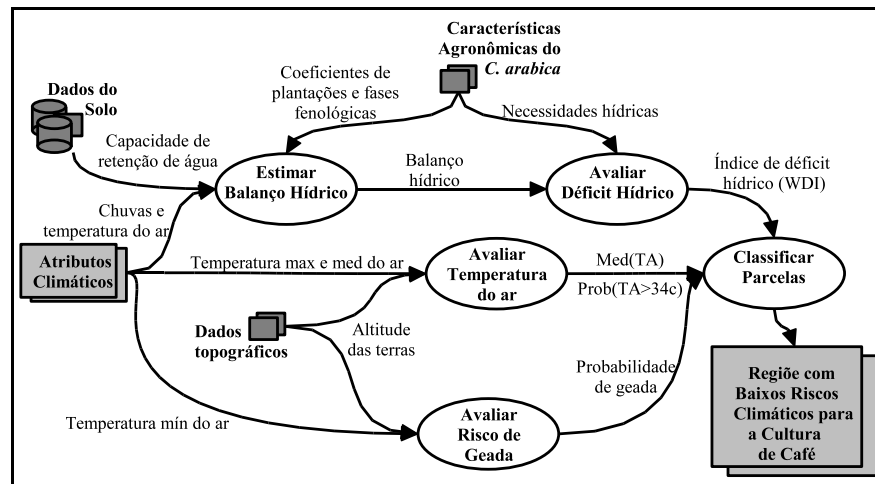


Figura 2.2: Adequabilidade de solo para *Coffea arabica* no Paraná (extraído de [FLP⁺03]).

O resultado final é um mapa mostrando as porções de solo adequadas ao cultivo de *C. arabica* no estado do Paraná [CCW⁺01]. Entretanto, o interesse pelo modelo não se restringe a essa cultura específica ou somente ao estado do Paraná. Com algumas modi-

ficações, o mesmo modelo poderia ser reutilizado. Como um exemplo, o modelo poderia ser utilizado para o estado de São Paulo substituindo as fontes de dados geográficos e talvez algum dado sobre as necessidades da cultura.

Os dois exemplos desta seção mostram como atividades de planejamento ambiental constituem um processo complexo e de múltiplas etapas, executado de forma cooperativa. Esses tipos de processo são exemplos de workflows científicos. A seção 2.2 detalha aspectos de workflows e discute workflows científicos.

2.2 Sistemas de Workflow

É crescente o uso de sistemas baseados em workflows tanto para descrever processos corporativos quanto científicos. As vantagens mais conhecidas da aplicação dessas técnicas incluem o aumento na eficácia na utilização de recursos, automação de tarefas e melhora em aspectos organizacionais como a diminuição de falhas de alocação ou paralisação de tarefas pelo aumento da capacidade de gerenciamento [All01, Ple02]. Além das vantagens relacionadas com o gerenciamento de tarefas individuais, tem-se percebido vantagens relacionadas ao gerenciamento de dados e de processos como um todo [CMC03, CMC⁺02].

Sistemas de workflows são quaisquer sistemas que façam uso de workflows como técnica para modelagem e execução de processos. Neste trabalho o foco de interesse é em sistemas de workflows automatizados, i.e., que utilizam ferramentas computacionais para a modelagem e/ou execução de processos. Os conceitos relacionados a esses sistemas são melhor definidos a seguir.

Vale notar que neste trabalho o foco será em processos científicos, ou seja, processos relacionados ao desenvolvimento de experimentos científicos. Outro enfoque usual, que não será explorado aqui, é o de processos de negócios que lidam com a especificação de processos relacionados à lógica comercial entre entidades.

Processos podem envolver atividades em locais diferentes e demandar cooperação de vários parceiros. Isso exige aprimorar a representação para especificações de workflows. Dessa necessidade surgiram alguns consórcios para tratar de aspectos de padronização. Dentre eles os de maior destaque são a *Workflow Management Coalition* (WfMC) [WfMa] e a *Business Process Management Initiative* (BPMI) [BPMA].

A WfMC existe desde 1993, e teve um papel importante na definição inicial dos conceitos básicos, de um modelo de referência que engloba representação de dados e uma arquitetura genérica para sistema de workflows. Entretanto, a busca de especificações genéricas prejudicou a padronização, deixando lacunas importantes como a representação de algumas estruturas complexas em workflows. Já a BPMI é uma iniciativa recente que busca tratar da representação e colaboração interinstitucional através de processos de negócios (representados como workflows). Essa entidade tem um foco maior em aspectos

computacionais e tem como objetivo dar apoio à colaboração.

2.2.1 Conceitos Básicos

Esta seção define alguns conceitos básicos relacionados com workflows, a partir do modelo de referência da WfMC.

Processo. Um *processo* é um conjunto de um ou mais procedimentos interdependentes que, coletivamente, cumprem um objetivo, normalmente dentro do contexto de uma estrutura organizacional definindo papéis (funções) e relacionamentos. É usual a divisão entre *processos de negócio* e *processos científicos*, diferenciados pelo contexto em que estão inseridos, respectivamente: envolvendo trocas comerciais e/ou monetárias; e contemplando a realização de experimentação científica.

Workflow. Um *workflow* representa a automação de um processo, parcial ou completamente, durante a qual documentos, informações ou tarefas são passadas de um participante a outro para a realização de alguma ação de acordo com um conjunto de regras procedurais. Pode-se diferenciar dois estados de um workflow, como notado em [BW95]: estático, sendo uma representação de um processo; e dinâmico, onde um modelo é instanciado e posto em execução sendo alimentado com dados para a obtenção de resultados concretos.

Sistema Gerenciador de Workflows (SGWf). Um *Sistema Gerenciador de Workflows*, ou *SGWf*, é um sistema automatizado para definir e instanciar workflows, gerenciando sua execução. Essa execução pode ocorrer em um ou mais motores de execução, que são capazes de interpretar as definições de um workflow e interagir com os participantes desse workflow (humanos ou sistemas automáticos).

Atividade. Uma *atividade* é uma descrição de uma parte do trabalho a ser realizado dentro de um processo. Uma atividade pode ser básica (ou atômica), representando uma ação indivisível para o SGWf, ou pode ser um sub-fluxo, composto de outras atividades. Além disso, uma atividade pode ser automatizada ou manual (dependente de intervenção humana). Atividades são os elementos básicos da construção de workflows.

Workflow Científico. Um *workflow científico* é a especificação de um processo que descreve um experimento científico [WWVM96]. Algumas características são inerentes a esses tipos de workflows, como: alto grau de flexibilidade, a existência de incertezas e exceções e a possibilidade de serem especificados a partir de experimentos e serem modificados durante sua execução.

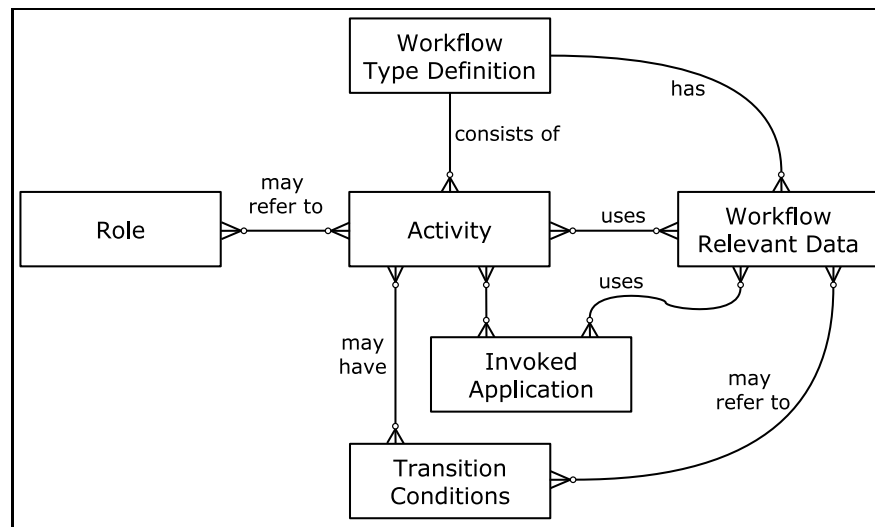


Figura 2.3: Modelo da WfMC para representação de workflows (extraído de [WfM95]).

Alguns desses conceitos se refletem na forma de se representar computacionalmente um workflow (ou um modelo de processo). O modelo de referência da WfMC [WfM95] dá uma visão geral dessa representação, ilustrada na Figura 2.3. Nele, um workflow é composto de atividades e tem dados de controle relacionados. Uma atividade pode fazer referência a um papel e pode fazer uso de dados de controle. Além disso, pode haver condições de transição, relacionando as dependências entre as atividades, e uma atividade pode invocar uma aplicação. Esse modelo conceitual pode servir de base para o desenvolvimento de um modelo de dados para representar workflows.

O uso de workflows para modelar processos científicos, como os mostrados na seção 2.1, ajuda a identificar e esclarecer os passos dos processos e as dependências e fluxos de dados entre esses passos. Pode-se enxergar nas Figuras 2.1 e 2.2 as elipses como atividades e as caixas como dados utilizados nessas atividades. Com isso obtém-se uma documentação para esses processos, fazendo-os auditáveis e reusáveis, por exemplo.

Além de definir um modelo de representação de workflows, a WfMC define também uma arquitetura básica para SGWf. Essa arquitetura, mostrada na Figura 2.4, considera cinco elementos básicos: um serviço de execução (*Enactment Service*) encapsulado por uma API (*Application Programming Interface*) e formatos de intercâmbio (*API and Interchange Formats*), uma ferramenta de definição de processos (*Process Definition Tool*), uma ferramenta de gerenciamento e monitoração (*Administration & Monitoring Tools*), as aplicações clientes (*Workflow Client Applications*) desse serviço, as aplicações invocadas (*Invoked Applications*) por esse serviço e outros serviços de execução. Cada um dos elementos pode estar distribuído em diferentes nós de uma rede. Esse modelo foi utilizado como base para especificar uma nova proposta de arquitetura para o sistema WOODSS

no Capítulo 4.

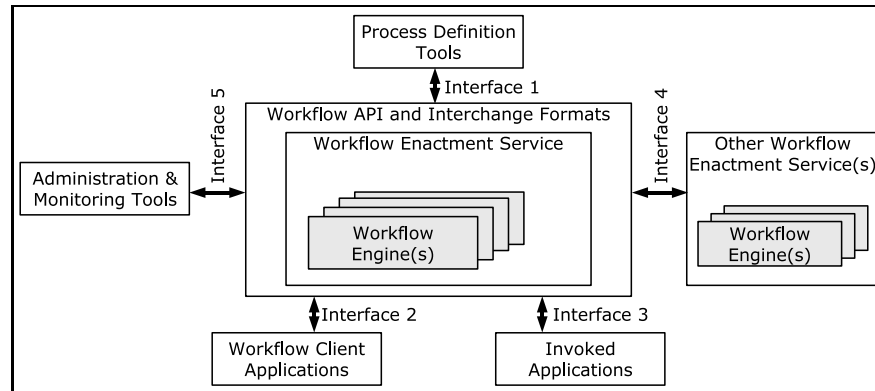


Figura 2.4: Modelo da WfMC de arquitetura de um SGWf (extraído de [WfM95]).

Um SGWf é composto, basicamente, de um serviço de execução, de uma ferramenta de especificação e de uma ferramenta de administração e monitoramento. Os outros elementos podem ser vistos como externos à sua estrutura.

A ferramenta de especificação de workflows permite a um usuário especificar um workflow que posteriormente pode ser transmitido para execução em um serviço de execução. Há dois fatores importantes a serem considerados nessa ferramenta: a interface com o usuário e o formato de transmissão do workflow para o serviço de execução. O formato de transmissão, discutido na seção 2.4, deve permitir expressar todos os tipos de construções e estruturas possíveis em um workflow. Já a interface com o usuário tem sua importância baseada em quão difícil será para o usuário especificar esse workflow. A WfMC não especifica nenhum padrão para a interface com o usuário. Essa interface pode ser feita, por exemplo, através de uma linguagem de scripts, como em [BWM03], ou de uma interface gráfica.

O serviço de execução recebe pedidos de execução através de sua API externa, acompanhados de especificações dos workflows que deve executar. Cada pedido de execução de um workflow instancia um motor de execução (*Workflow Engine*) que passará a cuidar da execução daquela instância de processo. Um motor de execução deve oferecer também funcionalidades de controle sobre a instância de processo para garantir a possibilidade de se gerenciar a execução.

As aplicações cliente de um SGWf também são usuários desse tipo de sistema. O modelo da WfMC está mais focado em usuários humanos, o que faz com que SGWf necessitem de uma interface com usuário. Entretanto workflows também podem ser invocados por sistemas automatizados.

As aplicações invocadas, nas atividades de um workflow, são os elementos que de fato realizam algum tipo de processamento. Essas aplicações podem ser completamente

automatizadas ou requerer a interferência humana. Sua interface de comunicação com o serviço de execução (Interface 3, na Figura 2.4) também é responsável pela comunicação de dados (parâmetros e resultados). Isso ocorre também no caso da Interface 2, com as aplicações cliente. Essas duas interfaces têm uma grande semelhança, compartilhando inclusive uma especificação comum no modelo da WfMC [WfM98].

A interface com outros sistemas de execução permite distribuir a execução de workflows. Com isso, a Interface 4 da Figura 2.4 especifica os protocolos de comunicação entre dois serviços de execução. Vale notar que essa interface deve lidar tanto com dados quanto com especificações de workflows.

A ferramenta de administração e gerenciamento fornece as funcionalidades de gerenciamento de um serviço de execução e de instâncias de workflows em execução. A idéia de se ter uma interface padrão entre essa ferramenta e o serviço de execução é que ferramentas de diferentes fabricantes possam interagir com um determinado serviço de execução.

2.2.2 O Sistema WOODSS

O sistema *WOODSS* (*WO*rkfl*O*w-based *s*patial *D*ecision *S*upport *S*ystem) [Sef98, SMRY99, Kas01, Res03, Roc03, KMR04] é um software desenvolvido no Laboratório de Sistemas de Informação do Instituto de Computação da UNICAMP. Ele foi implementado sobre um SIG (Idrisi [Claa]) comercial e vem sendo testado em diversas atividades de planejamento ambiental, a maioria no contexto de aplicações agrícolas.

O sistema é centrado em capturar dinamicamente a interação do usuário com um SIG, documentando-a através de workflows científicos. Essa ferramenta tem três propósitos durante atividades de planejamento ambiental: (i) documentação de procedimentos de usuários, visando reuso; (ii) suporte à tomada de decisão; e (iii) construção de uma base de modelos que descrevem soluções para problemas de planejamento.

O usuário, interagindo com o SIG, faz uma série de manipulações em mapas. As seqüências de manipulação formam, intuitivamente, procedimentos (ou algoritmos) para a resolução de problemas. Esses procedimentos são representados como workflows e são armazenados em um Sistema Gerenciador de Bancos de Dados (SGBD) relacional. Através do sistema, os usuários podem manipular, combinar e recuperar workflows científicos, favorecendo o reuso e facilitando a criação de novos workflows.

Essa versão do sistema tem uma organização arquitetural como a disposta, em alto nível, na Figura 2.5. A interface com o usuário permite a realização de consultas e atualizações refletidas na manipulação e atualização de elementos gráficos que representem conceitos relacionados a workflows (atividades, transições, etc.) e parâmetros das operações do SIG. Essas interações do usuário com a interface são manipuladas e validadas pela *Gerência* de workflows, que também controla as interações com a interface de

banco de dados, através de consultas, e de aplicação (Idrisi), através de scripts. O funcionamento do sistema se dá em um ambiente *desktop* com utilização de arquivos locais, permitindo interações somente com o SIG Idrisi.

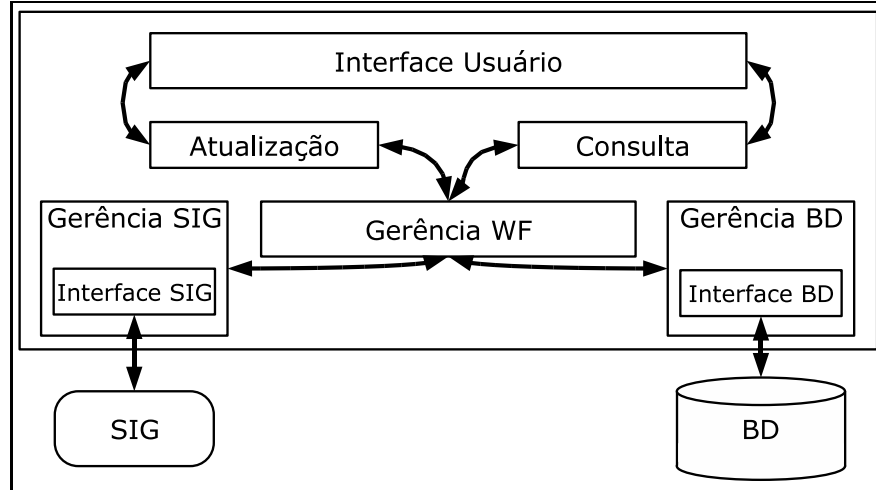


Figura 2.5: Arquitetura do sistema WOODSS (extraída de [Roc03]).

A Figura 2.6 é uma cópia de tela do sistema, após o trabalho desenvolvido em [Res03, Roc03], onde é mostrado um workflow parcial do zoneamento ilustrado na Figura 2.2. As atividades (caixas) representam operações do SIG Idrisi e a transições (setas) ilustram as dependências entre as atividades, podendo ter dados associados, tanto para entrada quanto para saída.

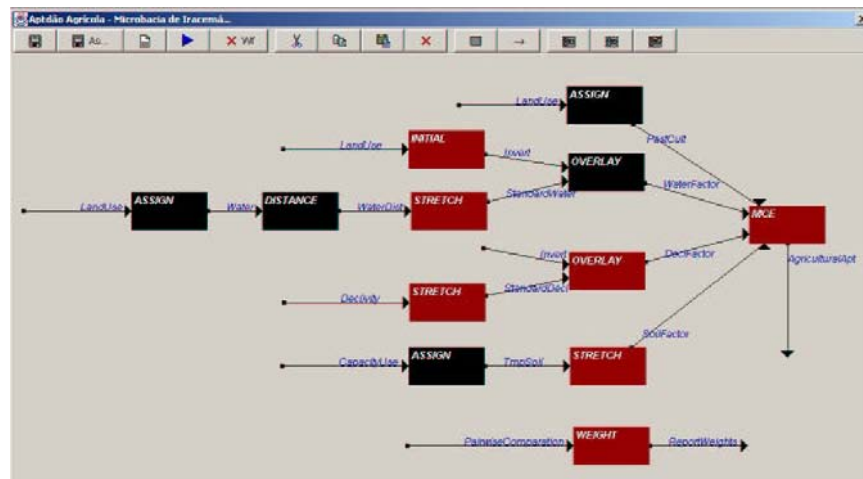


Figura 2.6: Cópia de tela do sistema WOODSS.

2.3 Serviços Web e a Web Semântica

A Web Semântica consiste em um esforço, coordenado pelo World Wide Web Consortium (W3C) [W3Cb], para fornecer semântica apropriada aos dados disponibilizados na Web, permitindo um maior nível de automatização. A evolução da Web para a Web Semântica tem propiciado um ambiente bastante adequado à publicação de dados, principalmente por causa da possibilidade de se acoplar semântica a esses dados através de descrições/metadados.

Entretanto, o uso das técnicas relacionadas à Web Semântica tem se mostrado restrito, principalmente por causa da estágio inicial de desenvolvimento de sistemas para processamento automático de dados publicados. Conseqüentemente, do ponto de vista de implementação, ainda é necessária intervenção humana na publicação e escolha das fontes de dados para alimentar aplicações.

É usual adotar serviços Web [W3Cd, ACKM04] como mecanismo de implementação para a Web Semântica. Serviços Web são softwares que seguem determinados padrões de funcionamento orientado a serviços. Com isso, pode-se enxergar duas dimensões no que diz respeito ao desenvolvimento de padrões para a Web: a descrição de dados, principal preocupação dos esforços envolvendo a Web Semântica até agora; e a descrição de serviços, que guia a implementação dos Serviços Web.

Pode-se relacionar o desenvolvimento da Web Semântica com serviços Web através de uma separação conceitual entre dados e serviços, levando a uma “implementação” para a Web Semântica. De um lado estão os dados que devem ser uniformemente compreendidos quanto à sua semântica. De outro, módulos de software – os serviços Web – que devem prover um grau satisfatório de automação quando lidando com esses dados. A Figura 2.7 estrutura algumas propostas de padrões para descrição de dados e serviços em camadas. As camadas em caixas pontilhadas ainda não possuem um padrão consensual.

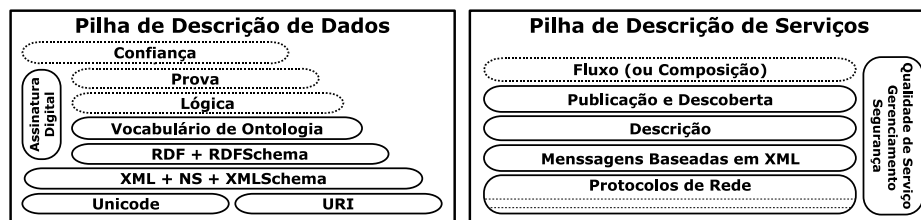


Figura 2.7: Os padrões para a Web Semântica e Serviços Web

Na parte de descrição de dados estão a codificação UNICODE, usada para compatibilidade no processamento de texto, e o conceito de URI (*Uniform Resource Identifier*) para identificar univocamente um recurso abstrato ou físico. Em seguida há a base sintática para a representação de dados de forma semi-estruturada usando XML (*eXtensible Mar-*

kup Language) e o padrão associado, XMLSchema, para espaços de nomes e definições de tipos. A camada de RDF (*Resource Description Framework*) aborda requisitos semânticos, formando a base para processar metadados e expressar relacionamentos. A camada de Vocabulário de Ontologia possibilita o uso de uma linguagem para descrever de forma não ambígua e formal o significado e a terminologia usados em documentos Web. A linguagem OWL (*Web Ontology Language*) é o padrão proposto para essa camada. A camada de Assinatura Digital dá aos dados um certificado, garantindo sua origem, por exemplo. A camada de Lógica estabelece um sistema lógico através do qual a camada de Prova pode realizar inferências sobre os dados representados em camadas inferiores. A Assinatura Digital, combinada com a Prova, asseguraria a validade da informação a ser entregue na camada de Confiança.

A Figura 2.7 mostra seis camadas referentes a serviços. A camada de Mensagens define um protocolo de formatação XML sobre protocolos usuais de redes TCP/IP, oferecendo uma abstração de alto nível para compor e trocar mensagens em um formato compatível com XML. SOAP (*Simple Object Access Protocol*) é o padrão recomendado pela W3C para esta camada. A camada de Descrição de Serviços provê um meio de descrever as funcionalidades de um Serviço Web através da sua interface, sendo que WSDL (*Web Service Description Language*) é o padrão para esta camada. Publicação e Descoberta de serviços, usando UDDI (*Universal Description, Discovery and Integration*) como padrão, fornece um meio de se divulgar e encontrar os Serviços de interesse. Qualidade de Serviço, Segurança e Gerenciamento são questões que devem ser consideradas em todos os níveis. A camada de Fluxo é responsável por coordenar a composição de serviços para se obter uma determinada funcionalidade, utilizando conceitos de workflows para tanto. Essa última camada é de especial interesse para este trabalho pois as técnicas utilizadas baseadas em workflows também tratam de aspectos de representação. Com isso, as propostas de padrão para essa camada são comparáveis às propostas de representação de workflows. Esse aspecto é explorado na seção 2.4.

A descoberta de serviços realizada dinamicamente, por exemplo utilizando o protocolo UDDI, é um grande avanço na execução distribuída de tarefas. Entretanto, pode haver problemas na descoberta caso a semântica associada à descrição desses serviços seja pobre ou ambígua. Uma solução que tem ganhado força ultimamente é a de se descrever serviços de forma mais precisa, à semelhança do que se tem feito com os dados. Indo nessa direção, ontologias e a linguagem OWL-S [OWL] estão sendo introduzidas como um complemento, agregador de semântica, à descrição, publicação, execução, composição e monitoramento de execução de serviços.

2.4 Padrões Para Representação de Workflows na Web

A representação de workflows na Web tem sido alvo de inúmeros esforços, tanto na área acadêmica [vtKB00, vt02] quanto na área industrial [BPMa, Oasb]. Isso vem ocorrendo em duas frentes: interoperabilidade de sistemas e controle de cooperação entre sistemas. A primeira, que é o foco deste trabalho, se preocupa com representar workflows em um formato utilizável na Web (usando XML) motivada pela necessidade de fazer com que sistemas que usam workflows possam se comunicar e funcionar em conjunto. A segunda utiliza técnicas de workflows para realizar o controle de composições de aplicações na Web.

Disso surgiram diversas propostas de padrões para representar workflows em XML. As que têm maior destaque são: (i) XPDL (*XML Process Definition Language*) [WfM02] da WfMC; WS-BPEL (*Web Services Business Process Execution Language*) [Oasa], anteriormente chamada de BPEL4WS (*Business Process Execution Language for Web Services*) [BPE03], de um consórcio de empresas coordenadas pelo OASIS [Oasb]; (iii) BPML (*Business Process Modeling Language*) [BPMb] e sua linguagem de notação visual, BPMN (*Business Process Modeling Notation*), ambas do BPMI.org [BPMa]; (iv) diagramas de atividade da linguagem UML [OMGb], e sua notação visual associada, do consórcio OMG [OMGa]; e (v) WSCI (*Web Service Choreography Interface*) [W3Cc] e WS-CDL (*Web Services Choreography Description Language*) [W3Ce], ambas propostas submetidas ao W3C [W3Cb]. Essas linguagens são apresentadas individualmente a seguir.

XPDL Esta linguagem foi introduzida pela WfMC para servir de meio de intercâmbio para especificações de workflows. Foi inicialmente proposta para representar os dados na Interface 1 (Figura 2.4) que serve para trocar especificações entre a ferramenta de definição e o serviço de execução. Com isso, XPDL visa possibilitar importar/exportar documentos entre SGWf. Entretanto, existem limitações inerentes à própria definição dos propósitos da linguagem. A linguagem busca o “meta-modelo mínimo composto de entidades que descrevem uma definição de um processo de workflow” [WfM02]. Além disso, a proposta especifica que o modelo deve ser completamente livre de requisitos de implementação. Com isso, algumas estruturas não podem ser incluídas na especificação, já que dependem de particularidades de implementação do SGWf, como será visto no Capítulo 3.

WS-BPEL WS-BPEL visa prover uma linguagem para expressar especificações de funcionamento de serviços em uma composição. Antes de ser passada ao controle do consórcio OASIS, esta linguagem tinha o nome de BPEL4WS. Origina-se da combinação de duas outras languages: WSFL (*Web Services Flow Language*), proposta pela IBM e XLANG (*Web Services for Business Process Design*), proposta pela Microsoft. A primeira é ba-

seada em grafos acíclicos, enquanto a segunda é estruturada em blocos, o que faz da combinação uma linguagem híbrida. Composições construídas nesta linguagem especificam, na verdade, workflows, o que torna a linguagem potencialmente adequada para os propósitos deste trabalho, como será mostrado no Capítulo 3.

BPML e BPMN Estas linguagens são propostas do BPML.org (*Business Process Management Initiative*) para expressar processos e as entidades que os suportam. BPML contempla a representação em XML enquanto que BPMN trata da notação gráfica. A proposta de representação em XML não foi considerada na dissertação por ser similar a WS-BPEL em termos de poder de representação e estruturas utilizadas para representar workflows. Entretanto a notação gráfica proposta com BPMN, é avaliada no Capítulo 3.

Diagramas de Atividades UML UML foi introduzida como linguagem para modelagem de sistemas (computacionais ou não) e é composta de diversos tipos de diagramas com uma semântica bem definida. Em particular, os diagramas de atividades da linguagem UML são adequados para definir seqüências e condições para coordenar comportamentos [OMGc]. Esses diagramas exprimem conceitos estreitamente relacionados com workflow, principalmente quando considerada a proposta da versão 2.0 da linguagem [OMGb]. Não só esses diagramas podem ser expressos graficamente, como existe também uma representação para eles em XML, chamada de XMI (*XML Metadata Interchange*). Com isso, os diagramas de atividades UML são capazes de representar workflows graficamente e em XML. Entretanto, a representação tem o foco em aspectos de modelagem de processos, deixando de lado questões de execução e instanciação de processos. Isso faz com que a linguagem não seja adequada para este trabalho, do ponto de vista de representação em XML. Mesmo assim, sua representação gráfica continua passível de uso, como será visto no Capítulo 3.

WSOI e WS-CDL Estas duas linguagens não suportam a representação de workflows explicitamente, mas servem para a definição de comportamentos individuais dos elementos participantes de um processo. Com isso pode-se inferir o workflow que modela essas especificações de comportamento. Entretanto, esse enfoque não é adequado para as necessidades de representação de processos como documento. Com isso, essa proposta não foi considerada como uma das alternativas para este trabalho.

2.5 Reuso e Workflows

Reuso pode ser definido como a utilização de artefatos existentes para construir novos artefatos fazendo uso, total ou parcialmente, do objeto original [SM04b, Roc00, GMP96].

Fazendo um paralelo com linguagens de programação, o reuso pode ser realizado “por valor” (cópia do objeto a ser reusado) ou “por referência” (via ligação ao objeto) [GMP96]. No primeiro caso, um objeto reusado fica integrado ao novo objeto, perdendo a conexão com a fonte original. No segundo, a conexão é conservada, através de uma ligação, mantendo com isso uma referência a um elemento adequado para uma determinada função. Isso faz com que a atualização desse elemento na fonte original seja propagada, de forma transparente (com encapsulamento adequado), aos outros elementos que o referenciam. Por outro lado, este último tipo de reuso pode ser mais custoso por exigir recuperar, a partir das ligações, todos os objetos reusados.

Vantagens associadas ao reuso incluem, por exemplo: (i) o aumento da consistência, já que um elemento é especificado uma vez e reusado muitas, é garantido que ele tem suas funcionalidades consistentes com o que foi originalmente proposto, aumentando a qualidade da composição; (ii) redução dos custos de desenvolvimento e manutenção, por concentrar os esforços em uma instância de desenvolvimento, ou manutenção, replicando-os quase sem custos; e (iii) reconfiguração rápida, já que com o uso da modularidade obtida com os componentes reusáveis pode-se obter novas organizações de forma mais ágil.

A ênfase desta dissertação é em buscar um modelo de workflows que garanta reuso de especificações não apenas localmente como também de forma distribuída, fazendo uso de referências dinâmicas remotas (através de URIs, por exemplo). A idéia de se gerenciar e disponibilizar workflows via Web visa permitir seu reuso em um ambiente distribuído. O reuso de modelos de processos especificados como workflows se beneficia de todas as vantagens listadas. Segundo [SM04b, SM04a] pesquisas de reuso de artefatos digitais se concentram na área de engenharia de software para processos e na área de engenharia de conteúdo para dados. Um dos poucos trabalhos que transporta noções de reuso para workflows é [BWM03]. Nele uma linguagem de especificação de workflows é definida com o objetivo de suportar uma *biblioteca* de workflows que podem ser reusados como módulos.

2.6 Conclusões

Este capítulo apresentou os principais conceitos usados como base neste trabalho. Primeiramente foram apresentadas as necessidades e dificuldades na realização de experimentos científicos, do ponto de vista de documentação, usando planejamento ambiental como pano de fundo. Em seguida o capítulo apresentou sistemas de workflows, mostrando como podem ser aplicados na resolução dos problemas apresentados. Os conceitos relacionados à Web e sua evolução foram apresentados logo após, esclarecendo as necessidades para um sistema funcionar integrado a esse ambiente. O capítulo discutiu a necessidade de se especificar workflows tanto do ponto de vista de linguagens voltadas à Web (*à la XML*)

quanto de representação gráfica. Por fim foram apresentadas as propostas de padrão para ambas as necessidades. Mais detalhes deste estudo comparativo pode ser encontrado em [PMRR05].

O Capítulo 3 trata da representação de workflows, incluindo uma proposta de modelo de dados que estende o modelo do WOODSS. O modelo comporta a representação para armazenamento em um SGBD da especificação de um workflow, de forma a permitir seu uso na Web. Inclui igualmente a proposta de uma representação gráfica.

Capítulo 3

Modelo Proposto para Representação de Workflows

Este capítulo trata da representação de workflows em sistemas computacionais. A seção 3.1 apresenta o modelo de dados proposto na dissertação, que é uma das contribuições da pesquisa. O modelo permite armazenar em um único repositório especificações de workflows e seus componentes, em diferentes níveis de abstração. Além disso, o modelo induz uma metodologia de especificação e documentação de workflows, outra contribuição deste trabalho. A seção 3.2 apresenta em detalhe os dois modelos que se mostraram mais adequados para representar workflows em XML a partir dos conceitos introduzidos no capítulo 2.

Em seguida, a seção 3.3 discute questões relacionadas à representação visual de *workflows*. Isso é relevante para este trabalho, para definir uma ferramenta gráfica de edição de *workflows* que funcione integrada a um ambiente Web. Por fim, a seção 3.4 finaliza o capítulo com uma discussão sobre as propostas adotadas.

3.1 O Novo Modelo de Dados para Workflows no WOODSS

Esta seção traz os modelos de representação de *workflows* utilizados na implementação anterior do sistema WOODSS (tratada aqui por modelo original ou antigo) e a proposta deste trabalho (novo modelo), usando modelagem Entidade-Relacionamento Estendida. Essa forma de representação foi usada pela simplicidade e poder de sintetização conceituais.

Vale notar que nesta seção serão abordadas principalmente questões de modelagem de dados e construção de especificações de workflows, sendo que os aspectos de execução

relevantes ao modelo serão discutidos em separado no final da seção.

3.1.1 O modelo de dados original do WOODSS

A Figura 3.1 mostra o modelo de dados original do WOODSS. O elemento central do modelo é a entidade *Activity*. A partir das atividades é que se expressam os demais elementos.

No modelo, atividades podem ser atômicas ou sub-fluxos, representados respectivamente pelas entidades *AtomicActivity* e *SubWorkflow*, que são especializações de *Activity*. Um workflow, materializado na entidade *Workflow*, possui um conjunto de zero ou mais atividades. Uma atividade do tipo sub-fluxo referencia um workflow que deve ser executado integralmente.

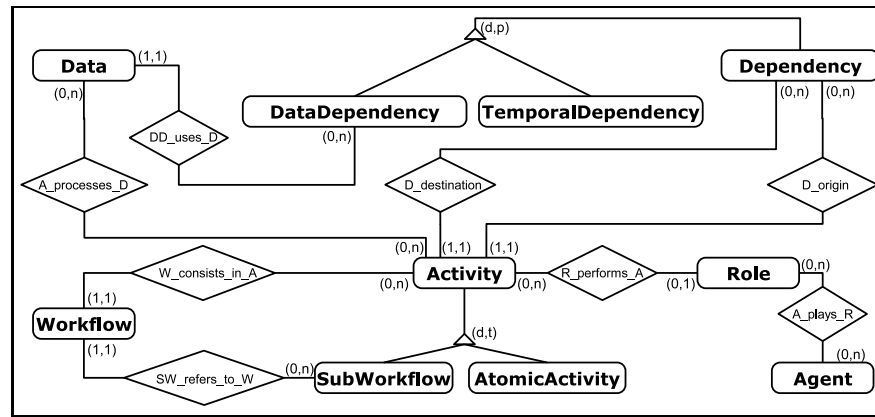


Figura 3.1: Modelo de Dados Original

As interdependências entre as atividades são representadas como elementos da entidade *Dependency*. Uma dependência tem uma origem e um destino, ambos em atividades, representados pelos relacionamentos *D_origin* e *D_destination*, sendo possíveis múltiplas instâncias de dependências em uma mesma atividade de origem e/ou destino, respectivamente. Além disso, uma dependência pode ser temporal (*TemporalDependency*) ou dependência de dados (*DataDependency*), não simultaneamente. Uma dependência temporal diz respeito à organização da precedência entre atividades. Uma dependência de dados mostra que uma atividade depende de dados gerados por uma atividade anterior. Uma dependência de dados é sempre uma dependência temporal, mesmo sendo tratadas como disjuntas neste modelo. Esse é um dos problemas eliminados no novo modelo proposto neste trabalho, como será visto.

O diagrama mostra que uma dependência de dados “leva” dados de uma atividade a outra (*DD_uses_D*) e que uma atividade processa dados (*A_processes_D*). Isso causa

duplicação de ocorrências em relacionamentos que ligam dados a atividades, aspecto corrigido no novo modelo.

Esse modelo apresenta ainda limitações quanto à capacidade de representação, como por exemplo, de sub-fluxos, atividades estruturadas e ramificações. Sub-fluxos não podem ser representados adequadamente por não haver uma forma, dentro do modelo, de se incluir em um workflow um sub-fluxo que é referenciado em outros workflows, sem quebrar o seu encapsulamento. O modelo original comporta apenas as estruturas de seqüência, ramificação *AND* (ou paralela) e junção *AND* (ou síncrona). Outras estruturas, como repetição do tipo *for* ou *while* não são consideradas.

O novo modelo resolve o problema de encapsulamento, aborda diferentes estruturas e inclui a possibilidade de se construir ciclos arbitrários como uma estrutura semântica mais abrangente que um *while*. Uma discussão sobre tipos de estruturas pode ser encontrada em [WP, vtKB00, vt02].

3.1.2 O modelo de dados proposto

Visão geral e conceitos básicos do modelo

A principal característica do novo modelo é de efetivamente permitir armazenar em um banco de dados diferentes níveis de abstração de um workflow. A idéia básica é que atividades (e portanto workflows) podem ser descritas em diferentes níveis de abstração. Assim, a noção de repositório de workflows que podem ser reusados e compostos é estendida para a noção de *repositório de especificações de componentes de workflow*.

Este repositório contém as especificações abstratas de dados e atividades, além de workflows especificados em diferentes níveis de detalhe. Elementos desse repositório podem ser refinados e especializados. Isso induz naturalmente regras adequadas de construção de workflows e sua documentação. A Figura 3.2 ilustra esse processo. O primeiro nível, de especificação abstrata de componentes, Figura 3.2(a), descreve em alto nível de abstração componentes que podem ser usados em um workflow, i.e., as atividades, seus elementos de entrada e saída e o seu comportamento esperado. As noções de *tipo de atividade* e *tipo de dados* são centrais neste nível de especificação. Assim, existe uma atividade de tipo *Overlay*, com entradas *E1* e *E2* e saída *S1*. Entradas e saídas também são abstratamente especificadas nesse nível, através de informações sobre seu tipo.

O segundo nível, Figura 3.2(b), já contempla a especificação de workflows propriamente ditos, através da combinação dos elementos previamente especificados no nível anterior. Neste caso, saídas e entradas de atividades são conectadas por *transições* – por exemplo, a transição *T1* liga a saída *S1* de *Overlay1* à entrada *I1* de *Impressão*. Um conceito associado é o de *conector*, que são pontos em cada atividade aos quais se conectam transições ou conjuntos de dados. Pensando em termos de orientação a objetos,

conectores compõem a única parte visível das interfaces de uma atividade e correspondem às ligações para entradas e saídas. Os workflows do nível (b), entretanto, não estão completamente instanciados: falta especificar, por exemplo, fontes de dados, e indicar quais agentes deverão fazer a execução de cada atividade.

Já no terceiro nível, Figura 3.2(c), um workflow já tem especificadas suas ligações com dados (por exemplo, a entrada *E1* é ligada a um arquivo *mapa1*), e oferece meios de determinar quais agentes devem ser invocados (através de chamadas específicas com destinatários já determinados, por exemplo). Nesse nível, um workflow é dito ser *executável*, pois já tem todos os elementos necessários para sua execução por um SGWf.

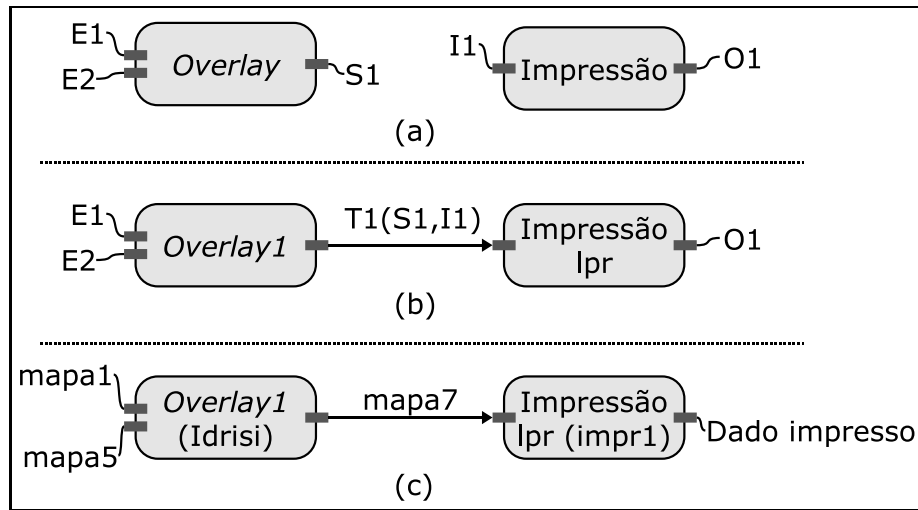


Figura 3.2: Níveis de abstração na especificação de um workflow

Uma especificação de um **workflow** segue, dessa forma, uma seqüência de passos. Primeiramente, são cadastrados os **tipos de dados** que serão usados. Em seguida, são cadastrados os **tipos de atividades** que podem fazer parte desse **workflow**, especificando seu comportamento e seu conjunto de entradas e saídas (**conectores de atividade**), agregando os **tipos de dados** já documentados. Os **papéis** que podem ser cumpridos por uma **atividade**, bem como os **agentes** capazes de executá-la, também devem estar disponíveis para se especificar um **workflow**. O segundo passo especifica o **workflow** através das transições adequadas entre atividades (dependências no modelo anterior). As **atividades** incluídas podem ser **básicas** (atômicas no modelo antigo) ou **sub-fluxos**. Uma **atividade básica** é uma unidade mínima de processamento. Já um **sub-fluxo** encapsula um **workflow** de forma transparente. O terceiro passo cria um **workflow** executável. Esta criação pode ser estática (antes da execução) ou dinâmica – o workflow é construído durante sua execução. O modelo, dessa forma, permite gerenciar “componentes” abstratos para construção de workflows, padrões de workflows e workflows

executáveis.

Os elementos em destaque no parágrafo anterior determinam as entidades básicas que compõem o novo modelo de dados. Existem ainda outras três entidades não listadas (**connector de workflow**, **especificador de parâmetro** e **instância de atividade**), duas das quais geradas por agregações, que serão explicadas, a seguir, na descrição detalhada do modelo.

Descrição do modelo

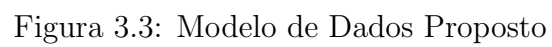
O modelo proposto estende o modelo antigo, buscando manter compatibilidade. Entretanto, alguns conceitos têm semântica diversa, como, por exemplo, é o caso dos conceitos de transição (chamado de dependência no modelo antigo) e atividade, cujo escopo era mais amplo.

As extensões visam corrigir os problemas apresentados na seção 3.1.1. Além disso, buscam permitir referenciar sistemas genéricos, enquanto que o modelo anterior contemplava apenas o SIG IDRISI [Clab]. Outro aspecto para o qual se buscou um tratamento mais generalizado foram as fontes de dados. Para isso, foi necessário introduzir o conceito de tipo de dados. Vale notar ainda que o modelo atual é compatível com o modelo de referência proposto pelo WfMC [WfM95]. As idéias centrais são apresentadas a seguir.

A Figura 3.3 mostra o diagrama ER estendido do modelo proposto, que mantém as entidades básicas *Activity*, *Workflow*, *SubWorkflow*, *Role*, *Agent* e *Data*. As entidades *AtomicActivity* e *Dependency* tiveram seus nomes mudados para *BasicActivity* e *Transition*, respectivamente, para se adequarem à nomenclatura corrente [WfM95]. As especializações de dependência, *DataDependency* e *TemporalDependency* do modelo antigo foram removidas. As entidades *DataType*, *ActivityType* e *WorkflowConnector* são novas, bem como as entidades geradas pelas agregações *ParameterSpecifier*, *ActivityInstance* e *ActivityConnector*.

A apresentação do modelo seguirá a ordem de construção de um workflow apresentada na descrição geral do modelo. Fica subentendido que toda entidade simples tem um identificador numérico único que será sua chave.

Um tipo de dados (*DataType*) especifica um tipo para caracterizar um dado. Essa caracterização é feita pela especificação do par <tipo, aplicação> onde ‘aplicação’ é o nome de uma aplicação capaz de tratar o dado descrito e ‘tipo’ é o tipo do dado. Exemplos de tipos de dados são: “inteiro de 32bits” (identificador do tipo) tratado pelo “somador de inteiros” (aplicação) e “mapa vetorial de relevo” (tipo) tratado pelo “SIG IDRISI”. Deve-se seguir um padrão de formação de nomes, para facilitar a caracterização de hierarquia de “tipos” e “aplicações”: usar um ponto, nos campos, para separar elementos de uma mesma hierarquia ficando mais específico da direita para a esquerda. Um exemplo desse padrão é uma hierarquia de para numerais na qual um inteiro de 32 bits não negativo



teria seu tipo representado pelo *string* “inteiro.32bits.nãoNegativo” e um mapa vetorial de relevo do Brasil originaria o *string* “mapa.vetorial.relevo.brasil”. Isso permite, por exemplo, utilizar caminhos *IS_A* em ontologias para caracterizar os tipos de dados e de operações.

Fontes de dados (*Data*) são especificados a partir de seu tipo. Um atributo da entidade *Data* guarda um ponteiro de acesso a fontes de dados armazenados em forma de URI (que pode ser também um caminho em um sistema de arquivos local). Fontes de dados podem ser vistas como sendo de entrada ou saída. No primeiro caso, o dado é fornecido à atividade para realizar o processamento; e no segundo, o dado originado pela atividade é armazenado no destino.

A entidade *ActivityType* determina o tipo de atividade, tendo como atributo um par <operação,aplicação> que designa uma operação passível de ser executada por uma aplicação. Por exemplo, o par <overlay,IDRISI> especifica que a atividade corresponde a uma operação de *overlay* nesse SIG.

O relacionamento *AT_has_io_DT* determina o conjunto de tipos de dados que formam as entradas e saídas de um tipo de atividade. Atributos do relacionamento determinam se o dado é de entrada ou saída e qual a sua ordem de ocorrência no tipo de atividade (para operações não comutativas, por exemplo). A agregação *ParameterSpecifier*, que encapsula esse mesmo relacionamento, é assim um conjunto de tuplas <tipo de dado, tipo de atividade, ordem, E/S> identificando univocamente cada um dos parâmetros de um tipo de atividade. Através dessa agregação é possível fixar um determinado dado como entrada ou saída de uma atividade usando o relacionamento *PS_has_D*. O atributo ‘ordem’ determina a precedência entre os parâmetros, por exemplo, o parâmetro *E1* é o primeiro (‘ordem’ 1) da atividade de *overlay* e o parâmetro *E2* é o segundo (‘ordem’ 2). O atributo serve, também, tanto para representação visual quanto para diferenciar parâmetros de mesmo tipo em um mesmo tipo de atividade.

Um workflow (*Workflow*) é composto por um conjunto de atividades (*Activity*) e transições (*Transition*) entre conectores dessas atividades e tem como atributos seu nome, uma descrição, data de criação e seu estado de completude. Este último atributo determina se o workflow tem todas as entradas e saídas de dados especificadas, as transições corretamente conectadas e os agentes para execução determinados. Em caso positivo, ele pode ser executado. Isso facilita a recuperação de especificações executáveis para disponibilizar ao usuário. Além disso, todo workflow corresponde a uma atividade (visto de forma abstrata). Dessa forma, precisa ser ligado (*W_has_AT*) ao tipo de atividade correspondente, para poder fazer parte de um outro workflow como sub-fluxo.

A entidade *Role* define um papel que pode ser responsável por uma atividade dentro de um processo. Pode-se ligar um, ou mais, papéis a um tipo de atividade (relacionamento *AT_may_have_R*). Um agente (*Agent*) pode cumprir um determinado papel e executar uma

determinada atividade de tipo compatível com o papel que esse agente está se propondo a cumprir.

Uma atividade é representada pela entidade *Activity* e tem um tipo determinado pela entidade *ActivityType*. Os atributos de uma atividade são seu nome, descrição, uma pré-condição à sua execução, e uma pós-condição que deve ser válida ao seu término, e modos de controle de entrada e saída de atividade. Pré- e pós-condições são expressões booleanas envolvendo os dados da atividade e o estado do workflow. Se sua pré-condição falha, a atividade não é executada e todas as suas transições de saída são avaliadas como falsas; caso avaliada como verdadeira, a atividade pode ser executada normalmente. Se a pós-condição é avaliada como falsa, todas as transições de saída são avaliadas como falsas; caso verdadeira, o fluxo continua normalmente. No caso de todas as transições de saída serem avaliadas como falsas, o fluxo é interrompido no ponto em que aquela atividade se localiza, sem implicações em fluxos paralelos. Os modos de controle serão explicados após a discussão sobre transições, já que são conceitos relacionados.

Uma atividade é especializada de forma disjunta e total em *WorkflowConnector*, *SubWorkflow* e *BasicActivity*. Uma atividade que é um conector de workflow está relacionada com a interface de um workflow e será explicada posteriormente. Um subworkflow, ou sub-fluxo, encapsula um outro workflow (relacionamento *SW_expands_to_W*). Já uma atividade básica representa uma operação atômica (ao sistema) a ser realizada.

Uma instância de atividade (*ActivityInstance*) corresponde à agregação que liga atividade a seu tipo. A separação entre *ActivityType*, *Activity* e *ActivityInstance* permite especificar atividades (e workflows) em vários níveis de abstração. Isso não seria possível caso houvesse apenas uma entidade *Activity*, como no modelo anterior.

Um conector de atividade (*ActivityConnector*) liga uma instância de atividade a um parâmetro (*ParameterSpecifier*). Isso possibilita discriminar com exatidão cada ponto de entrada e saída de uma atividade. Um conector de atividade tem ainda um atributo que é o modo de controle de conector. Esse atributo tem relação com os modos de controle de atividade e também será explicado após a discussão sobre transições.

Na Figura 3.4 os elementos *X*, *Y*, *Z* e *N* são os conectores de entrada e *K*, *L* e *M* são de saída da Atividade *A*. O conector de atividade é responsável por fazer a ligação de atividades a dados (relacionamento *AC_has_D*) ou a transições (*T_destination* e *T_origin*). Ressalte-se que um mesmo conector pode receber entradas de (ou gerar saídas para) dados e transições ao mesmo tempo (conector *Y* da Figura 3.4(b)). Esse ponto será melhor esclarecido juntamente com os aspectos de execução do modelo. Vale ainda ressaltar algumas características de conectores, usando o conector *Y* como exemplo. O tipo de dados associado a este conector precisa ser o mesmo dos dados *Mapa T* e *Mapa U*. *Mapa T* é um conjunto de dados unido diretamente de alguma fonte de dados, enquanto *Mapa U* é um conjunto de dados que chega à Atividade *A* a partir de alguma outra atividade.

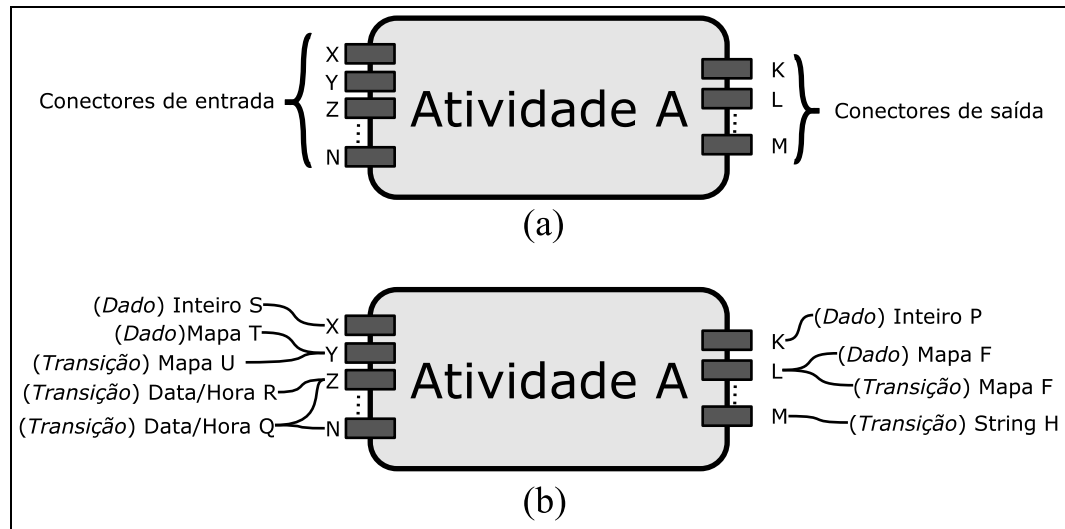


Figura 3.4: Ilustração do conceito de conectores e seu comportamento

A semântica da entidade *Transition* é independente dos dados sendo transportados de uma atividade à seguinte. Além disso, toda transição é considerada como temporal, ou de controle, já que estabelece relação de precedência entre atividades quer elas troquem dados ou não.

Os atributos de uma transição são seu nome, sua condição de ativação e a validade de sua ativação. A condição de ativação é uma pré-condição para ativar a atividade seguinte, respeitando as premissas e limitações impostas pelos modos de controle. A validade determina o número de vezes que a transição deve ser habilitada, i.e., ativar a atividade destino. A necessidade desse atributo é discutida juntamente com as questões de execução.

Os modos de controle de atividade podem ser de entrada e saída. Estes atributos permitem definir, em tempo de execução, como interpretar ramificações de transições de entrada (*AND*, *OR*, *XOR*) e saída (*AND*, *OR* e *XOR*).

O modo de controle de saída de atividade determina o comportamento de uma ramificação, i.e., quando uma saída serve de entrada a múltiplas atividades. O atributo indica o tipo de operação lógica a ser usada na ramificação – *AND*, *OR* ou *XOR*: *AND* indica que todas ramificações devem ser ativadas, independentemente de suas condições de ativação; *OR* que devem ser ativadas somente as transições cujas condições de ativação sejam avaliadas como verdadeiro; e *XOR* que qualquer uma das transições com condições de ativação verdadeiras deve ser tomada, mas apenas uma.

O modo de controle de entrada de atividade define, para cada atividade que tem mais de um conector de entrada, como aceitar transições de forma semelhante ao modo de saída: *AND* (sincroniza todas as ramificações), *OR* (sincroniza somente as ramificações

ativas) e XOR (recebe apenas uma ramificação – assume-se, neste caso, que somente uma ramificação estará ativa). Os modos de controle de saída OR e XOR exigem, em tempo de execução, um controle das ramificações que foram ativadas para saber quais (ou quantas) devem ser esperadas.

O atributo modo de controle de conector, de *ActivityConnector*, serve para definir como aceitar várias transições com destino em um mesmo conector: todas, cada uma causando uma ativação separada da atividade (*MULTIPLE*); ou apenas uma, descartando as demais (*DISCRIMINATOR*).

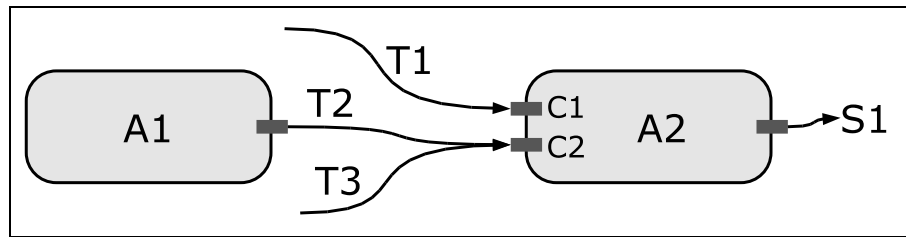


Figura 3.5: Ilustração do conceito atributos de modos de controle

A Figura 3.5 traz um exemplo para ilustrar o uso desses atributos. O atributo de controle de conector para *C2* vai dizer, por exemplo, se *A2* deve aceitar apenas uma transição (*T2* ou *T3*, exclusivamente – discriminador) ou ambas, na ordem em que ocorrerem (múltiplo). Um exemplo de controle de conector múltiplo é o caso de ciclos (veja explicação da Figura 3.6). Já o valor discriminador, pode ocorrer, por exemplo, na admissão de uma mesma entrada advinda de várias fontes, onde a primeira a fornecer o dado é considerada e as demais descartadas. O atributo de controle de entrada de atividade referencia todos os conectores – no caso, indica se ambos *C1* e *C2* aceitam transições ou apenas um deles. O atributo de controle de entrada de atividade tem precedência na determinação dos conectores que aceitam transições, i.e., se um controle de atividade determina que um conector não deve aceitar transições, todas as transições que chegam nele serão desconsideradas, independente do controle de conector.

O conceito de conector aplicado a um workflow, representado pela entidade *WorkflowConnector*, permite encapsular corretamente um workflow como sub-fluxo de outros workflows. Essa entidade serve como uma abstração análoga ao conceito de conector para atividade, i.e., para estabelecer uma interface independente do que é representado internamente em um workflow. As instâncias dessa entidade são atividades “vazias” que são usadas somente para estabelecer os elementos de interface de um workflow. Um conector de workflow tem ainda um atributo extra, seu tipo, que determina se a atividade é inicial (início de processo) ou final (finalização de fluxo). As atividades que não são conectores de workflow são consideradas regulares, ou seja, representam pontos intermediários (executáveis) do processo.

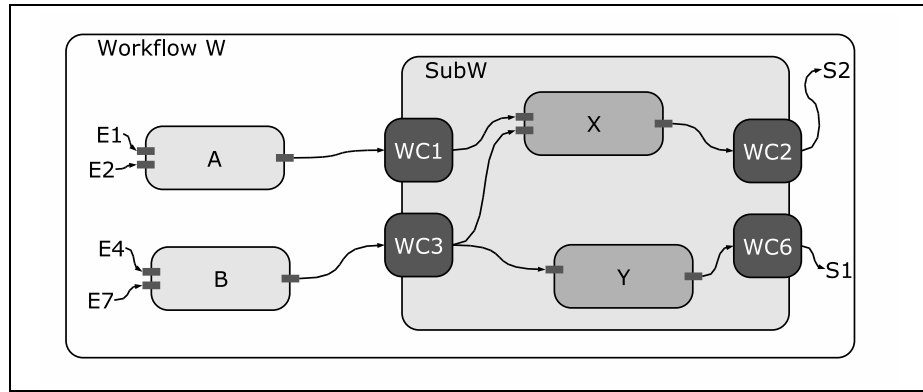


Figura 3.6: Funcionamento de elementos de *WorkflowConnector*

A Figura 3.6 ilustra a necessidade desse elemento através de um exemplo. Considere um workflow W , com atividades A , B e $SubW$ (um sub-workflow), com seus conectores de entrada e saída. Os conectores de $SubW$ são ao mesmo tempo conectores de atividade (a atividade $SubW$ do workflow W) e conectores de workflow (do sub-fluxo $SubW$). Para cada conector existe uma atividade ($WC1$, $WC2$, $WC3$ e $WC6$) em $SubW$. A transição entre as atividades A e $SubW$ é na verdade modelada como uma transição entre as atividades A (regular) e $WC1$ (vazia). Essas atividades vazias são elementos de *WorkflowConnector*. Internamente, as atividades X e Y do workflow $SubW$ são ligadas a essas atividades vazias para permitir expressar as possíveis conexões indiretas com workflows externos. Dessa forma, uma modificação no workflow $SubW$ não afeta os outros workflows que o utilizam como sub-fluxo, já que a interface sempre permanece inalterada.

Este novo modelo possibilita expressar diversas novas estruturas para um workflow. Além disso, possibilita a inclusão, na execução do workflow, de seleção dinâmica de dados e agentes, usando descrições que associem semântica a esses elementos.

Aspectos de execução de workflows

Alguns aspectos do modelo de dados apresentados somente podem ser entendidos quando considerado o tempo de execução de um workflow por um SGWf. Esses aspectos são discutidos a seguir.

Uma transição habilitada, i.e., que tenha sua condição de ativação verdadeira, estabelece sempre a transferência do controle de fluxo para a próxima atividade, independentemente de corresponder a fluxo de dados ou não. Com isso, uma transição sempre determina uma ordem de precedência entre as atividades.

No início da execução de um workflow, todas as condições de ativação das transições devem ter valor falso, sendo reavaliadas durante a execução. Isso permite a construção e controle de ciclos (transições que têm seu destino em atividades que já foram executadas

anteriormente na mesma instância do workflow).

A Figura 3.7 ilustra os casos possíveis para a execução de ciclos. A atividade *A1* tem dois conectores de entrada e está posicionada antes da atividade *A2* no workflow. No conector superior, existem duas transições chegando: *T1* e *T9*. Note que os conectores *C1* e *S1* precisam ser do mesmo tipo. Na primeira vez que *A1* for executada, *A2* ainda não terá sido executada, e a condição de ativação de *T9* ainda terá seu valor inicial falso. Quando *A2* for executada e ativar *T9*, o que se espera é que: (i) *A1* seja executada novamente com a mesma entrada (de *T2*) no conector inferior; ou (ii) *A1* espere pela reativação de *T2*. No primeiro caso, usa-se o atributo validade da transição *T2* para deixá-la ativa por um determinado número de ativações de *A1*. No segundo, deixa-se a validade de *T2* com o valor zero, fazendo com que a condição de ativação de *T2* assuma novamente o valor falso.

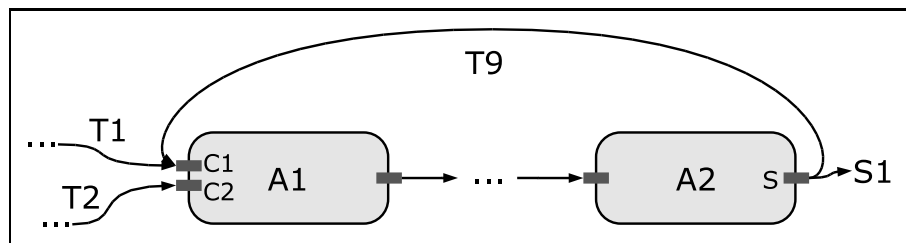


Figura 3.7: Funcionamento do atributo validade aplicado a ciclos

Existem ainda os casos de compartilhamento de conector por transições e dados. Um conector de entrada pode ser compartilhado por várias transições ou um dado e uma ou mais transições. Note que não se pode conectar mais de um dado em um conector de entrada. O primeiro caso, somente transições, já foi tratado na descrição do modo de controle de conector. Já quando um dado e uma conexão compartilham o mesmo conector de entrada, o comportamento fica a cargo da aplicação executando o workflow. A interpretação desse caso deve ser de sempre dar prioridade às transições, i.e., o dado somente é fornecido como entrada se todas as outras transições que compartilham o conector de entrada forem avaliadas como falsas; nesse caso o dado é sempre fornecido. Em outras palavras, o dado é sempre “avaliado” como verdadeiro, mas sempre é a última opção.

Para conectores de saída, a interpretação é mais simples. Pode-se ligar quantas transições e dados forem desejados, sem restrições de combinação. Os dados não interferem no fluxo de execução e, caso a atividade execute corretamente, todos os dados ligados a um mesmo conector de saída são atualizados com o mesmo valor. As eventuais transições devem seguir seu comportamento normal, como se nenhum dado estivesse conectado.

Vale notar ainda que atividades que sejam um conector de workflow (*WorkflowConnector*) devem ter somente uma entrada e somente uma saída (ambas do mesmo tipo).

Uma última restrição é que um elemento da entidade *ParameterSpecifier* que seja parte de um relacionamento com um dado, através de *PS_has_D*, impede que o mesmo elemento faça parte do relacionamento *AC_has_D* com a entidade *Data*, já que esse elemento já tem sua ligação com um dado fixada, fazendo parte da definição do tipo de atividade.

3.2 Representação de Workflows em XML

Esta seção discute a representação de workflows em XML. Em particular são apresentadas as duas propostas de padrão de representação de maior abrangência tanto técnica quanto do ponto de vista de suporte de usuários, a saber XPDL e WS-BPEL. Ambas já foram brevemente apresentadas no Capítulo 2, e serão discutidas em detalhe nesta seção, comparando-as com o modelo proposto neste trabalho.

3.2.1 XPDL

XPDL (XML Process Definition Language) [WfM02] é uma linguagem baseada em XML, com definição em XML Schema, para representar workflows estaticamente. Seu propósito é servir como uma linguagem para troca de especificações de workflows entre parceiros. A linguagem utiliza conceitos do modelo de referência da WfMC [WfM95], que especifica uma arquitetura básica para o funcionamento de um SGWf. A seguir serão discutidos os principais elementos que compõem a linguagem e sua correspondência com o modelo proposto (seção 3.1.2).

A Figura 3.8 é um diagrama de classes UML mostrando os conceitos de XPDL mais relevantes para este trabalho. A figura foi construída a partir do documento em XML Schema que define a linguagem [WfMb]. Os termos classe e entidade serão utilizados de forma intercambiável nesta seção, facilitando a compreensão dos conceitos. Alguns conceitos da linguagem foram agrupados para estabelecer relacionamento com outros elementos, como é o caso de *FormalParameter* e *FormalParameters* e *DataField* e *DataFields*. Essa estratégia, contudo, não interfere na discussão dos conceitos.

A classe *DataTypes* forma um conjunto de tipos de dados que podem ser usados para descrever os dados usados em um workflow. Um tipo pode ser básico, declarado ou externo. Os tipos básicos são tratados na própria linguagem, embora a definição não especifique a forma de implementação desses tipos. Um tipo declarado (*DeclaredType*) possibilita que o usuário crie novos tipos na linguagem. Por fim, um tipo externo faz referência a uma definição externa ao documento de especificação de workflows. A entidade *DataType* do modelo proposto contempla todas essas possibilidades, permitindo também definir a forma de representação do tipo.

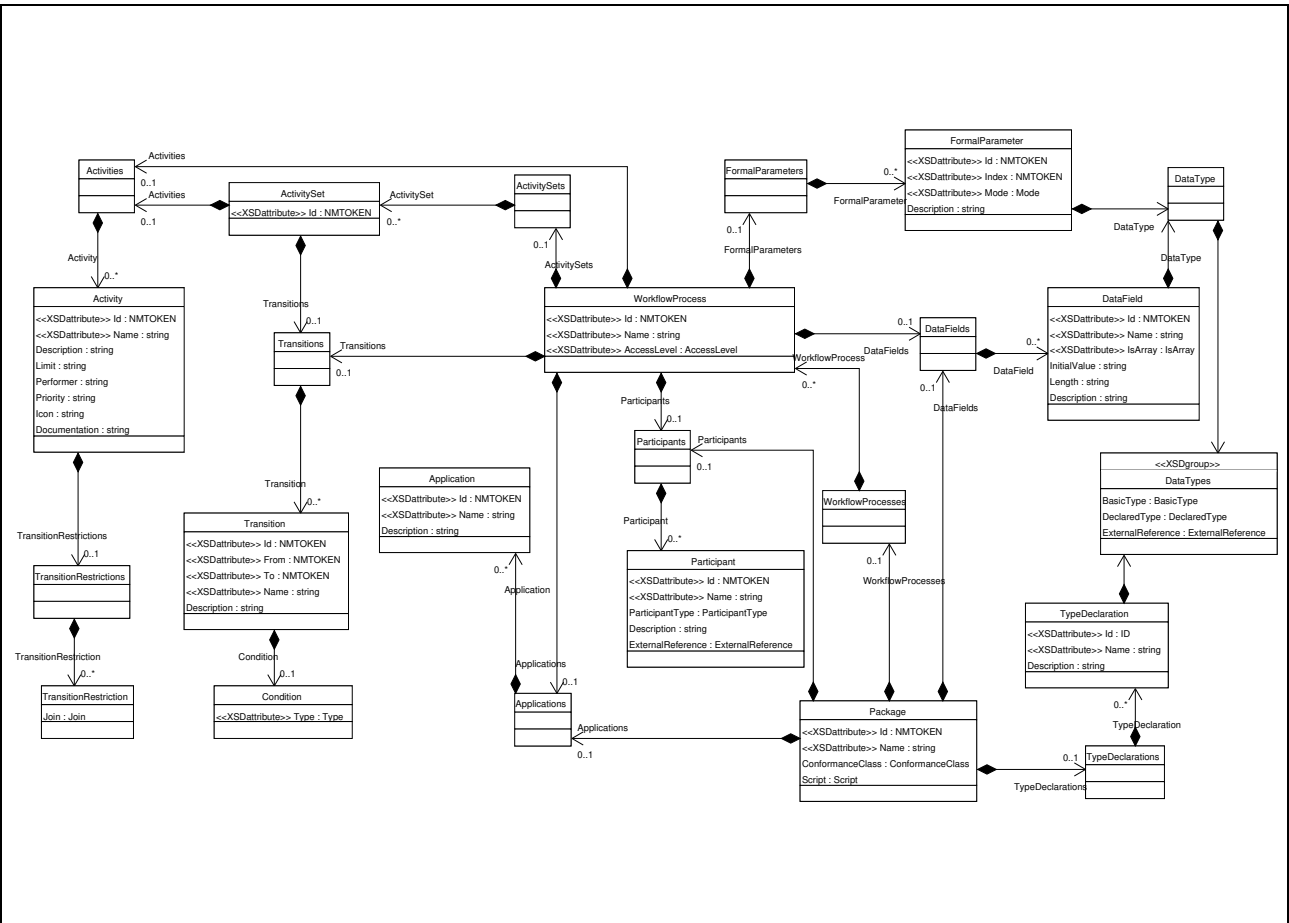


Figura 3.8: Modelo de dados para a proposta XPDL

Os dados são listados e disponibilizados através de uma coleção, ou repositório, (*WorkflowRelevantData*, não ilustrado na figura). O conjunto de entradas e saídas de um workflow é abstratamente especificado através de *FormalParameter*; posteriormente, um mapeamento é feito com o conjunto de dados disponível na coleção daquele workflow. A linguagem considera que os parâmetros necessários a cada atividade estarão disponíveis no repositório, deixando-os implícitos. Apesar de facilitar a representação do workflow, isso dificulta a composição de tarefas complexas e a visualização transparente de sub-fluxos, uma vez que as interfaces dessas atividades não são definidas. No modelo proposto, não só as interfaces são claramente definidas (e visíveis) como os dados podem ser explicitamente ligados.

Como as interfaces de atividades são suprimidas e o comportamento de uma atividade só é determinado no momento da execução, o conceito de tipo de atividade do modelo proposto (*ActivityType*) não pode ser refletido diretamente em XPDL.

O conceito de tipo de atividade, em XPDL, tem outra conotação. Ele é usado para determinar se uma atividade é executável de alguma forma (*implementation*), se ela é de roteamento (*route*) ou se é de bloco (*Block*). O primeiro caso se subdivide em execução manual (agente não computacional, executada por um participante cadastrado), execução automática (executada por uma aplicação cadastrada) e sub-fluxo (outro processo). Uma atividade de roteamento é uma atividade vazia que serve para elaborar estruturas de fluxo mais complexa. Já atividade de bloco é um conjunto fechado de atividades (sem transições externas), semelhante a um sub-fluxo, mas sem um workflow associado. Alguns desses conceitos não se refletem diretamente no modelo proposto. Se um agente é uma aplicação ou humano, por exemplo, não faz diferença no modelo proposto; o tratamento disso é delegado para o SGWf. O modelo proposto contempla a especificação de estruturas complexas e blocos de atividades como sub-fluxos.

Um workflow (ou processo) é representado pela entidade *WorkflowProcess* através da agregação de atividades (*Activity*), conjuntos de atividades (ou atividades em bloco, *ActivitySet*), parâmetros abstratos (*FormalParameters*), participantes (*Participant*), aplicações (*Application*) e transições (*Transition*). Um workflow pode ainda fazer parte de um pacote (*Package*), a partir do qual pode compartilhar aplicações, participantes e repositório de dados com outros workflows. No modelo proposto, tanto aplicações quanto participantes são tratados como agentes que podem cumprir papéis.

O conceito de transição em XPDL é representado pela entidade *Transition* e tem seu escopo restrito a somente explicitar o controle de fluxo entre as atividades, já que os dados são processados implicitamente pelas atividades a partir do repositório. Cada transição tem, entre outros, os campos origem, destino e uma condição associada (*Condition*). A condição determina a ativação da transição, como a pré-condição no modelo proposto. Existe outro ponto de ligação entre transições e atividades, além desses campos origem

e destino: o conceito de *TransitionRestriction*. Elementos dessa entidade podem compor uma atividade determinando seu modo de funcionamento, com respeito a transições: basicamente ramificação *AND* e *XOR* e junção *AND* e *XOR*. Variações podem ser obtidas usando combinações com atividades vazias de roteamento e essas restrições. Essas e outras estruturas de funcionamento são contempladas nativamente no modelo proposto.

Análise de XPDL

Algumas limitações são críticas para a linguagem XPDL, como evidenciado em [van03]. O primeiro, e mais grave, ponto é o da inexatidão na definição das condições de junção através das restrições de transição aplicadas a atividades. Para ilustrar essa limitação seguem as definições das condições de junção *AND* e *XOR*:

AND Junção de todos os fluxos concorrentes dentro de uma instância de processo com todas as transições de entrada na atividade: sincronização é necessária. O número de fluxos a serem sincronizados pode ser dependente do resultado das condições das ramificações *AND* anteriores.

XOR Junção para fluxos alternativos: nenhuma sincronização é necessária.

A interpretação usual de uma junção *AND* é que dois (ou mais) fluxos originados em atividades *A1* e *A2* chegando em uma atividade *A3* devem ser sincronizados para que *A3* possa ser executada. Caso uma das atividades *A1* ou *A2* não seja executada, *A3* também não será executada. Entretanto, de acordo com a definição de *AND* em [WfM02], pode ser que seja necessário considerar os splits anteriores para fazer a sincronização, de forma que *A3* pode vir a ser executada mesmo que alguma atividade de origem anterior não o tenha sido.

No caso do *XOR*, não é possível determinar o que ocorre quando mais de uma transição chega em uma atividade e essas transições são agrupadas por esse operador. Pode ser que somente a primeira ative a atividade, ou todas a ativem sequencialmente. É possível ainda que se esteja usando a interpretação usual de uma junção *XOR*, i.e., é determinado previamente que somente uma das transições de entrada será ativada em uma dada execução.

Não é possível, a partir das definições de [WfM02] determinar qual a interpretação correta para esses operadores. Com isso, a representação fica comprometida indiretamente, já que SGWf diferentes podem implementar os operadores com interpretações conflitantes.

Outra questão em aberto é o que deve ocorrer quando há mais de uma atividade inicial e/ou final. É possível especificar tal situação em XPDL, mas o que deve de fato ser executado não é claramente definido. Outras construções não contempladas incluem: determinar dinamicamente o número de instâncias de uma atividade; impossibilidade de

delegar uma escolha para o ambiente de execução (usuário, por exemplo) fora de uma atividade; e não existência de formas de cancelar uma atividade ou mesmo um workflow, a não ser através do elemento *deadline*.

Além desses problemas, um problema específico ao modelo proposto é a dificuldade de se representar tipos de atividades e conectores em atividades. Isso ocorre por causa da inexistência de interface de atividade em XPDL.

3.2.2 WS-BPEL

A linguagem WS-BPEL (chamada somente de BPEL no restante deste capítulo) também é baseada em XML, usando definições em XML Schema [Oasc]. Entretanto, seu objetivo é servir como linguagem de especificação para composição de serviços, mais especificamente, serviços Web. Os elementos dessa linguagem e sua compatibilização com o modelo proposto são discutidos a seguir.

As Figuras 3.9 e 3.10 ilustram o modelo extraído do documento XML Schema que define a linguagem. O modelo foi dividido em duas fases para facilitar a visualização dos conceitos. A primeira figura (3.9) ilustra os conceitos básicos e a estruturação de um processo; já a segunda (3.10) mostra os conceitos de estruturação de atividades. Existem ainda os conceitos *partnerLinkType* e *tRole*, definidos em um documento à parte [Oasd], que não são ilustrados no diagrama.

Vale notar que BPEL é uma linguagem que especifica um workflow hierarquicamente, i.e., embutindo atividades em outras atividades de maior nível. Uma atividade mais externa, por exemplo, pode especificar uma seqüência de duas atividades que por sua vez especificam ramificações internamente. Essa abordagem de representação difere da abordagem do modelo de dados proposto, bem como da linguagem XPDL, que utilizam uma representação em forma de grafos. Na realidade, BPEL também permite a utilização de uma representação em forma de transições entre grafos, mas com uma série de limitações.

Como BPEL é especificado a partir de descrições de serviços Web, i.e., documentos WSDL, como mostrado no Capítulo 2, o tratamento de dados e agentes executores é feito dentro deste contexto. Dados são tratados através da entidade *tVariable*, que especifica variáveis que podem ser usadas para processamento das atividades invocadas e/ou para o controle do estado da execução. Uma variável pode ser: mensagem (referente a uma definição de tipo de mensagem WSDL); um atributo de tipo simples definido usando XML Schema; ou um elemento de XML Schema. Para tratar tipos de dados complexos, deve-se usar esta última alternativa, definindo um novo tipo de elemento em um documento XML Schema. No modelo proposto isso significa que os tipos de dados devem ser refletidos em tipos simples da linguagem XML Schema e, caso não seja possível, em elementos definidos em XML Schema.

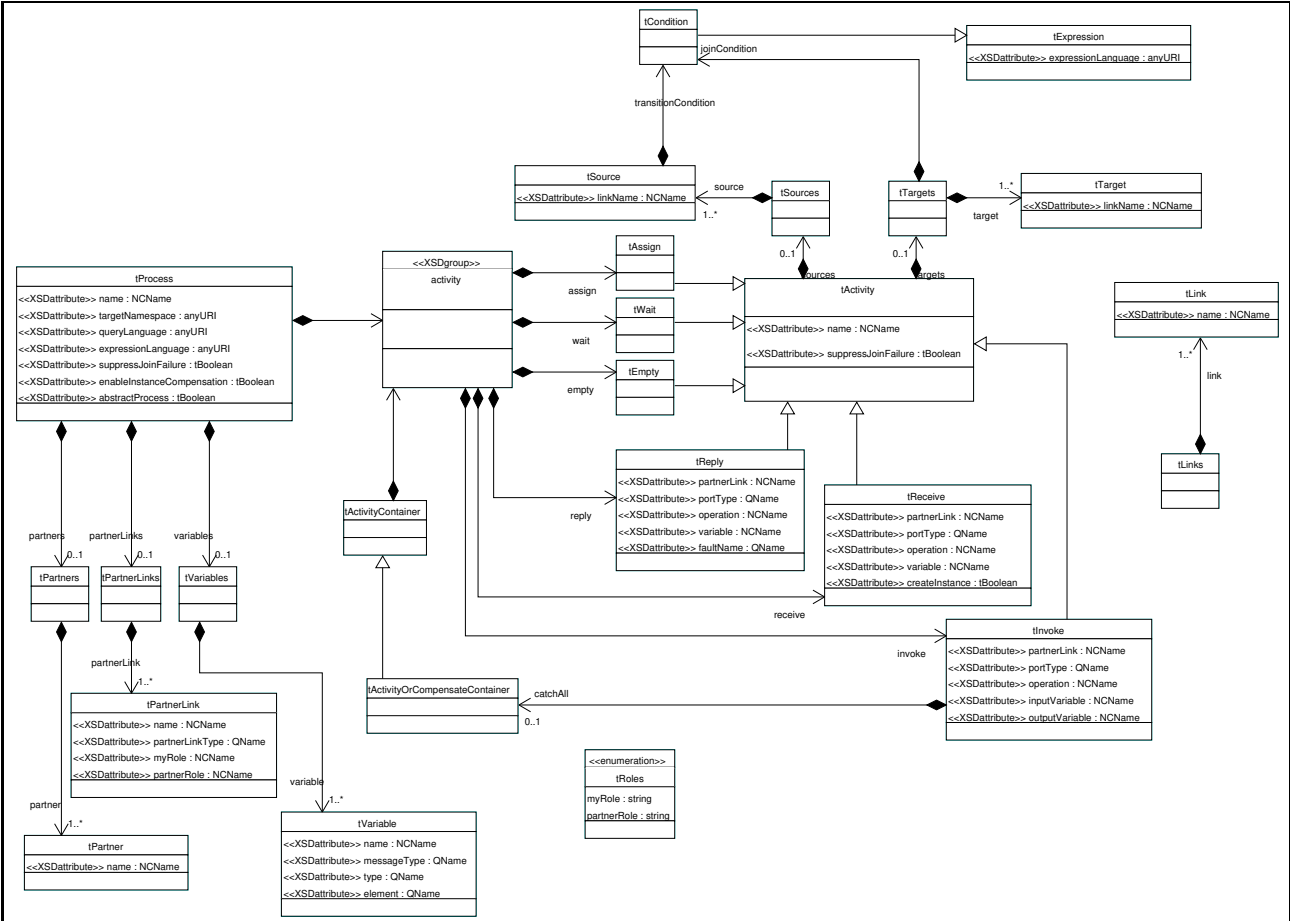


Figura 3.9: Modelo de dados para a proposta WS-BPEL: processo, atividades básicas e transições

Em seguida são especificados tipos de parceiros (*PartnerLinkTypes*, não mostrado nas figuras para manter a legibilidade) definidos em um documento separado. Esses tipos fazem a ligação com papéis (*tRole*) aos quais esses tipos estão associados e a portas WSDL, que definem a operação efetiva a ser chamada em um serviço Web. A partir desses tipos de parceiros são definidos os parceiros efetivamente. Esses é que serão invocados para a execução do processamento de atividades. Um papel em BPEL tem um escopo um pouco mais estrito que no modelo proposto, descrevendo o comportamento esperado de um serviço. O conceito *ActivityType* do modelo proposto pode ser usado para comparar um tipo de parceiro (e a definição de porta WSDL correspondente) a um tipo de atividade. Já um parceiro específico de BPEL pode ser visto como um agente do modelo proposto.

Uma atividade é representada pela entidade *tActivity* e pode ser especializada em diversos subtipos: (i) básicos, na Figura 3.9 *tInvoke*, *tReceive*, *tReply*, *tEmpty*, *tWait* e *tAssign*; e (ii) estruturados, na Figura 3.10 *tPick*, *tSwitch*, *tTerminate*, *tWhile*, *tSequence* e *tFlow*. Os subtipos básicos de atividades lidam com a chamada e controle de execução das atividades (*tInvoke*, *tReceive*, *tReply*), com a execução de atividades vazias (*tEmpty*), com atividades de temporização de execução (*tWait*) e atividades para cópia de dados de uma variável para outra (*tAssign*). Já os subtipos de atividades estruturados materializam os comportamentos estruturados possíveis dentro de BPEL, como descrito a seguir.

Uma sequência (*tSequence*) contém uma ou mais atividades que devem ser executadas sequencialmente. Uma seleção (*tSwitch*) lista uma série de atividades associadas a condições; a primeira atividade da lista que tiver sua condição avaliada como verdadeira é executada e as demais são descartadas, tendo comportamento de uma ramificação *XOR*. Essa classe de atividade permite ainda que uma atividade seja executada caso todas as alternativas anteriores falhem (condição *otherwise*). Uma repetição do tipo *while* pode ser construída com uma atividade *tWhile*, que repete a atividade interna enquanto a condição de avaliação associada for verdadeira. Uma atividade do tipo *tPick* aguarda a ocorrência de um evento, que pode ser uma mensagem ou um alarme (*time-out*) associado e então executa a atividade interna. Uma ramificação do tipo *AND* é obtida através da atividade *tFlow*, que executa concorrentemente todas as atividades internamente especificadas. A atividade *tTerminate* termina a execução da instância de processo corrente.

Dentro de uma atividade *tFlow* é possível o uso de arestas, especificando atividades internas como origem ou destino, tornando possível refinar o controle de dependências entre essas atividades.

As atividades básicas de BPEL são, no fundo, serviços Web descritos em WSDL. Portanto, para estabelecer compatibilidade com o modelo proposto, as atividades básicas do modelo devem ser traduzidas em documentos WSDL especificando suas interfaces associadas a parceiros *tPartners* em BPEL. Essas especificações devem respeitar os tipos de atividades correspondentes, que são representados em documentos WSDL associados

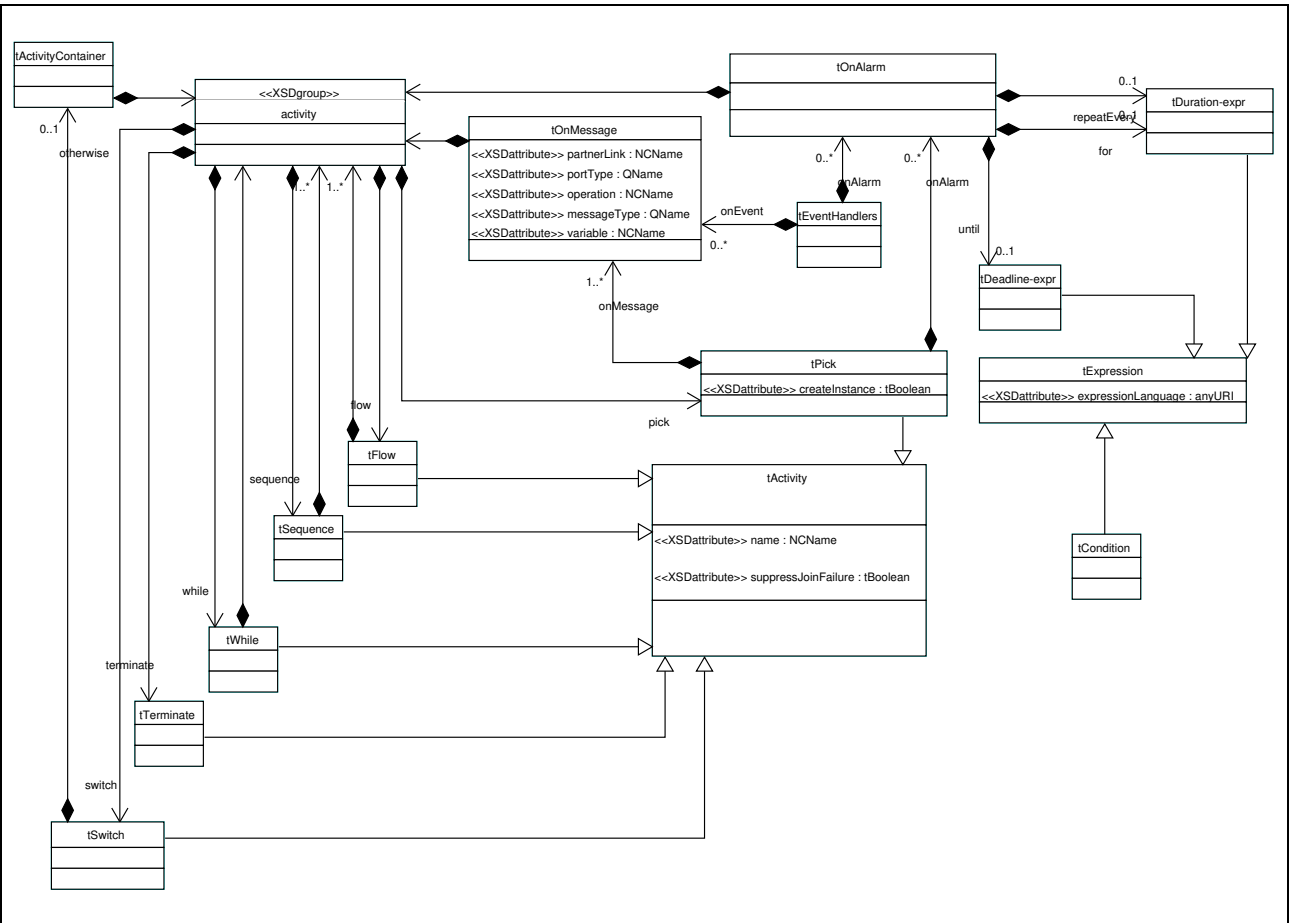


Figura 3.10: Modelo de dados para a proposta WS-BPEL: atividades estruturadas

a tipos de parceiros em BPEL.

A compatibilização do modelo com as atividades estruturadas de BPEL é feita considerando um conjunto fechado de atividades do modelo proposto, selecionado de acordo com atributos que especifiquem algum comportamento estruturado. Como exemplo, uma atividade do modelo proposto que tem duas atividades seguidas e atributo de modo de controle de saída com valor *XOR* pode ser representada como uma seqüência (*tSequence*) de duas atividades em BPEL: uma atividade básica, que executa o processamento da atividade original, seguida de uma atividade *tSwitch*, que leva às outras duas. Mais detalhes de como essa tradução deve ser feita são apresentados no Capítulo 5.

A entidade *tProcess* é responsável por agregar a definição do workflow, sendo composta de um conjunto de atividades, tipos de parceiros, parceiros e variáveis, todos usados na especificação desse workflow.

Análise de WS-BPEL

Mesmo sendo mais completa que XPDL, a BPEL também tem algumas limitações importantes. Em primeiro lugar, não permite a construção de estruturas dos tipos ciclo arbitrário e junções discriminadora e múltipla [WvdADtH03]. Com isso, não é possível nativamente representar ciclos com pontos de saída arbitrários, somente ciclos com semântica de *while*, nem fazer junções com múltiplas ativações ou ativação imediata após o recebimento da primeira transição disponível. Além disso, não há suporte direto à especificação de sub-fluxos.

Entretanto esses problemas podem ser resolvidos com paliativos ou estendendo a linguagem, como mostrado no Capítulo 5. As restrições de XPDL são mais difíceis de contornar por terem origem na própria semântica da linguagem.

Esses motivos levaram à escolha de BPEL, com pequenas adaptações, como a linguagem para representar workflows em XML para o sistema WOODSS. Outro fator que também foi importante na escolha é o crescente número de aplicações que estão adotando essa linguagem como padrão de representação de workflows.

3.3 Representação Visual de Workflows

A representação computacional de um workflow precisa ser aliada a uma representação visual. Isso é um aspecto indispensável quando se considera que workflows devem poder ser especificados por usuários em ferramentas visuais.

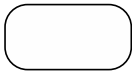
Existem duas linguagens atualmente cujo objetivo é estabelecer uma simbologia e notação para representar visualmente um workflow: Diagramas de Atividade de UML e a linguagem BPMN (Business Process Modeling Notation), introduzidas no Capítulo 2.

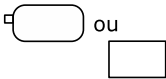
As duas são mostradas em mais detalhes a seguir e comparadas de forma a justificar a escolha da mais adequada às necessidades do sistema WOODSS.

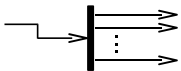
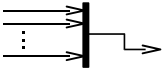
A notação escolhida foi a de diagramas de atividades de UML, pela proximidade de conceitos, como conector, por exemplo, e pela objetividade da linguagem. Além disso, inúmeras aplicações utilizam UML atualmente para os mais diversos fins.

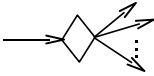
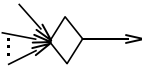
3.3.1 UML

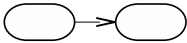
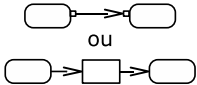
Um sub-conjunto dos elementos gráficos de um diagrama de atividades da linguagem UML pode ser usado para expressar os conceitos relacionados a workflows definidos pelo modelo proposto. A Tabela 3.1 mostra estes elementos, incluindo extensões necessárias para expressar todos os elementos do modelo proposto.

Elemento UML	Notação Gráfica	Descrição
<i>Action</i>		Corresponde a uma atividade básica no modelo proposto, i.e, a uma unidade básica de execução.
<i>ActivityNode</i>	Conceito abstrato, subdividido em <i>ExecutableNode</i> , <i>ControlNode</i> e <i>ObjectNode</i> .	Nós de atividade são os elementos básicos, junto com as ações, nos diagramas de atividades representando dados e controles de fluxo.

<i>ControlNode</i>	Conceito abstrato, materializado em <i>InitialNode</i>, <i>FinalNode</i>, <i>ForkNode</i>, <i>JoinNode</i>, <i>DecisionNode</i>, <i>MergeNode</i>.	Nós de controle servem para especificar o comportamento dos fluxos em um workflow, como por exemplo, as ramificações e junções.
<i>ObjectNode</i>		Um nó objeto representa um dado em um workflow. Esse conceito está relacionado com o de conector no modelo proposto. Para especificar que um dado está presente em um fluxo, além do controle, é preciso explicitar esse dado na representação. A figura da esquerda é mais apropriada para o WOODSS. Para contemplar os casos de ativação única ou múltipla de uma atividade por uma transição, serão usados os símbolos <i>1</i> e <i>*</i> dentro do conector da atividade, estendendo sua notação e interpretação.

<i>InitialNode</i>		Nós iniciais estabelecem o início de fluxos e determinam os pontos onde a execução deve ser iniciada em um workflow. São equivalentes a atividades do tipo <i>WorkflowConnector</i> no modelo proposto, configuradas como iniciais.
<i>FinalNode</i>	Conceito abstrato especializado em <i>ActivityFinal</i> e <i>FlowFinal</i>	Um nó final determina o término da execução de um ou mais fluxos.
<i>ForkNode</i>		Um <i>ForkNode</i> estabelece o comportamento de ramificação <i>AND</i> aplicado a uma atividade do modelo proposto.
<i>JoinNode</i>		Um <i>JoinNode</i> equivale a uma junção <i>AND</i> em uma atividade do modelo proposto.

<i>DecisionNode</i>		Um nó de decisão estabelece o comportamento de uma ramificação <i>XOR</i> em UML. A semântica para comportamento do tipo <i>OR</i> não é definida nessa linguagem. Para o caso do WOODSS, colocando um símbolo dentro do ‘diamante’: <i>X</i> para comportamento <i>XOR</i> ou <i>O</i> para comportamento <i>OR</i> .
<i>MergeNode</i>		Um <i>MergeNode</i> estabelece o comportamento de uma junção <i>XOR</i> . Novamente a semântica será estendida para permitir uma junção <i>OR</i> , com a mesma simbologia usada para estender <i>DecisionNode</i> .
<i>ActivityFinal</i>		Um <i>ActivityFinal</i> estabelece o final da execução de um workflow (ou de um sub-fluxo). Todos os fluxos correntes do processo são terminados.

<i>FlowFinal</i>		Um elemento <i>FlowFinal</i> estabelece o final de um fluxo específico, sem interferir nos demais.
<i>ActivityEdge</i>	Conceito abstrato subdividido em <i>ControlFlow</i> e <i>ObjectFlow</i>	Uma aresta de atividade estabelece uma conexão direcionada entre atividades. Pode determinar a passagem de fluxo de controle e/ou de dados.
<i>ControlFlow</i>		Um <i>ControlFlow</i> determina exclusivamente a passagem do fluxo para a atividade de destino.
<i>ObjectFlow</i>		Um <i>ObjectFlow</i> , além de passar o fluxo para a atividade seguinte, determina uma dependência dos dados produzidos pela atividade anterior.

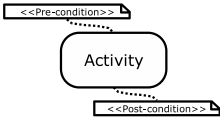
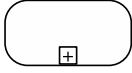
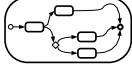
<i>Pre- / Post-conditions</i>	 The diagram shows a rounded rectangle labeled 'Activity'. Above it is a note box containing '<<Pre-condition>>'. Below it is another note box containing '<<Post-condition>>'. Dotted lines connect the note boxes to the activity node.	Pré-condições servem para avaliar o estado necessário para se iniciar a execução de uma atividade; uma pós-condição determina o estado que deve ser alcançado após a execução da atividade.
-------------------------------	---	--

Tabela 3.1: Elementos visuais para diagramas de atividades UML

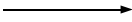
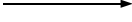
3.3.2 BPMN

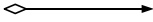
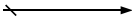
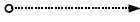
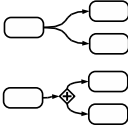
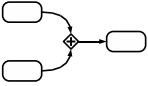
A linguagem notacional BPMN (Business Process Modeling Language) busca oferecer um padrão para a representação visual de workflows (processos). A relação de elementos de BPMN que podem ser usados para representar conceitos do modelo proposto está na Tabela 3.2.

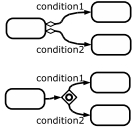
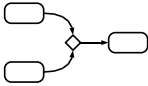
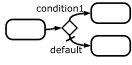
Elemento BPMN	Notação Gráfica	Descrição
<i>Activity</i>		Corresponde a <i>Activity</i> no modelo proposto, sendo especializada em <i>Task</i> , ou atividade básica, e <i>Sub-Process</i> , ou sub-fluxo.
<i>Task</i>		<i>Task</i> corresponde a uma atividade básica do modelo proposto.

<i>Process/Sub-Process</i>	Conceito abstrato subdividido em <i>CollapsedSub-Process</i> e <i>ExpandedSub-Process</i> .	Um sub-processo equivale a uma atividade do tipo <i>SubWorkflow</i> do modelo proposto. Visualmente, pode estar colapsado, “escondendo” os elementos externos, ou expandido.
<i>Collapsed Sub-Process</i>		Sub-processo colapsado.
<i>Expanded Sub-Process</i>		Sub-processo expandido, evidenciando funcionamento de elementos internos.
<i>Event</i>		Eventos marcam uma acontecimento dentro do processo, como começo e fim de processo (<i>FlowDimension</i>) e outros, como mensagens e temporizadores.

<i>Gateway</i>		<i>Gateway</i> é o elemento genérico para simbolizar controle de fluxo. Sinais internos ao símbolo de ‘diamante’ o especializam e dão significado.
<i>DataObject</i>		Um objeto de dados explicita o uso de dados nas especificações de workflows, podendo estar associado a transições e/ou atividades.
<i>FlowDimension</i>	Start ○ Intermediate ◎ End ○	Eventos <i>Start</i> (<i>End</i>) marcam o início (fim) da execução de um processo. Já eventos <i>Intermediate</i> afetam os fluxos dentro do processo mas não iniciam ou finalizam processos diretamente.

<i>GatewayControl Types</i>	XOR  ou  Event-Based  OR  Complex  AND 	Tipos de controle disponíveis. <i>AND</i>, <i>OR</i> e <i>XOR</i> têm seus significados usuais, enquanto que <i>Event-Based</i> define uma condição dependente de um evento e <i>Complex</i> define construções complexas baseadas em expressões.
<i>SequenceFlow</i>	Conceito abstrado subdividido em <i>NormalFlow</i>, <i>UncontrolledFlow</i>, <i>ConditionalFlow</i>, <i>DefaultFlow</i>, <i>MessageFlow</i>.	Determina a ordem de execução das atividades em um workflow.
<i>NormalFlow</i>		Fluxo de seguimento normal que pode passar por elementos de controle.
<i>Uncontrolled Flow</i>		Fluxo de seguimento normal que não passa por elementos de controle.

<i>Conditional Flow</i>		Fluxo cuja condição associada deve ser avaliada antes de ser tomado.
<i>DefaultFlow</i>		Fluxo <i>default</i> a ser executado em ramificações <i>XOR</i> ou <i>OR</i> caso todos os outros fluxos não possam ser executados.
<i>MessageFlow</i>		Ilustra a troca de mensagens entre entidades.
<i>Fork</i>		Ramificação do tipo <i>AND</i> .
<i>Join</i>		Junção do tipo <i>AND</i> .
<i>Decision</i>	Conceito abstrato subdividido em <i>Exclusive</i> , <i>Inclusive</i> e <i>Merging</i> .	Nó de controle que pode expressar ramificações <i>XOR</i> ou <i>OR</i> e junção <i>OR</i> .

<i>Exclusive</i>	Conceito abstrato materializado em <i>DataBased</i> e <i>Event-Based</i>	Ramificação do tipo <i>XOR</i> que pode ser baseada em dados ou em mensagens.
<i>Inclusive</i>		Ramificação do tipo <i>OR</i> .
<i>Merging</i>		Junção do tipo <i>OR</i> .
<i>DataBased</i>		Junção do tipo <i>XOR</i> baseada em valores de dados.
<i>Looping</i>	Conceito abstrato subdividido em <i>ActivityLooping</i> e <i>SequenceFlowLooping</i> .	Construção que permite a execução de repetições de uma mesma atividade ou de um fluxo.
<i>Activity Looping</i>		Repetição de uma mesma atividade.

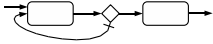
<i>SequenceFlow</i> <i>Looping</i>		Repetição de um fluxo baseado na conexão com atividade prévia.
---------------------------------------	---	--

Tabela 3.2: Elementos visuais para BPMN

3.4 Conclusões

Este capítulo apresentou o modelo de dados proposto para representação de workflows para o sistema WOODSS, comparando-o com o modelo anterior. Ambos permitem armazenar especificações de workflows em SGBD. O novo modelo permite armazenar especificações de componentes em diferentes níveis de abstração, além de indicar também condições de execução. Esses diferentes níveis de abstração induzem um método de especificação de workflow que leva a uma corretude sintática na especificação. Em seguida foram apresentados os dois modelos de dados em XML para representar workflows de maior destaque atualmente, desenvolvendo uma análise comparativa entre eles e o modelo proposto. Com isso, foi possível determinar que a linguagem WS-BPEL oferece maior poder de representação. Por fim, foram mostradas as notações de duas linguagens para representação visual de workflows. WS-BPEL, como conclusão, foi escolhida para representar workflows na Web. O diagrama de atividade de UML foi escolhido para servir como notação básica para o sistema WOODSS.

A Figura 3.11 mostra o resumo de características dos modelos em XML comparados e o proposto. Ela complementa o estudo feito na seção 2.4 e justificativa da opção por WS-BPEL para a representação em XML.

Os aspectos discutidos neste capítulo formam a base para o restante desta dissertação, a saber, a arquitetura proposta para o sistema WOODSS (Capítulo 4) e implementação das alterações no sistema e do método de exportação/importação de workflows em XML (Capítulo 5).

Aspecto	Proposta	XPDL	WS-BPEL
<i>Ramificação AND</i>	Sim	Sim	Sim (B / G)
<i>Junção AND</i>	Sim	Definição Inexata	Sim (B / G)
<i>Ramificação OR</i>	Sim	Sim	Sim (G)
<i>Junção OR</i>	Sim	? (Junção AND?)	Sim (G)
<i>Ramificação XOR</i>	Sim	Sim	Sim (B / G)
<i>Junção XOR</i>	Sim	Definição Inexata	Sim (B / G)
<i>Múltiplas Instâncias</i>	Sim	? (Junção XOR?)	Não
<i>Discriminação</i>	Sim	? (Junção XOR?)	Não
<i>Sub-Fluxos</i>	Sim	Sim (Sem Interfaces)	Parcial (Novo Serviço)
<i>Encapsulamento</i>	Sim	Não	"Externo"
<i>Ciclos</i>	Sim	Sim	Tipo <i>while</i>
<i>Estrutura</i>	Grafo	Grafo	Bloco + Grafo

Figura 3.11: Comparação entre o Modelo de Dados Proposto, XPDL e WS-BPEL

Capítulo 4

Arquitetura

Este capítulo trata da nova arquitetura proposta para o sistema WOODSS, considerando as necessidades de evolução do sistema para funcionamento integrado à Web. A seção 4.1 apresenta uma visão informal de alto nível de todos os elementos que constituem a proposta, evidenciando suas interações e as motivações para isso. A seção 4.2 descreve em mais detalhes cada um dos componentes, enfocando os mais importantes para este trabalho.

4.1 Visão Geral

Esta seção discute as alterações na estruturação do sistema WOODSS para adaptá-lo para um ambiente Web, descrevendo a solução adotada. Essa descrição tem como objetivo dar uma visão de alto nível da nova estrutura do sistema usando uma abordagem informal de módulos e seus pontos de intercomunicação.

O sistema WOODSS se acoplava a somente um software, o SIG IDRISI, e funcionava em um ambiente *desktop* mono-usuário [SMRY99, Res03, Roc03], armazenando dados em um SGBD relacional. Dessa forma, o usuário podia interagir com o SIG, realizando operações para análise geográfica, e com o WOODSS, fazendo edição de workflows.

Nesta arquitetura, o usuário passa a poder interagir com o WOODSS também por meio da Web, através de páginas dinâmicas e de uma aplicação para edição de workflows, além do acesso às aplicações e ao próprio sistema no modo *desktop*. Do ponto de vista das aplicações que podem ser utilizadas pelo sistema, as extensões incluem suporte à inclusão de aplicações genéricas, através da construção de *drivers* acopláveis especializados. Além disso, a arquitetura também prevê o uso de serviços Web como aplicações que podem ser invocadas. Os *drivers*, nesse caso, passam a poder ser implementados de forma bastante simplificada. Com isso, o espectro de aplicações das quais uma especificação de workflow pode fazer uso é bastante expandido.

Outro ponto importante é a possibilidade de se oferecer a execução automatizada de workflows como um serviço Web. Com isso, processos complexos especificados como workflows podem ser executados de forma transparente levando seus resultados a outras aplicações Web de forma automatizada.

O primeiro ponto alterado na arquitetura foi no sentido de considerar aplicações genéricas, ao invés de somente um SIG. Dessa forma, a interface de comunicação com os outros sistemas deveria ser generalizável. Isso foi feito eliminando referências específicas a operações do IDRISI. Uma solução que facilita essa tarefa é a escolha de serviços Web como forma de comunicação com aplicações em geral. Isso não elimina a possibilidade de utilizar módulos especializados para comunicação com aplicações não encapsuladas por serviços Web.

Isso leva ao segundo ponto buscado, o funcionamento do sistema na Web. A Figura 4.1, organizada em camadas, ilustra o funcionamento desejado. A *camada de aplicações* oferece funcionalidades de aplicações específicas para serem usadas como atividades básicas em workflows. Essas aplicações podem ser aplicações locais (*desktop*) ou remotas (através de serviços Web). A *camada de gerência de workflows* contempla as atividades de especificação e execução de workflows. Já a *camada de serviços Web* ilustra a possibilidade do funcionamento do sistema WOODSS como um serviço Web, i.e., fornecendo funcionalidades a outros serviços Web, baseadas no gerenciamento e execução de workflows.

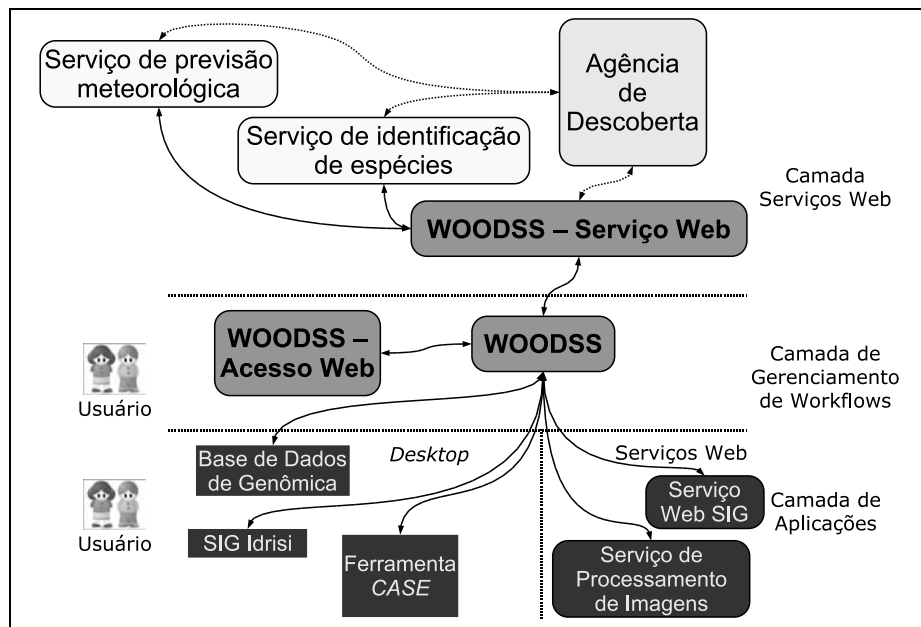


Figura 4.1: Visão de funcionamento desejado em alto nível.

Tendo em vista o funcionamento desejado ilustrado na Figura 4.1, foi especificada para o WOODSS uma nova arquitetura, mostrada na Figura 4.2. Esta figura detalha a camada

de gerenciamento de workflows da Figura 4.1, com os serviços da camada de serviços Web sendo representados pelo elemento “Serviço Usuário”.

A arquitetura está centrada em dois serviços: *serviço gerenciador de documentos* e *serviço gerenciador de workflows*. Um workflow é considerado um documento executável, cuja especificação é armazenada e mantida dentro do serviço de documentos. A arquitetura contempla ainda um serviço de ontologias e vários serviços usuários.

Essa arquitetura considera dois tipos de usuários: humanos e serviços Web. O primeiro tipo pode interagir de formas diferentes com o sistema: (i) especificando workflows (especialistas do domínio sobre o qual o workflow atua); (ii) gerenciando a execução de workflows (administradores de sistemas de execução); e (iii) solicitando a execução de workflows e acessando os dados produzidos. Serviços Web podem inserir especificações de workflows, pedir a execução de workflows previamente especificados, interferir (gerenciando ou não) na execução de workflows e fazer uso de dados produzidos pela execução desses workflows. Os usuários humanos têm disponíveis ferramentas específicas para cada uma das tarefas, como será visto. Já as interações de serviços Web com o sistema são todas feitas através do serviço gerenciador de documentos.

As ferramentas de especificação (ou criação), gerenciamento e as páginas dinâmicas são oferecidas como interface aos usuários humanos para interagir com o sistema. Um gerenciamento de perfis de usuário pode ser feito para personalizar o ambiente de desenvolvimento para especificação de workflows.

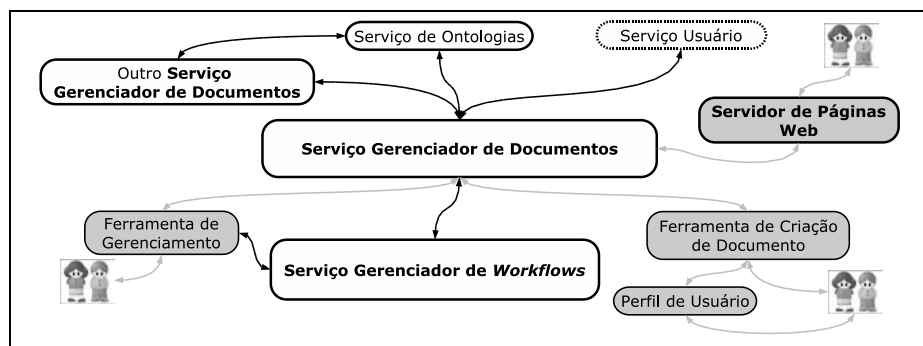


Figura 4.2: Proposta de arquitetura em alto nível para o sistema WOODSS – camada intermediária da Figura 4.1.

A ferramenta de criação é uma das partes principais do trabalho. Ela permite especificar e editar workflows (armazenados pelo gerenciador de documentos) e pedir sua execução ao gerenciador de documentos, que a repassa a um serviço gerenciador de workflows ativo. Esta ferramenta também permite criar outros tipos de documentos como os introduzidos em [PMRR05, Res03] para documentação de processos de decisão e descrição de planos. Entretanto esses outros dois tipos de documentos são estáticos e somente os

workflows são passíveis de execução. A ferramenta tem duas formas de funcionamento: como *desktop* e integrada à Web.

A ferramenta de gerenciamento cuida da interface do usuário com o gerenciador de documentos e com um ou mais serviços gerenciador de workflows ativos. Ela pode ter interface Web ou *desktop*. As funcionalidades deste módulo incluem: edição de atributos e metadados de documentos, no gerenciador de documentos, e execução e controle dinâmico de execução nos gerenciadores de workflows. Esse não é um módulo essencial para o trabalho, não sendo por isso explorado em todo seu potencial.

O serviço de gerenciamento de documentos cuida do armazenamento e disponibilização dos documentos criados. São considerados documentos especificações de workflows, descrição de processos de tomada de decisão e descrições de planos. Além disso, esse serviço também disponibiliza especificações de workflow para serem executadas e cuida de invocar um sistema gerenciador de workflows para fazer a execução de um workflow selecionado por um usuário. Ao final da execução, encaminha os resultados para os interessados. Esse serviço mantém assim interfaces com outros serviços e ferramentas. Essas interfaces podem ser especializadas (setas em tom de cinza) ou de serviços Web (setas pretas). As interfaces de serviço Web serão descritas em WSDL e usarão mensagens em XML. Essas mensagens XML são outro ponto crucial para o trabalho, já que as especificações de workflows em XML são indispensáveis para integrar um sistema desse tipo à Web. Por esse motivo, essa interface e a especificação das mensagens serão tratadas em maiores detalhes na seção 4.2 e no Capítulo 5.

Podem existir várias instâncias de gerenciadores de documentos. As interfaces entre esses serviços serão abordadas de forma marginal na seção 4.2, já que esse não é um aspecto importante para este trabalho. Além disso, serão desconsideradas as possíveis interações entre duas instâncias de serviços gerenciadores de workflows.

O serviço gerenciador de workflows recebe especificações de workflows em BPEL para executá-los. Aspectos de execução não são a preocupação primária deste trabalho. O foco é a representação das especificações de workflows, tanto em XML quanto em SGBD relacionais (Capítulo 3).

O serviço Web de ontologias oferece meios de se validar e anotar especificações de workflows e suas atividades de acordo com o domínio. Inicialmente estarão disponíveis ontologias para os domínios agrícola e espacial, introduzidas em [Fil03, FLP⁺03].

Em resumo, a ferramenta de criação de documentos, o armazenamento e recuperação de documentos e as interfaces de comunicação que transmitem especificações de workflows são os elementos mais relevantes para este trabalho.

4.2 Visão Detalhada

Esta seção discute em detalhes os principais módulos da arquitetura proposta para o sistema WOODSS. Três diferentes aspectos são considerados para essa discussão, usando diagramas UML para auxiliar a visualização das estruturas. O primeiro trata da arquitetura lógica, usando diagramas de pacotes detalhados com suas principais classes para mostrar a estrutura para a implementação do sistema. A arquitetura física usa diagramas de distribuição (*deployment*) para mostrar a organização dos principais componentes em tempo de execução. O terceiro aspecto considera problemas relacionados à concorrência entre tarefas, *threads* ou processos, por exemplo.

4.2.1 Visão lógica

A arquitetura lógica evidencia aspectos estáticos da implementação do sistema, mostrando sua organização em alto nível. Em uma implementação utilizando orientação a objetos, os elementos discutidos são os pacotes e as principais classes do sistema. Para o funcionamento na Web o sistema WOODSS foi dividido em módulos que devem ser executados como processos independentes mas que se comunicam. Os principais módulos são o *serviço de gerenciamento de documentos*, o *serviço de gerenciamento de workflows* e as *ferramentas de criação e de gerenciamento*. Outros módulos, que serão discutidos em menor detalhe, são o *serviço de ontologia* e o *servidor de páginas Web*.

Os principais módulos do sistema estão ilustrados na Figura 4.3, com diagramas de pacotes e suas principais classes. O funcionamento em conjunto desses elementos será discutido posteriormente no texto, juntamente com a arquitetura física e a visão de processo. Aqui serão discutidas as funcionalidades e comportamentos esperados de cada um dos módulos.

A ferramenta de criação (*DocumentCreationTool*) é composta por cinco pacotes. O pacote *WoodssDocClient* é o pacote principal da ferramenta (inicia a execução) e é responsável pelo gerenciamento e manipulação em memória dos documentos em edição. Este pacote é responsável por disponibilizar operações possíveis para a interface com o usuário e validar as operações requisitadas pelo usuário. Operações podem ser, por exemplo, incluir uma nova atividade ou conectar uma transição a uma atividade em um workflow. O pacote *UserInterface* gerencia os elementos de interface gráfica com o usuário, incluindo menus, barras de ferramentas e as próprias representações gráficas dos documentos. O pacote *ImportExport* controla os mecanismos para gravar e abrir documentos para edição. Esse pacote baseia-se em *drivers* que traduzem de uma representação específica (BPEL, por exemplo) para uma compatível com a do pacote *WoodssDocClient*. Os drivers inicialmente disponíveis são para BPEL (*BpelDriver*) e para bancos de dados relacionais (*DataBaseDriver*) – mais especificamente PostgreSQL [Pos] na implementação atual do

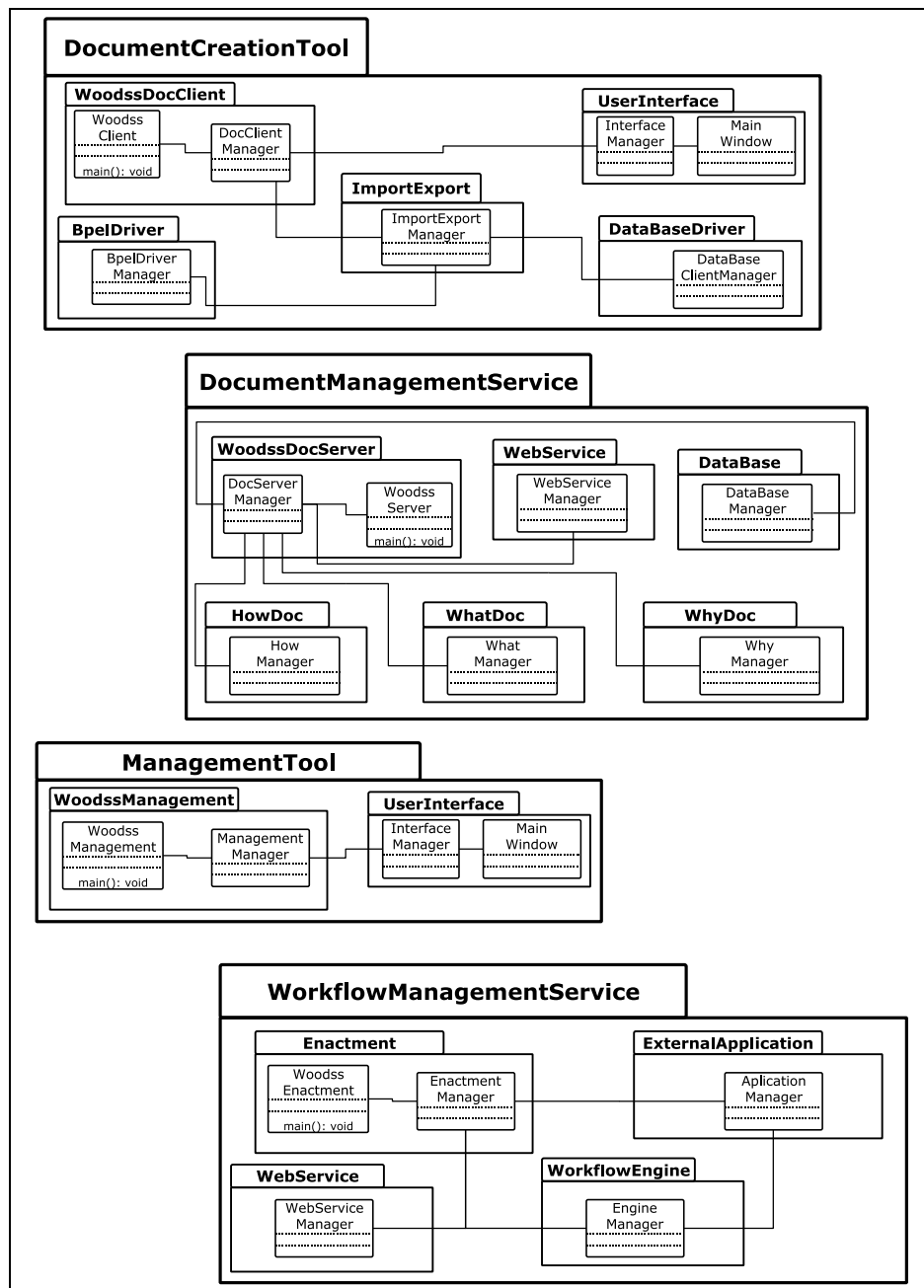


Figura 4.3: Principais módulos do sistema detalhados em pacotes

WOODSS.

Os diferentes tipos de documentos são armazenados e gerenciados pelo serviço de gerenciamento de documentos (*DocumentManagementService*). O principal pacote desse módulo é *WoodssDocServer* que também serve de porta de entrada para as operações realizadas através de protocolos diferentes de serviços Web. A comunicação usando mensagens para serviços Web é tratada no pacote *WebService*. O pacote *DataBase* é responsável por realizar operações no banco de dados a partir das requisições originadas em *WoodssDocService*. Os pacotes *HowDoc* (workflows), *WhatDoc* (hipermídia) e *WhyDoc* (*design rationale*) são responsáveis pela realização de operações especializadas de gerenciamento sobre os respectivos tipos de documentos.

Para gerenciar os documentos e a execução de workflows, um usuário conta com a ferramenta de gerenciamento (*ManagementTool*). Essa ferramenta tem basicamente um pacote principal (*WoodssManagement*) que controla a comunicação e as funcionalidades principais e um pacote de interface com o usuário (*UserInterface*). Este módulo é responsável por oferecer ao usuário formas de controle: para acesso ao banco de dados (incluindo aspectos de configurações de concorrência); para execução e gerenciamento de execução de workflows (no serviço gerenciador de workflows); e para manipulação de dados e metadados relacionados aos documentos armazenados no serviço gerenciador de documentos.

A execução de especificações de workflows é feita pelo serviço de gerenciamento de workflows (*WorkflowManagementService*). O pacote *Enactment* é o principal pacote deste módulo e gerencia os pedidos de execução da especificação do workflow em BPEL. Para cada solicitação é criada uma instância de um motor de execução, dentro do pacote *WorkflowEngine*. Este pacote é responsável por gerenciar as instâncias de workflow em execução. As aplicações que são invocadas pelas atividades durante a execução de um workflow são acessadas através do pacote *ExternalApplication*. Essas aplicações são cadastradas previamente através de comunicação com o pacote *Enactment*.

O serviço de ontologias é também um módulo independente. Esse serviço é acessado pelo gerenciador de documentos fornecendo referências a ontologias de domínio que são usadas para anotar e validar os workflows e suas atividades. Uma versão dessa ontologia é repassada à ferramenta de criação para que se escolha os termos que devem descrever um workflow e atividades.

4.2.2 Visão física

O ponto de vista oferecido pela descrição da arquitetura física de um sistema evidencia aspectos de distribuição dos módulos em tempo de execução. Uma das formas de se representar essa estrutura é utilizando diagramas UML de distribuição (*deployment*).

Esse tipo de diagrama é tipicamente composto por *nós* () , *componentes* () , *link de comunicação* (ligação entre nós), *interfaces* () e *dependências* (entre componentes). É usual também ilustrar instâncias de nós e componentes, no lugar de utilizar elementos abstratos, quantificando os possíveis participantes.

A Figura 4.4 é um diagrama de distribuição para a arquitetura proposta neste trabalho. Não são usadas interfaces de componentes e suas dependências para não sobrecarregar a figura. Os nós ilustrados são: ferramenta de criação de documentos; ferramenta de gerenciamento; serviço gerenciador de documentos; serviço de ontologias; servidor Web; e duas instâncias de serviço gerenciador de workflows.

As ferramentas de criação de documento e de gerenciamento são clientes do serviço gerenciador de documentos, usando Java RMI como protocolo para comunicação em uma conexão na Internet (TCP/IP). Além disso, a ferramenta de gerenciamento também pode ser cliente dos serviços gerenciadores de workflows usando mensagens SOAP na Internet. O serviço gerenciador de documentos é cliente do serviço de ontologias, para consultar referências a ontologias e recuperando partes de ontologias para realizar anotações em workflows, usando protocolo http na Internet. O servidor Web funciona como cliente do serviço gerenciador de documentos para recuperar documentos e dados relacionados. Entretanto, o papel principal do servidor Web é o de servir páginas dinâmicas para clientes usando navegadores Web (não ilustrados na figura). O serviço gerenciador de documentos aparece como cliente dos serviços gerenciadores de workflows, pedindo a execução de um determinado workflow a um desses serviços. Para executar um workflow, um gerenciador de workflows pode receber os dados juntamente com os pedidos ou fazer uso de dados disponíveis na Web através de URIs.

O elemento central do sistema é o serviço gerenciador de documentos. Esse serviço engloba um banco de dados de documentos e referências a dados genéricos para alimentar esses documentos (não necessariamente armazenados no banco de dados). O sistema operacional sobre o qual os elementos servidores serão executados é GNU/Linux. O SGBD utilizado é o PostgreSQL ([Pos]) e o servidor de aplicações (responsável pelas interfaces de serviços Web) utiliza JBoss e Tomcat ([JBo, Apa]). As funcionalidades de registro RMI do JBoss também serão aproveitadas. Para cada instância de serviço gerenciador de documentos, há exatamente uma instância de um servidor Web e no máximo uma instância ativa de ferramenta de gerenciamento. Várias instâncias de ferramentas de criação, de serviços de ontologias e de serviços gerenciadores de workflows podem ser utilizadas simultaneamente com um mesmo gerenciador de documentos.

A ferramenta de criação de documentos é composta por um módulo de interface gráfica com o usuário, um conversor XML e um comunicador RMI. Os documentos criados pelo usuário são convertidos em uma representação XML e são transmitidos ao serviço gerenciador de documentos usando RMI. O caminho inverso para importar documentos

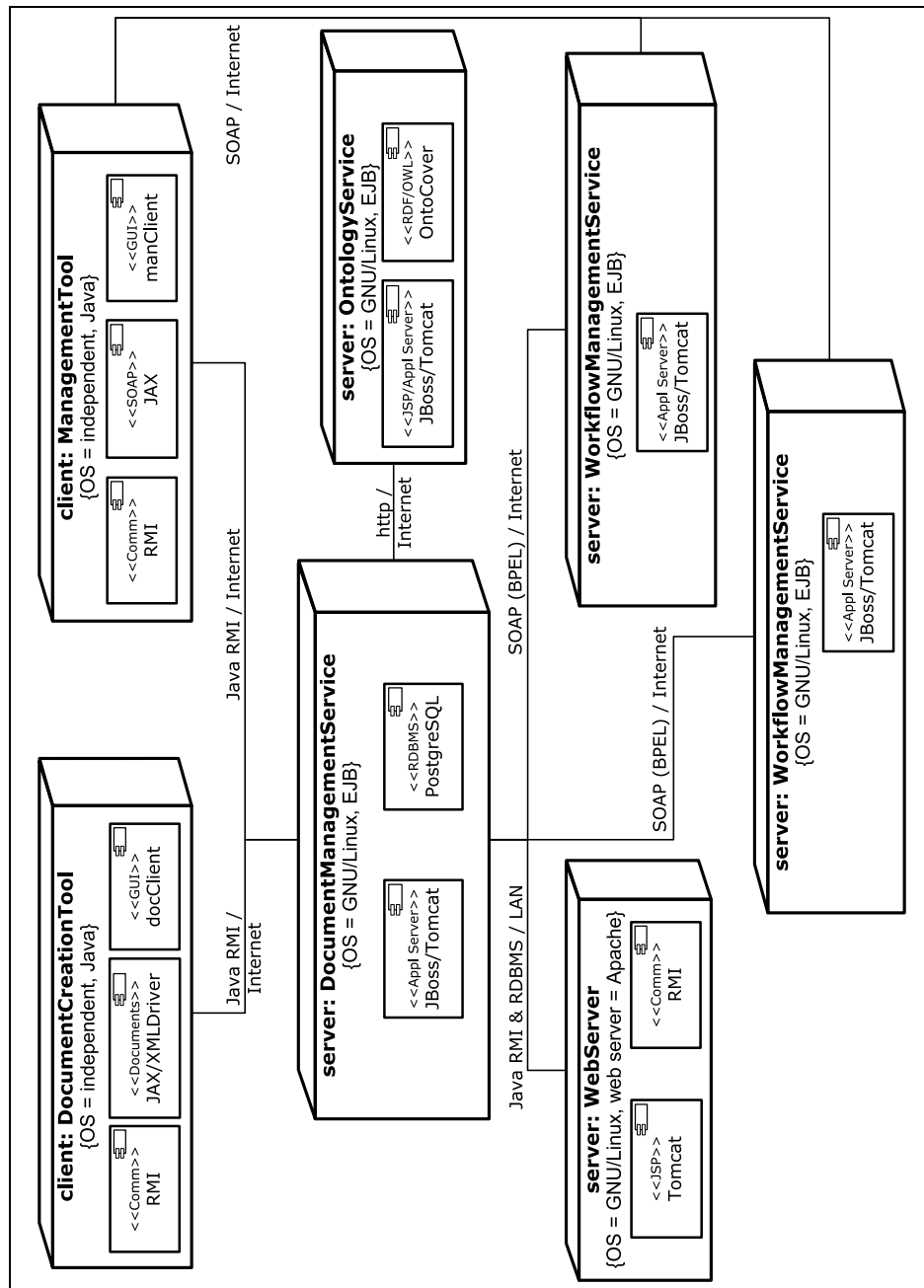


Figura 4.4: Distribuição física do sistema em nós de processamento

também é válido. A ferramenta de criação também permite gravar/abrir cópias locais dos documentos em arquivos XML. A ferramenta permite ainda solicitar a execução de um determinado workflow ao gerenciador de documentos que, por sua vez, escolherá o gerenciador de workflows mais adequado para realizar a execução.

A ferramenta de gerenciamento utiliza duas interfaces de comunicação: RMI e SOAP. A primeira é utilizada na comunicação com o gerenciador de documentos, para realizar configurações do serviço e editar documentos específicos. A segunda trata das interações com os serviços gerenciadores de workflows, lidando com aspectos de controle de execução. A ferramenta de gerenciamento permite iniciar a execução de um workflow, visualizar e interferir nas execuções.

O serviço de ontologias utiliza o sistema OntoCover ([Fil03]) desenvolvido no Laboratório de Sistemas de Informação da UNICAMP para fazer acesso a ontologias. O acesso pode ser feito através de URIs com referências a páginas dinâmicas. Com isso pode-se validar uma cobertura ontológica ([FLP⁺03]), obter uma ontologia (ou partes dela) ou atualizar ontologias. As páginas dinâmicas são suportadas pelo Tomcat e a execução de operações sobre as ontologias usa componentes Java (EJB).

O servidor Web utiliza páginas dinâmicas para oferecer ao usuário, utilizando um navegador Web, visualizações dos documentos e dos dados relacionados a eles. Pode-se, por exemplo, visualizar um workflow (imagem estática, sem possibilidades de edição) e sua descrição, pedir a execução desse workflow e visualizar os resultados obtidos (caso o tipo de dados gerado seja suportado). Para isso, o servidor faz uma chamada RMI, acessa o banco de dados e o sistema de arquivos compartilhado do serviço gerenciador de documentos, e se encarrega de construir as páginas dinamicamente, utilizando o Tomcat.

O serviço gerenciador de workflows recebe pedidos de execução de workflows especificados em BPEL. Implementa um servidor de aplicações que utiliza JBoss e Tomcat para tratar das interações e chamadas de execução. O gerenciamento desse serviço pode ser feito por uma ou mais ferramentas de gerenciamento (no máximo uma por instância de workflow).

Um ponto importante é a comunicação entre instâncias de serviços gerenciadores de documentos. Isso possibilita o tratamento distribuído dos documentos, distribuindo a carga de uso do serviço. Apesar de ter a implementação facilitada pela arquitetura proposta, aspectos de consistência e concorrência, por exemplo, limitam o uso desse mecanismo. Esse é, entretanto, um aspecto secundário neste trabalho.

Tem-se, assim, uma visão dos nós computacionais presentes no modelo e suas responsabilidades em tempo de execução, juntamente com os sistemas responsáveis por essa execução. Alguns detalhes desses sistemas serão apresentados no Capítulo 5.

4.2.3 Visão de processo

A visão de processo de uma arquitetura lida com as seqüências de ações que são executadas no sistema, ou seja, os processos que podem ocorrer no sistema. Quando se trata de um sistema que tem execução distribuída, como é o caso do WOODSS, essa visão é essencial para se entender algumas sutilezas de execução como por exemplo aspectos de controle de concorrência e tolerância a falhas. Para isso, serão descritas as seqüências de ações no sistema para a execução das principais funcionalidades do sistema.

A inicialização de alguns módulos do sistema deve ser feita em uma ordem específica. O módulo principal, essencial ao funcionamento dos demais, é o serviço gerenciador de documentos. Este deve ser o primeiro a ser inicializado, disponibilizando o banco de dados de documentos. Em seguida, e em paralelo, podem ser iniciados o servidor Web, e as ferramentas de criação e de gerenciamento. Os serviços gerenciadores de workflows e de ontologias podem ser iniciados de forma independente do restante dos módulos, já que somente realizam operações sob demanda e sem utilização dos demais módulos.

Um serviço gerenciador de documentos deve se registrar em uma agência de descoberta (UDDI) disponibilizando suas interfaces para consulta por outros serviços. Já o uso por humanos será baseado em conhecimento prévio da localização do serviço, através de endereços em páginas Web, por exemplo. Esse serviço também disponibiliza um registro RMI (*RMI Register*) no qual as instâncias da ferramenta de criação e da ferramenta de gerenciamento se registrarão. Isso é feito para possibilitar o controle de acesso aos recursos, como será visto.

Somente um servidor Web é instanciado para cada gerenciador de documentos. Portanto, a inicialização deste servidor pressupõe o conhecimento prévio da localização do gerenciador dentro da mesma LAN (*Local Area Network*). O uso de uma rede local foi escolhido por facilitar a administração, principalmente do ponto de vista de segurança, por impor um número menor de restrições de desempenho e por estar sujeita a um risco menor de falhas (de comunicação, por exemplo).

Uma ferramenta de criação, ao ser iniciada, se registra no serviço gerenciador de documentos. Uma nova *thread* é iniciada no gerenciador para cuidar da comunicação com cada instância registrada de ferramenta de criação. Para fazer o controle de concorrência de acesso aos documentos do banco de dados, duas possibilidades são consideradas: acesso simultâneo e atualização assíncrona de versão. A questão de acesso simultâneo é delegada ao SGBD, que cuida de fazer o controle de consistência de transações. Isso é possível porque o SGBD escolhido (PostgreSQL) suporta transações e faz o controle das restrições de integridade entre as tabelas do banco. A atualização assíncrona de versão diz respeito ao problema de um usuário solicitar um documento do banco para edição e posteriormente, após um período de tempo indeterminado, fazer a atualização deste documento. Nesse caso, não é razoável travar o documento para atualização, por não ser possível determinar

se a atualização vai efetivamente ocorrer. A solução proposta é simples mas não tão precisa: delegar a escolha de o que fazer ao usuário, caso modificações tenham ocorrido no documento desde quando este foi lido para edição. As opções oferecidas são gravar o documento com outro identificador ou carregar a nova versão e refazer as modificações. A opção de sobrescrever o documento não é razoável por oferecer risco de perda de dados sem checagem prévia. O mecanismo é baseado em versões dos documentos, que são atualizadas a cada modificação e comparadas (atomicamente) antes de uma atualização.

O serviço gerenciador de workflows é independente dos demais módulos, oferecendo a execução de workflows especificados em BPEL. Várias execuções podem estar sendo realizadas simultaneamente, usando identificadores para cada uma delas. As mensagens que dizem respeito a um determinado workflow sendo executado (inclusive o pedido inicial) carregam esse identificador juntamente com os dados da mensagem. Esse mesmo identificador é utilizado para manter o controle dos diversos processos paralelos iniciados para cada execução. Cada processo corresponde a uma instância de um motor de workflows (*WorkflowEngine*, na Figura 4.3). Os dados necessários ao processamento podem ser passados juntamente com a especificação do workflow ou serem referenciados através de URIs.

Para o caso da ferramenta de gerenciamento, foi adotada uma política simples de instanciação, já que esse não é o foco principal do trabalho, como já foi dito. Determina-se que somente uma ferramenta de gerenciamento pode ter controle sobre um gerenciador de documentos e sobre os processos em execução em gerenciadores de workflows originados no mesmo gerenciador de documentos.

Os aspectos de desempenho, *throughput*, escalabilidade podem ser contemplados através da execução de várias instâncias de gerenciadores de documentos. Novamente, o controle das interações entre essas instâncias é deixado como trabalho futuro.

Os tempos de resposta esperados do sistema têm como limitador as conexões de rede, já que o processamento em si não deve apresentar problemas. Portanto, para conexões via internet, o tempo de resposta esperado é o mesmo estipulado pelo protocolo subjacente (*timeout* TCP, por exemplo). Conexões locais não devem ter impacto no tempo de resposta.

Como o sistema tem seu funcionamento fortemente baseado em redes, as principais falhas esperadas são as de comunicação. Esse é também o tipo de falha que oferece mais riscos, como no caso de se interromper uma comunicação não finalizada entre dois módulos. No caso da ferramenta de criação, caso a edição esteja em curso e haja falha de comunicação, é possível gravar o trabalho localmente em sua representação XML em um arquivo texto, posteriormente sincronizando com o servidor. Para a ferramenta de gerenciamento, um *timeout* é estabelecido para que se possa alocar o controle a uma nova instância desta ferramenta. Os serviços gerenciadores de workflows devem guardar os

resultados da execução para transmiti-los de volta ao gerenciador de documentos assim que possível, ou descartá-los após um período pré-estabelecido. Caso não seja possível ao gerenciador de workflows acessar um determinado dado (ou conjunto de dados) após um número estabelecido de tentativas, o gerenciador de documentos deve receber uma notificação com a listagem dos dados inacessíveis para avisar o usuário que requisitou a execução.

Com isso, foram cobertos os aspectos de seqüência de execução mais importantes para o sistema, definindo o comportamento esperado.

4.3 Conclusões

Este capítulo apresentou a arquitetura proposta para o sistema WOODSS, considerando dois aspectos que fizeram necessárias as alterações: funcionamento integrado à Web e generalização da capacidade de lidar com documentos. Foram discutidas diferentes visões da proposta, descrevendo o funcionamento esperado da arquitetura sob diferentes pontos de vista.

A arquitetura proposta não só suporta a integração completa do sistema para funcionamento na Web, como também facilita sua extensão, alterando ou incluindo módulos autocontidos. Isso facilita, por exemplo, a adaptação para utilização com diferentes plataformas e aplicações de suporte (sistemas operacionais ou SGBDs, por exemplo) ou inclusão de novas tecnologias (como novos padrões de representação de documentos em XML, por exemplo).

O Capítulo 5 trata de aspectos de implementação, dando detalhes do funcionamento dos elementos da arquitetura mais relevantes para este trabalho.

Capítulo 5

Aspectos de Implementação

Este capítulo traz uma discussão sobre alguns aspectos de implementação das extensões propostas para o sistema WOODSS visando seu funcionamento integrado à Web. São duas as principais visões abordadas: a de representação de dados e a da ordem funcionamento dos elementos do sistema. As seções 5.1 e 5.2 tratam da representação em um banco de dados relacional e da tradução entre essa representação e WS-BPEL, respectivamente. A seção 5.3 aborda a questão do acoplamento do módulo responsável pela tradução ao restante do sistema, identificando onde este trabalho teve maior relevância dentro da arquitetura proposta no capítulo 4. Discussões sobre o histórico de desenvolvimento do sistema WOODSS são desenvolvidas em [Roc03, Res03].

5.1 Do Modelo Proposto ao Sistema Gerenciador de Banco de Dados Relacional

Esta seção trata da tradução do modelo Entidade-Relacionamento Estendido introduzido no capítulo 3 para o modelo relacional, possibilitando sua implementação em um Sistema Gerenciador de Banco de Dados (SGBD) Relacional. O SGBD escolhido para a implementação deste novo modelo foi o PostgreSQL [Pos] por ser o sistema desse tipo, implementado como software livre, mais completo disponível. Com isso o sistema MySQL [MyS], utilizado nas implementações anteriores, foi abandonado. O *script* para a criação do banco de dados usando o PostgreSQL é listado no apêndice A.

O modelo relacional ainda se mostra mais indicado para o armazenamento e recuperação de modelos por diversos fatores, como desempenho, padronização de representação e manutenção de consistência. Essas características são mais difíceis de manter ao se considerar o armazenamento diretamente em XML.

O modelo traduzido é ilustrado na Figura 5.1. Cada uma das tabelas será apresentada

brevemente, identificando os conceitos discutidos nos atributos introduzidos. A interpretação da notação da figura é a usual: linhas cheias identificam chaves estrangeiras que não podem ser nulas e linhas pontilhadas chaves estrangeiras que podem assumir o valor nulo; o círculo no final das linhas mostra quem adota a chave estrangeira (identificada nas tabelas pelo tipo “id” com a tabela de origem entre parêntesis); as chaves primárias são mostradas no campo do meio da tabela, entre o nome e os atributos.

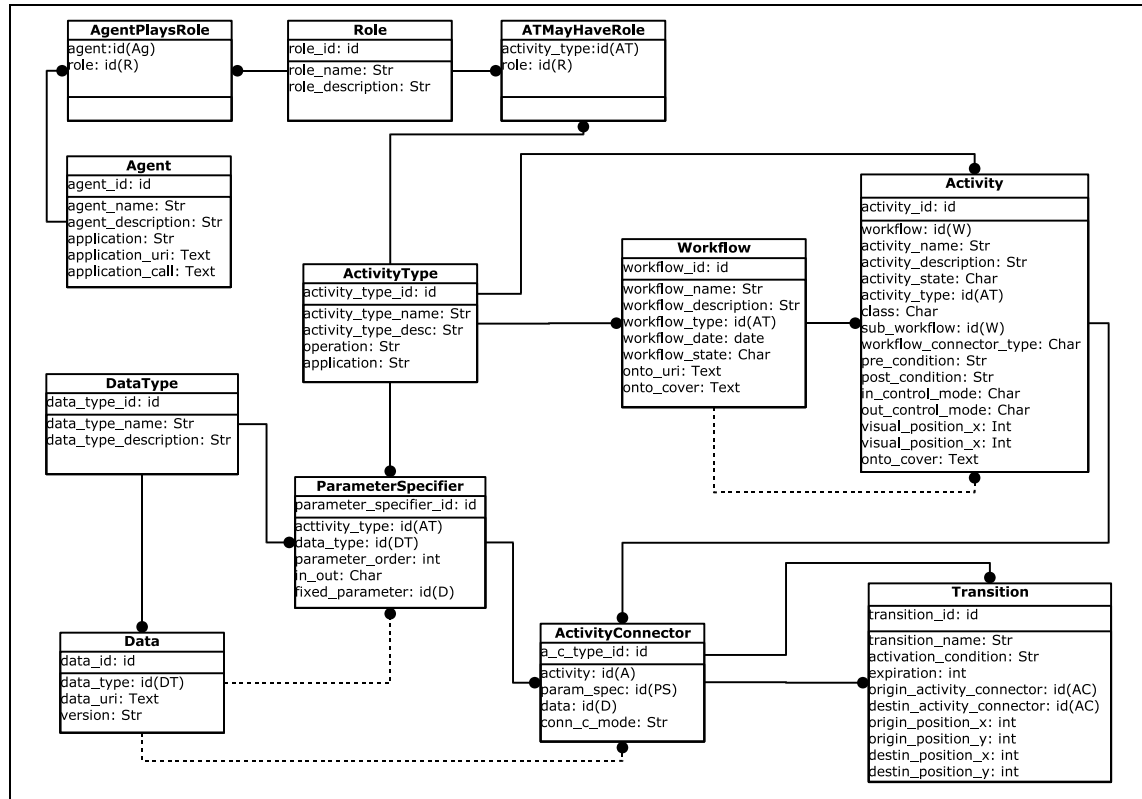


Figura 5.1: Tradução do modelo EER para o modelo relacional.

A tabela *DataType* armazena o cadastro de tipos de dados disponíveis para serem usados e as descrições desses tipos. A tabela *Data* contém referências (usando uma URI) à localização dos dados que podem ser utilizados nos processos, juntamente com o tipo do dado e a versão correspondente.

Na tabela *ActivityType* estão os tipos de atividades cadastradas, sua descrição e a operação de uma determinada aplicação que esse tipo de atividade representa dentro de um processo. A tabela *ParameterSpecifier* armazena cada elemento (parâmetro) de interface de um tipo de atividade, por isso tendo como elementos não nulos o tipo de atividade e o tipo de dado correspondentes. Essa mesma tabela guarda a ordem em que o parâmetro aparece na interface do tipo de atividade e se esse parâmetro é de entrada ou

saída. Além disso, é possível fixar um determinado parâmetro para todas as atividades de um determinado tipo, especializando a operação representada por esse tipo.

A tabela *Workflow* armazena os dados referentes a um modelo de processo, incluindo: nome, descrição textual e data de criação. Além desses, existem os atributos: (i) tipo, que é uma chave estrangeira para tipo de atividade (para o caso do workflow ser utilizado com sub-fluxo em outro modelo de processo); (ii) estado, que indica se o workflow está completo e pode ser executado (isso permite a recuperação eficiente de workflows executáveis); e (iii) os atributos *onto_uri* e *onto_cover* que indicam o endereço de uma ontologia de domínio e a cobertura ontológica dentro dessa ontologia para o modelo de processo em questão, conforme introduzido em [Fil03, VFM03].

As instâncias de atividade de um workflow são armazenadas na tabela *Activity*, onde são atributos o nome da atividade, sua descrição textual, seu estado (completo ou não, com relação a parâmetros de entrada e saída), seu tipo (referenciando a tabela *ActivityType*), classe (se é uma atividade básica, um sub-fluxo ou um conector de workflow), sub_workflow (no caso de ser um sub-fluxo, qual workflow é referenciado), tipo de conector de workflow (no caso de ser um conector de workflow, determinando se é de entrada ou saída), as pré e pós-condições, os modos de controle de entrada e de saída, sua posição visual na tela na última edição e, finalmente, sua cobertura ontológica, referente à mesma ontologia definida para o workflow. *Workflow* também faz parte da chave primária dessa tabela por ser necessário identificar dentro de qual workflow uma determinada atividade se encontra, possibilitando a utilização de sub-fluxos.

Para cada atividade, baseado no seu tipo e nos especificadores de parâmetro, é definido um conjunto de conectores de atividades na tabela *ActivityConnector*. É através desses conectores que será possível fazer as ligações de transições e dados de entrada e saída de um processo. Além disso, cada conector tem um modo de controle de conector, que pode ser discriminador (“D”) ou múltiplo (“M”), conforme explicado no capítulo 3.

As transições entre atividades são armazenadas na tabela *Transition*, que tem como atributos seu nome, condição de ativação, validade (*expiration* – que determina o número de vezes que a transição pode ser ativada), os conectores de origem e de destino e as posições visuais de origem e destino quando da última edição feita na ferramenta gráfica.

As últimas tabelas tratam dos papéis (*Role*) e agentes (*Agents*), que determinam o comportamento esperado de um atividade de um determinado tipo e quem pode executar a atividade de acordo com esse comportamento, respectivamente. Os atributos de papel são somente seu nome e uma descrição textual, enquanto agente conta ainda com a aplicação que representa e uma URI da aplicação (caso seja um serviço Web, por exemplo) ou sua forma de chamada, caso seja uma aplicação local. A tabela *AgentPlaysRole* determina que agentes podem cumprir determinado papel enquanto a tabela *ATMayHaveRole* determina os papéis que podem ser originados pelos tipos de atividade respectivos.

Com esse modelo, o banco de dados está pronto para receber dados de modelos de processos, em diferentes níveis de abstração, baseados no novo modelo introduzido.

5.2 Traduzindo entre os Modelos Relacional e de WS-BPEL

Esta seção apresenta a compatibilização entre o modelo de dados proposto, detalhado no modelo relacional apresentado na seção 5.1, e o modelo de dados de WS-BPEL. Isso será feito em passos que consideram as tabelas do modelo relacional apresentado como elementos principais, facilitando o entendimento da tradução. Além disso, usando as tabelas é mais simples de se fazer a cobertura de todos os atributos que necessitam de representação.

Um processo descrito em WS-BPEL considera que os elementos envolvidos como executores de atividades são serviços Web. Com isso a conexão desse processo com a descrição dos serviços é bastante estreita. É indispensável a utilização dessas descrições dos serviços, feita em WSDL, como mostrado no capítulo 2. Um processo exportado pelo sistema WOODSS, portanto, será composto basicamente de dois arquivos: um contendo a descrição dos "serviços" envolvidos (*“.wsdl”*) e outro com a descrição do processo em si (*“.bpel”*), ambos em XML seguindo seu respectivos esquemas.

Os esquemas, em XMLSchema utilizados são os publicados nos endereços (listados no Anexo B): <http://schemas.xmlsoap.org/wsdl/>, <http://schemas.xmlsoap.org/ws/2003/05/partner-link/> e <http://schemas.xmlsoap.org/ws/2003/03/business-process/>. O primeiro esquematiza o documento WSDL, o segundo elementos de tipagem de ligação e papéis (considerados aqui como parte integrante da descrição WSDL) e o terceiro delinea os processos em si (em WS-BPEL). Esses documentos estão listados no apêndice B deste trabalho.

Alguns dos atributos utilizados nos documentos XML não fazem parte dos esquemas originais em XMLSchema e são introduzidos utilizando o conceito de extensibilidade oferecido nos esquemas de WSDL e WS-BPEL. Outra observação diz respeito à abreviação “*lns*” utilizada como espaço de nomes nos documentos. Essa abreviação se refere ao espaço de nomes local (*Local Name Space*), utilizado para introduzir novos tipos em esquemas dentro dos documentos gerados. Vale ainda notar que por ser o processo a peça principal da exportação em XML, somente serão exportados em conjunto os repositórios (de tipos de dados e de atividades e papéis) que sejam necessários para aquele processo. Outra abordagem seria disponibilizar esses repositórios em uma determinada URI e referenciá-los do processo, mas isso distorceria o uso proposto de WS-BPEL, como será visto durante o desenvolvimento da seção. Além disso, os valores listados dentro de aspas nos códigos

de exemplo que estiverem entre parêntesis indicam que aquele nome deve ser substituído pelo valor do atributo que consta na tabela original, por exemplo, “(data_type_name)” indica que o que deve ser substituído é “tipoDeDadoA”, sendo que *tipoDeDadoA* é um valor real para *data_type_name* na tabela *Data Type*.

A primeira tabela a ser tratada é a de tipos de dados, que é exportada no arquivo WSDL dentro da seção de introdução de novos tipos (<types>). Um novo tipo deve ser introduzido, sob a definição de um novo tipo complexo de XMLSchema (<complexType>), para cada tipo de dado utilizado na especificação do processo. O nome desse tipo deve ser o valor do atributo nome da tabela (*data_type_name*) e a descrição do tipo (*data_type_description*) outro atributo. Essa definição de tipo inclui dois elementos, *data_uri* e *version* que podem receber valores no caso de uso desses tipos em variáveis representando dados, como será visto. O resultado originado deve ser parecido com:

```
<types>
  <xsd:schema targetNamespace="lns:local.name.space">
    <xsd:complexType name="(data_type_name)"
      description="(data_type_description)">
      <xsd:sequence>
        <xsd:element name="data_uri" type="xsd:anyURI"/>
        <xsd:element name="version" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

A tabela de tipos de atividade é mapeada para a definição de tipos de portas em WSDL (*portType*). Com isso um elemento *portType* tem o nome e a descrição do tipo de atividade como atributos e seu elemento interno *operation*, que deve ser único nesse processo de tradução, tem como atributos a operação e a aplicação do tipo de atividade. Os elementos *input* e *output* devem ser compostos com mensagens que são agrupadores de especificadores de parâmetros, como será visto a seguir. A especificação de um tipo de atividade em um arquivo WSDL deve ser algo como:

```
<portType name="(activity_type_name)"
  description="(activity_type_description)">
  <operation name="(operation)"
    application="(application)">
    <input message="lns:(input_parameter_specifiers)"/>
    <output message="lns:(output_parameter_specifiers)"/>
  </operation>
</portType>
```

Os especificadores de parâmetros (da tabela *ParameterSpecifier*) são representados nos dois arquivos de saída do processo. No arquivo WSDL são definidos os agrupamentos de

especificadores em mensagens de entrada ou saída, sendo que cada especificador será um elemento *part* com seu nome indicando a ordem. Esses agrupamentos são usados como tipos de mensagem para os elementos *portType*, onde é determinado se o conjunto é de entrada ou saída. No arquivo BPEL é representado o *fixed_parameter*, último elemento da tabela *ParameterSpecifier*. Para isso são introduzidas variáveis (como é feito para os conectores de atividades, como será visto), e logo no início da elemento principal da especificação da estrutura do processo, são feitas associações (*assign*) dos valores dos atributos *data_uri* e *version* para cada especificador que tem um parâmetro fixo. O sub-elemento exato do destino da atribuição é indicado através de uma expressão XPath dentro do campo *query* do elemento *to*. O código resultante é como o listado a seguir.

```
<!-- ARQUIVO:workflow.wsdl -->
<message name="(in/out_parameter_specifier_activity_type_name)">
  <part name="activity_connector_1" type="lns:(data_type_name)"/>
  <part name="activity_connector_2" type="lns:(data_type_name)"/>
  <part name="activity_connector_n" type="lns:(data_type_name)"/>
</message>

<!-- ARQUIVO:workflow.bpel -->
<variables>
  <variable name="(in/out_activity_connector_activity_name)"
    messageType=
      "lns:(in/out_parameter_specifier_activity_type_name)"
    connector_control_mode="connector_control_mode"/>
  ...
</variables>

<sequence name="main">
  <assign>
    <copy>
      <from>(data_uri)</from>
      <to variable="(in/out_activity_connector_activity_name)"
        part="activity_connector_1"
        query="//(data_type_name)/data_uri"/>
    </copy>
    <copy>
      <from>(version)</from>
      <to variable="(in/out_activity_connector_activity_name)"
        part="activity_connector_1"
        query="//(data_type_name)/version"/>
    </copy>
    <copy>
      <from>(data_uri)</from>
      <to variable="(in/out_activity_connector_activity_name)"
        part="activity_connector_2"
```

```

        query="/(data_type_name)/data_uri"/>
    </copy>
    <copy>
        <from>(version)</from>
        <to variable="(in/out_activity_connector_activity_name)"
            part="activity_connector_2"
            query="/(data_type_name)/version"/>
    </copy>
</assign>
</sequence>

```

Os papéis da tabela *Role* são traduzidos em tipos de links (*partnerLinkType*) listados no arquivo WSDL. O nome do tipo de link é composto de nome do papel (*role_name*) mais a string “Link”. O nome do elemento *role* é o nome do papel e o sub-elemento *portType* recebe o nome do tipo de atividade ao qual o papel está associado além da descrição do papel. A duplicação do uso do nome do papel é usada para manter a consistência do documento WSDL, que tem mais níveis de abstração que o modelo proposto. O trecho a seguir mostra o tipo de saída esperado.

```

<plnk:partnerLinkType name="(role_name)Link">
    <plnk:role name="(role_name)">
        <plnk:portType name="lns:(activity_type_name)"
            description="(role_description)"/>
    </plnk:role>
</plnk:partnerLinkType>

```

A ocorrência de agentes é exportada para o arquivo BPEL na forma de parceiros (*partners*), que são ligados às atividades, elementos *partnerLink*, como será visto. O trecho abaixo ilustra o tipo de saída para esse elemento.

```

<partners>
    <partner name="(agent_name)"
        partnerLink="(activity_name)"
        description="(agent_decription)"
        application="(application)"
        application_uri="(application_uri)"
        application_call="(application_call)"/>
    ...
</partners>

```

O dados do workflow, tabela *Workflow*, são exportados para o elemento raiz do arquivo BPEL, como mostrado no trecho abaixo.

```

<process name="(workflow_name)"
    targetNamespace="lns:local.name.space"
    abstractProcess="(workflow_state)"
    description="(workflow_description)"
    type="(workflow_type)"
    workflow_date="(workflow_date)"
    onto_uri="onto_cover" onto_cover="onto_cover">
    ...
</process>

```

Os elementos da tabela *Activity* são traduzidos, no documento BPEL, como *partnerLinks*, listando as atividades do processo, e *invokes* na especificação da estrutura do processo. Os atributos da tabela são convertidos em atributos dos elementos *partnerLink*, exceto os atributos *activity_type* e *workflow*, que são determinados indiretamente: tipo de atividade pelo tipo de link, pelo papel e finalmente pelo *portType*; e o *workflow* é o próprio sendo especificado (todas as atividades são do *workflow* representado, nenhuma pode ser de outro). Vale destacar o caso do atributo *sub_workflow*, ao qual é atribuído o nome do *workflow* que deve ser substituído. Esse sub-workflow origina um novo par de arquivos WSDL e BPEL, disponibilizando sua interface através de elementos *receive* e *reply* no início e fim da seqüência principal do processo.

Dentro da estruturação do processo, o uso do elemento *invoke* exige a atribuição dos parâmetros de entrada da chamada através de variáveis, que representam os conectores de atividade, como será visto em seguida. Abaixo está representada a estrutura geral da saída esperada.

```

<partnerLinks>
  <partnerLink name="(activity_name)"
    partnerLinkType="(role_name)Link"
    myRole="(role_name)"
    description="(activity_description)"
    state="(activity_state)"
    class="(class)"
    sub_workflow="(workflow_name)"
    pre_condition="(pre_condition)"
    post_condition="(post_condition)"
    in_control_mode="(in_control_mode)"
    out_control_mode="(out_control_mode)"
    onto_cover="(onto_cover)"/>
    ...
</partnerLinks>

<sequence name="main">
  ...

```

```

    <invoke name="(activity_name)"
           partnerLink="(activity_name)"
           portType="lns:(activity_type_name)"
           operation="(operation)"
           inputVariable="in/out_activity_connector_activity_name"/>
    ...
</sequence>

```

O conceito de conector de atividade é traduzido em elementos *variable* de BPEL que são usados em elementos *invoke*, especificando a atividade à qual os conectores estão ligados. O tipo da variável está associado ao tipo da mensagem que a especifica, dependendo, portanto, do especificador de parâmetro, estando de acordo com o modelo proposto. O atributo de modo de controle de conector é representado como atributo de variável, mas o comportamento modelado por esse atributo não é diretamente suportado por BPEL, como mostrado em [WvdADtH03]. Para associar os dados possivelmente atribuídos a um conector é necessário fazer uma atribuição através do elemento *assign* um passo antes de se posicionar a invocação da atividade. Para essa atribuição é necessário especificar a URI do dado e sua versão, o que só é possível através do uso de uma expressão XPath para determinar o sub-elemento exato do tipo de mensagem ao qual deve ser atribuído o valor. A expressão é especificada no atributo *query* do elemento *to*. A saída esperada tem estrutura semelhante à mostrada abaixo.

```

<variables>
  <variable name="(in/out_activity_connector_activity_name)"
            messageType="lns:(in/out_parameter_specifier_activity_type_name)"
            connector_control_mode="(connector_control_mode)"/>
  ...
</variables>
...
<sequence>
  <assign>
    <copy>
      <from>(data_uri)</from>
      <to variable="(in/out_activity_connector_activity_name)"
         part="activity_connector_x"
         query="//(data_type_name)/data_uri"/>
    </copy>
    <copy>
      <from>(version)</from>
      <to variable="(in/out_activity_connector_activity_name)"
         part="activity_connector_x"
         query="//(data_type_name)/version"/>
    </copy>
  
```

```

    </assign>
    ...
</sequence>

```

O último conceito traduzido é o de transição, que determinará também a estruturação do processo através do conceito de *link* em BPEL. Uma transição será representada por um *link* que terá sua origem (*source*) e destino (*target*) em atividades em uma determinada ordem (*link_order*). O atributo *data_type* de *link* indica o tipo de dados que é passado através dessa transição. Para expressar os possíveis dados de entrada ou saída são utilizadas atribuições (*assign*) às variáveis que representam os conectores de atividades correspondentes. O atributo *expiration* também é adicionado ao elemento *link*.

A construção do processo é feita em passos incrementais, começando pelas atividades iniciais e expandindo-o através das transições de saída. Para tanto, três casos são considerados, de acordo com o modo de controle de saída da atividade. O primeiro caso considera o modo de controle de saída “AND”. Para esse modo, as condições de transição (*transitionCondition*) dos elementos *source* são todas atribuídas o valor verdadeiro, de forma que todos os *links* possam ser tomados. Esse caso é ilustrado abaixo.

```

<sequence>
  <assign>
    <copy>
      <from>(data_uri)</from>
      <to variable="(in/out_activity_connector_activity_name)"
        part="activity_connector_x"
        query="//(data_type_name)/data_uri"/>
    </copy>
    <copy>
      <from>(version)</from>
      <to variable="(in/out_activity_connector_activity_name)"
        part="activity_connector_x"
        query="//(data_type_name)/version"/>
    </copy>
  </assign>
  <flow>
    <links>
      <link name="(transition_name)"
        data_type="parameter_specifier_data_type"
        expiration="expiration"/>
      ...
    </links>
    <invoke name="(activity_name)"
      partnerLink="(activity_name)"
      portType="lns:(activity_type_name)"
      operation="(operation)"

```

```

        inputVariable="in/out_activity_connector_activity_name">
    <source linkName="(transition_name)"
        transitionCondition="true"
        link_order="x"/>
</invoke>
<invoke name="(activity_name)"
    partnerLink="(activity_name)"
    portType="lns:(activity_type_name)"
    operation="(operation)"
    inputVariable="in/out_activity_connector_activity_name">
    <target linkName="(transition_name)"
        link_order="x"/>
</invoke>
...
</flow>
</sequence>

```

A segunda possibilidade é a de o modo de controle de saída ser do tipo “OR”, atribuindo aos atributos *transitionCondition* o valor do atributo *activation_condition* da transição. A estrutura de código para essa possibilidade é mostrada a seguir.

```

<sequence>
    <assign>
        <copy>
            <from>(data_uri)</from>
            <to variable="(in/out_activity_connector_activity_name)"
                part="activity_connector_x"
                query="/(data_type_name)/data_uri"/>
        </copy>
        <copy>
            <from>(version)</from>
            <to variable="(in/out_activity_connector_activity_name)"
                part="activity_connector_x"
                query="/(data_type_name)/version"/>
        </copy>
    </assign>
    <flow>
        <links>
            <link name="(transition_name)"
                data_type="parameter_specifier_data_type"
                expiration="expiration"/>
            ...
        </links>
        <invoke name="(activity_name)"
            partnerLink="(activity_name)"
            portType="lns:(activity_type_name)"

```

```

        operation="(operation)"
        inputVariable="in/out_activity_connector_activity_name">
<source linkName="(transition_name)"
        transitionCondition="(activation_condition)"
        link_order="x"/>
<source linkName="(transition_name)"
        transitionCondition="(activation_condition)"
        link_order="y"/>
</invoke>
<invoke name="(activity_name)"
        partnerLink="(activity_name)"
        portType="lns:(activity_type_name)"
        operation="(operation)"
        inputVariable="in/out_activity_connector_activity_name">
<target linkName="(transition_name)"
        link_order="x"/>
</invoke>
...
<invoke name="(activity_name)"
        partnerLink="(activity_name)"
        portType="lns:(activity_type_name)"
        operation="(operation)"
        inputVariable="in/out_activity_connector_activity_name">
<target linkName="(transition_name)"
        link_order="x"/>
</invoke>
...
</flow>
</sequence>

```

O terceiro caso considera o valor “XOR” do modo de controle de saída. Para esse modo o valor de cada *transitionCondition* deve ser uma conjunção lógica da *activation_condition* da transição correspondente ao *link* e a negação lógica de cada uma das condições de ativação das outras transições, como mostrado abaixo.

```

<sequence>
<assign>
<copy>
<from>(data_uri)</from>
<to variable="(in/out_activity_connector_activity_name)"
    part="activity_connector_x"
    query="/(data_type_name)/data_uri"/>
</copy>
<copy>
<from>(version)</from>
<to variable="(in/out_activity_connector_activity_name)"

```



```

        part="activity_connector_x"
        query="//(data_type_name)/version"/>
    </copy>
</assign>
<flow>
    <links>
        <link name="(transition_name)"
            data_type="parameter_specifier_data_type"
            expiration="expiration"/>
        ...
    </links>
    <invoke name="(activity_name)"
        partnerLink="(activity_name)"
        portType="lns:(activity_type_name)"
        operation="(operation)"
        inputVariable="in/out_activity_connector_activity_name">
    <source linkName="(transition_name)"
        transitionCondition="(activation_condition)A
        AND NOT[(activation_condition)B] AND NOT..."
        link_order="x"/>
    <source linkName="(transition_name)"
        transitionCondition="(activation_condition)B
        AND NOT[(activation_condition)A] AND NOT..."
        link_order="y"/>
    </invoke>
    <invoke name="(activity_name)"
        partnerLink="(activity_name)"
        portType="lns:(activity_type_name)"
        operation="(operation)"
        inputVariable="in/out_activity_connector_activity_name">
        <target linkName="(transition_name)"
            link_order="x"/>
    </invoke>
    ...
    <invoke name="(activity_name)"
        partnerLink="(activity_name)"
        portType="lns:(activity_type_name)"
        operation="(operation)"
        inputVariable="in/out_activity_connector_activity_name">
        <target linkName="(transition_name)"
            link_order="x"/>
    </invoke>
    ...
</flow>
</sequence>

```

Dois pontos importantes durante o processo de tradução são o tratamento de ciclos e de sub-fluxos. A ocorrência de ciclos no modelo deve ser detectada anteriormente ao início do processo, transformando o ciclo em uma repetição do tipo *while* de BPEL. A ocorrência de um sub-fluxo é tratada com a criação de dois novos arquivos (um WSDL e um BPEL) para representar esse sub-fluxo disponibilizando suas interfaces através dos elementos *receive*, para entrada, e *reply* para saída, conectando-os com as atividades correspondentes no workflow inicial.

A forma escolhida de traduzir não é a única. Pode-se utilizar ainda outros elementos estruturadores, como *switch* para “XOR”, por exemplo. Entretanto a representação escolhida é mais próxima do modelo escolhido, sendo portanto mais natural a sua utilização.

O processo de tradução de BPEL para o modelo proposto é feito seguindo a correspondência exata que foi apresentada para a tradução ao contrário. Os outros elementos estruturais também são suportados no modelo proposto, exceto o elemento *pick* e seus sub-elementos, que são baseados em eventos, o que não é suportado pelo modelo proposto.

5.3 Módulo Tradutor na Arquitetura Proposta

A troca de documentos entre alguns dos elementos integrantes da arquitetura proposta será feita toda na Web, incluindo a possibilidade de interação com outros serviços Web que não os presentes na arquitetura. Com isso, o módulo de tradução entre WS-BPEL e o modelo proposto será utilizado em diversos dos elementos da arquitetura. Mais precisamente, o módulo deverá estar presente em todos os elementos que precisem manipular os documentos. Com isso, exceto o serviço de ontologia não fará uso direto do módulo tradutor.

Como ilustrado na Figura 5.2, os elementos da arquitetura fazem uso do módulo tradutor para se comunicar com o gerenciador de documentos. Os documentos são armazenados em SGBD relacional e convertidos em documentos XML sob demanda.

Um dos usos do módulo é para a tarefa de se carregar um documento para edição pelo usuário na ferramenta de criação. O módulo é usado para converter em XML o documento, que é transformado de volta na sua representação padrão (do modelo proposto) em memória. Após as alterações (o mesmo para o caso de criação de um novo documento) o usuário estabelece o caminho inverso de conversão, chegando de volta no gerenciador de documentos.

Quando se utiliza a ferramenta de gerenciamento pode ser necessário conhecer alguns detalhes do workflow. Por exemplo, para solicitar a execução poder ser necessário checar a disponibilidade de recursos necessários para essa execução. Para tanto é necessário conhecer a especificação do processo.

Outro ponto, e talvez o mais crucial da arquitetura, é a submissão de um documento de

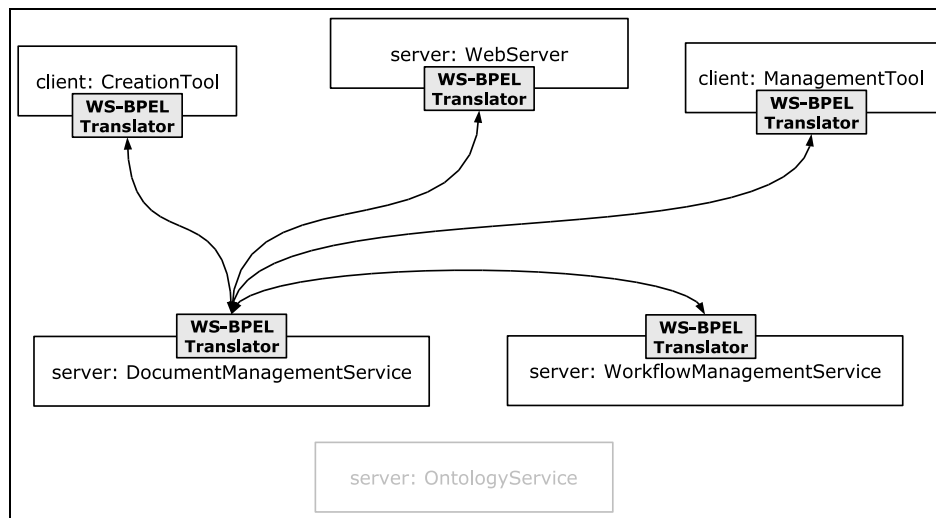


Figura 5.2: Presença do módulo tradutor na proposta de arquitetura.

workflow para a execução em um serviço de gerenciamento de workflows. Um serviço desse tipo deve ter uma interface de serviço Web para poder atender a demanda de execução de processos de fontes mais genéricas. Entretanto, a execução não pode ser feita sobre XML apenas, é necessária uma implementação concreta por baixo da interface de serviço Web. É nesse ponto que o tradutor atua.

Para o servidor Web pode ser interessante a capacidade de enviar e receber documentos em XML para fornecer esses documentos a usuários de páginas dinâmicas ou para receber esses documentos e construir essas páginas dinâmicas para exibição.

Com isso, pode-se perceber a importância do módulo tradutor nos elementos da arquitetura de modo a permitir que seja alcançada a flexibilidade do gerenciamento de documentos na Web.

O estado corrente da implementação trata apenas da Ferramenta de Criação e do módulo tradutor. Ambos estão apenas parcialmente implementados, disponibilizando funcionalidades básicas dentro do sistema.

A Ferramenta de Criação contempla apenas os elementos gráficos mais básicos da notação de diagramas de atividade UML. A inclusão dos demais elementos é um dos próximos passos de implementação. Além disso, a conexão da ferramenta com formas de importar/exportar workflows também precisa ser testada. A Figura 5.3 mostra uma cópia de tela da ferramenta, na qual dois workflows estão sendo editados.

O módulo de tradução já conta com a funcionalidade de exportação de workflows em XML. A segunda parte, de importação, é o próximo passo. Com isso pode-se acrescentar o módulo na Ferramenta de Criação, disponibilizando mais um formato de importação/exportação de documentos.

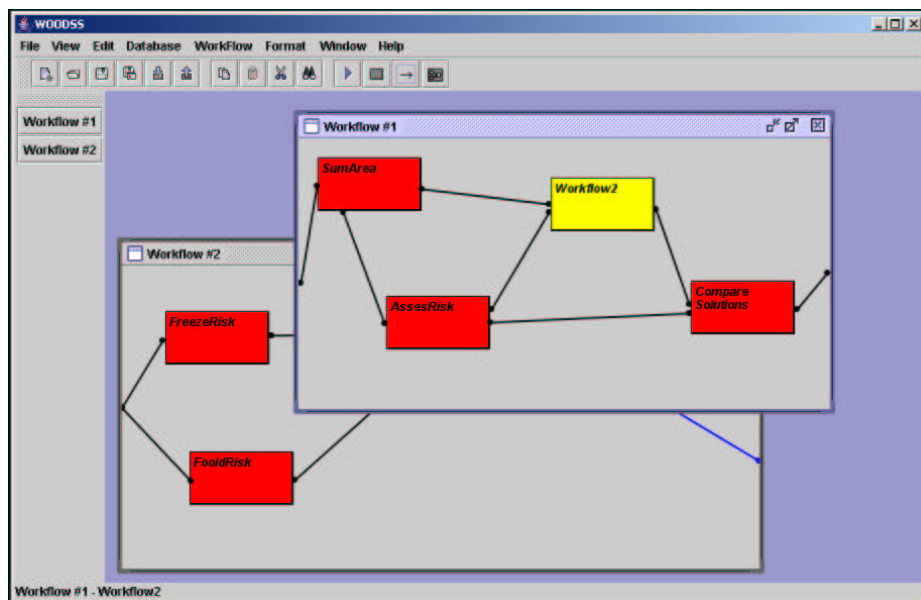


Figura 5.3: Cópia de tela da Ferramenta de Criação do sistema WOODSS

5.4 Conclusões

Esta seção tratou de algumas questões voltadas à expansão do sistema WOODSS para funcionar integrado a um ambiente Web. Para isso, o banco de dados foi detalhado, apresentando o modelo relacional correspondente. A partir desse modelo foi detalhado o procedimento de tradução de e para WS-BPEL. Finalmente foi discutida a relevância do módulo tradutor dentro da arquitetura proposta.

Capítulo 6

Conclusões e Extensões

Atividades científicas envolvem processos complexos e, freqüentemente, diversos especialistas. O foco deste trabalho foi em aprimorar os mecanismos de documentação de processos e em possibilitar sua publicação e integração na Web. Para isso a área de planejamento ambiental foi usada como base para o trabalho, tanto no aspecto de elucidação de requisitos quanto no de validação da proposta. Workflows foram os principais documentos utilizados, sendo vistos como formas de modelar processos científicos. O trabalho representa um primeiro passo para a integração e interoperabilidade de ferramentas de suporte à decisão em ambientes distribuídos na web. Para isso foram usados padrões visando a integração com a Web Semântica, tanto no ponto de vista de representação de dados quanto no de organização arquitetural do sistema computacional subjacente.

6.1 Contribuições

O trabalho partiu de um levantamento de aspectos operacionais e de documentação de atividades de planejamento ambiental. Este estudo permitiu constatar a eficiência da utilização de sistemas baseados em workflows como ferramenta de apoio a atividades de planejamento ambiental, tanto do ponto de vista de sua execução quanto de sua anotação/documentação.

O passo seguinte foi analisar as propostas já existentes sobre a utilização de workflows em atividades científicas bem como em outros ambientes. Tal análise resultou na proposta de um novo modelo de dados para a representação de workflows. A proposta do modelo levou à criação de um método incremental para a especificação de workflows, que facilita a verificação de incongruências e posterior interoperabilidade e reuso de elementos. Primeiro define-se os elementos de processamento que por sua vez podem ser conectados entre si originando novos elementos, de maior nível de abstração. Todos os elementos podem ser combinados de forma transparente para o usuário.

O modelo proposto teve o foco no uso de SGBD relacionais. Para estender o alcance dessa proposta à Web, foi necessário exportar o modelo em uma linguagem padrão utilizada nesse ambiente. Uma conversão direta para XML, utilizando XMLSchema para estruturar o modelo, teria sido a abordagem mais direta e simples de solução. Entretanto, mesmo que resolvesse o problema de publicação dos documentos na Web, o problema de integração continuaria não solucionado. Isso ocorre pela dificuldade em se definir um padrão de representação universalmente aceito. Foi necessário, desta forma, pesquisar propostas de padrões para modelos de dados utilizando XML para se conseguir o alcance desejado do ponto de vista de integração.

Cinco propostas foram estudadas, das quais duas foram selecionadas como mais adequadas aos objetivos do trabalho: XPDL (*XML Process Definition Language*) e WS-BPEL (*Web Services Business Process Execution Language*). Enquanto a primeira é uma proposta para representação de workflows em XML visando sua publicação e intercâmbio, a segunda tem um contexto mais específico: o de composição de serviços Web. Para realizar a composição de serviços utiliza-se largamente conceitos relacionados a workflows, tornando a própria estrutura do documento publicado uma especificação de um workflow. Mesmo não tendo sido criada com o objetivo específico de representar workflows em XML como é o caso de XPDL, a linguagem WS-BPEL se mostrou mais completa e adequada às necessidades do trabalho, tendo sido, por isso, escolhida.

Como esta linguagem foi criada para especificar composições de serviços Web, os conceitos associados a essa tecnologia precisaram ser levados em consideração. Com isso pôde-se vislumbrar uma maior integração do sistema com a Web, em particular com o esquema de funcionamento de serviços Web. Para tanto uma arquitetura compatível foi necessária, o que deu origem à nova arquitetura proposta na dissertação. O novo banco de dados foi criado e parte da arquitetura que diz respeito à tradução entre o banco de dados relacional e documentos WS-BPEL e a ferramenta de especificação de workflows foram parcialmente implementadas.

As principais contribuições deste trabalho são, portanto:

- Um modelo de dados para representar workflows em diferentes níveis de abstração e capaz de expressar estruturas complexas. Este modelo induz os projetistas de experimentos científicos a uma metodologia incremental de especificação de workflows;
- Análise comparativa das propostas de padrões para representar workflows em XML e adaptação de WS-BPEL para suportar o modelo proposto;
- Proposta de arquitetura para gerenciamento de documentos e execução de workflows na Web;
- Implementação parcial da arquitetura proposta.

6.2 Extensões

Trabalhos futuros podem envolver extensões diretas de implementação da arquitetura ou aumento da abrangência dos modelos de dados. Existem ainda extensões do ponto de vista conceitual, envolvendo evoluções em direções diversas.

As extensões de implementação da arquitetura proposta dizem respeito às partes restantes dessa arquitetura. Dentre elas pode-se destacar:

- **Ferramenta de Especificação.** Implementação de requisitos adicionais da ferramenta de especificação, incluindo todas as estruturas de representação disponíveis no modelo de dados para workflows, mantendo compatibilidade com a representação visual adotada (diagramas de atividades UML) e adicionando funcionalidades para facilitar a interação do usuário como controle de desfazer/refazer, associação automatizada de entradas e saídas de acordo com o tipo de dados, etc.;
- **Ferramenta de Gerenciamento.** Implementação da ferramenta de gerenciamento, especificando o tipo de interface mais adequado para o conjunto de funcionalidades previstas, levando em consideração os objetivos de integração na Web e usabilidade. As funções a serem disponibilizadas devem incluir início/finalização de execução, gerenciamento de dados de entrada e de dados resultantes (parciais e finais), controles de acesso à execução, dentre outras. Além disso, lidar com mais de uma instância de Ferramenta de Gerenciamento é interessante para um ambiente Web, mesmo exigindo um esforço de implementação maior;
- **Serviço Gerenciador de Workflows.** Implementação ou adaptação de um motor de execução de workflows que aceite especificações em WS-BPEL, encapsulando esse motor em um serviço Web. Um ponto de interesse é a interação entre dois serviços gerenciadores de workflows, em especial se considerado que pode-se delegar a execução de sub-fluxos a outros serviços. Isso se torna particularmente importante em ambientes distribuídos de execução, como o proporcionado por grids computacionais, por exemplo. Além disso, em ambientes de alto desempenho pode ser interessante utilizar protocolos de comunicação e troca de dados mais eficientes, usando tecnologias mais especializadas, oferecendo outras interfaces para a utilização do sistema além da interface Web;
- **Serviço Gerenciador de Documentos.** Extensão das capacidades de um Serviço Gerenciador de Documentos com possibilidade de funcionamento em conjunto de várias instâncias desses serviços, fazendo controles de consistência, sincronismo e concorrência. Dessa forma, pode-se obter serviços eficientes sem abrir mão da possibilidade de um significativo repositório de documentos. Outros aspectos relacionados ao Gerenciador de Documentos que podem ser incluídos são um controle de

versões mais eficiente e a manutenção de um histórico de execuções, armazenando dados sobre a execução e os dados gerados.

Diversas extensões podem ser consideradas para os modelos de dados adotados. Deve-se considerar, entretanto, que a aceitação do modelo de dados para a Web sempre estará sujeito à aceitação generalizada das propostas de extensão. Com isso, essas propostas devem contar com o aval dos comitês responsáveis pelas especificações. Algumas possíveis extensões a modelos são:

- **Exceções, transações e Eventos.** Inclusão de suporte a transações, tratamento de exceções e de eventos externos (mensagens, alarmes, etc.) nas representações de workflows. Isto possibilita, por exemplo, especificar processos que incluam seqüências indissociáveis de ações, resposta adequada a comportamentos não previstos e uso de sensores como fontes de dados para processos;
- **Semântica.** Inclusão de anotação semântica dos dados que pode ser feita utilizando ontologias de domínio, nos moldes das apresentadas em [Fil03, FLP⁺03] para aspectos geográficos e agrícolas. Outro aspecto é a anotação semântica de serviços, assunto de intensa pesquisa atualmente e com várias questões em aberto. Essas anotações facilitam a um motor de execução de workflows o estabelecimento dinâmico do serviço mais adequado às necessidades de cada atividade. Essas descrições possibilitam a criação de workflows em maior nível de abstração, delegando a escolha dos dados e serviços específicos para o motor de execução, que passa a realizar essa escolha dinamicamente. A utilização direta de metadados (sem utilização de padrões de representação) também auxiliaria, sendo menos abrangente mas mais efetivo;
- **Raias de Execução.** Inclusão de suporte a raias (*swimlanes*) de execução, como nos diagramas de atividade UML, facilitando a separação de nós de execução ou de responsabilidades em um processo, por exemplo;
- **Estruturas Extra.** Inclusão de estruturas não contempladas nos modelos propostos, como, por exemplo, as estruturas *N-de-M*, *Múltiplas instâncias sem sincronização*, *Múltiplas instâncias com número conhecido em tempo de projeto (com sincronização)*, *Múltiplas instâncias com número conhecido em tempo de execução (com sincronização)*, *Ramificação do tipo 'XOR' postergada (ou escolha postergada)*, *Ramificação paralela intercalada*, como especificadas em [WP].

De um ponto de vista conceitual, várias idéias podem ser exploradas. Algumas são listadas a seguir:

- **Componentes de Conteúdo Digital.** Adoção de componentes de conteúdo digital, tais como definidos em [SM04b], expandindo as possibilidades do ponto de vista de modelagem de processos. Os objetos a serem compostos seriam não apenas operações de ferramentas específicas, mas sim componentes completos. Isso facilita a adição de novas funcionalidades (vistas como atividades), e por conseguinte facilita a atividade de especificação de um workflow como um processo. É claro que nesse caso a complexidade seria transferida para a criação dos componentes, mas isso seria feito por alguém especializado nesse aspecto do desenvolvimento. Além disso, como a metodologia de desenvolvimento de workflows proposta se assemelha ao uso desses componentes, essa mudança não seria drástica para o modelo;
- **Perfis de usuário.** Utilização de perfis de usuários, visando tanto aspectos de segurança quanto de modularização do trabalho entre especialistas, atribuindo privilégios e permissões, de forma que um documento possa ter controle de acesso, modificação e execução;
- **Seleção Automatizada de Agentes.** Oferecimento de seleção automatizada de agentes, usando anotações semânticas adequadas. Isso permitiria que tipos de atividades com diferentes níveis de complexidade – de simples, por exemplo, “adição”, a complexos, por exemplo “execução de análise espacial” – possam ser executados automaticamente. Essa idéia se mostrou ineficaz nesse estágio, já que mecanismos de descrição completos o suficiente para permitir uma descrição de tão alto nível ainda não estão disponíveis. Entretanto, existem esforços nessa direção que incluem aspectos de seleção automatizada de agentes como [OWL, W3Ca];
- **Qualidade de Dados e Processos.** Adição de um controle da qualidade dos dados utilizados, campo já bastante desenvolvido, e também da qualidade dos processos. Esse último sendo avaliado dos pontos de vista de especificação, de eficiência na execução e na geração de resultados, por exemplo. Com isso pode-se determinar (talvez até de forma automatizada) qual processo seria mais adequado para uma situação específica.

Referências Bibliográficas

- [ACKM04] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer-Verlag, 2004.
- [All01] R. Allen. *Workflow Handbook 2001*, chapter Workflow: An Introduction. Workflow Management Coalition (WfMC), 2001. http://www.wfmc.org/information/Workflow-An_Introduction.pdf (Acessado em Nov 2004).
- [Apa] Apache Software Foundation. Jakarta Tomcat. <http://jakarta.apache.org/tomcat/> (Acessado em Nov 2004).
- [BPE03] BPEL4WS. Business Process Execution Language for Web Services Version 1.1. Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, 2003. www-106.ibm.com/developerworks/webservices/library/ws-bpel/ (Acessado em Nov 2004).
- [BPMa] BPMI.org. Business Process Management Initiative. <http://www.bpmi.org> (Acessado em Nov 2004).
- [BPMb] BPMI.org. Business Process Modeling Language 1.0. <http://www.bpmi.org/bpml-spec.htm> (Acessado em Nov 2004).
- [BW95] P. Barthelmess and J. Wainer. Workflow systems: a few definitions and a few suggestions. In *Proc. of Conference on Organizational Computing Systems*, 1995.
- [BWM03] M. J. Blin, J. Wainer, and C. B. Medeiros. A reuse-oriented workflow definition language. *International Journal of Cooperative Information Systems*, 12(1), 2003.
- [CCH⁺96] G. Câmara, M. A. Casanova, A. S. Hemerly, G. C. Magalhães, and C. B. Medeiros. *Anatomia de Sistemas de Informação Geográfica*. X Escola

- de Computação. Instituto de Computação – UNICAMP, Campinas – SP, 1996.
- [CCW⁺01] P. H. Camori, J. H. Caviglione, M. S. Wrege, S. L. Gonçalves, R. T. Faria, A. Androcioli Filho, T. Será, J. C. D. Chaves, and M. S. Koguishi. Climatic risk for coffee in Paraná state. *Revista Brasileira de Agrometeorologia*, 9(3), 2001.
- [Claa] Clark Labs. Geographic Analysis and Image Processing Software. www.clarklabs.org (Acessado em Nov 2004).
- [Clab] Clark Labs. Geographic Analysis and Image Processing Software. www.clarklabs.org (Acessado em Nov 2004).
- [CMC⁺02] M. C. R. Cavalcanti, M. L. Q. Mattoso, M. L. M. Campos, F. Llirbat, and E. Simon. Sharing scientific models in environmental applications. In *Proc. of ACM Symposium on Applied Computing*, 2002.
- [CMC03] M. C. R. Cavalcanti, M. L. Q. Mattoso, and M. L. M. Campos. *Scientific Resources Management: Towards An In Silico Laboratory*. PhD thesis, Coppe – UFRJ, Rio de Janeiro–RJ, 2003.
- [Fil03] R. Fileto. *The POESIA Approach for Services and Data Integration On the Semantic Web*. PhD thesis, IC–UNICAMP, Campinas–SP, 2003.
- [FLP⁺03] R. Fileto, L. Liu, C. Pu, E. D. Assad, and C. B. Medeiros. POESIA: An Ontological Workflow Approach for Composing Web Services in Agriculture. *The VLDB Journal*, 12(4):352–367, 2003.
- [Fra00] M. A. R. Franco. *Planejamento Ambiental para a Cidade Sustentável*. Annablume (FAPESP), São Paulo – SP, 2000.
- [GMP96] F. Garzotto, L. Mainetti, and P. Paolini. Information Reuse in Hypermedia Applications. In *Proc. of the the 7th ACM conf. on Hypertext*, 1996.
- [JBo] JBoss Inc. JBoss Application Server. <http://www.jboss.org/> (Acessado em Nov 2004).
- [Kas01] D. S. Kaster. Combinando bancos de dados e raciocínio baseado em casos para apoio à decisão em planejamento ambiental. Master’s thesis, IC–UNICAMP, Campinas–SP, 2001.

- [KMR04] D. Kaster, C. B. Medeiros, and H. Rocha. Supporting Modeling and Problem Solving from Precedent Experiences: The Role of Workflows and Case-Based Reasoning. *Environmental Modeling and Software*, 2004. Aceito para publicação – disponível na revista online.
- [Lim03] J. G. S. Lima. Gerenciamento de dados climatológicos heterogêneos para aplicações em agricultura. Master's thesis, IC–UNICAMP, Campinas–SP, 2003.
- [MA99] C. M. B. Medeiros and A. C. Alencar. Qualidade dos dados e interoperabilidade em sig. In *Proc. I Simpósio Brasileiro de Geoinformática (GEOINFO)*, 1999.
- [MyS] MySQL. MySQL Database Server. <http://www.mysql.com/> (Acessado em Nov 2004).
- [Oasa] Oasis-Open. OASIS Web Services Business Process Execution Language Technical Committee. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (Acessado em Nov 2004).
- [Oasb] Oasis-Open. Organization for the Advancement of Structured Information Standards. www.oasis-open.org (Acessado em Nov 2004).
- [Oasc] Oasis-Open (Organization for the Advancement of Structured Information Standards). WS-BPEL (Main) XML Schema. <http://www.oasis-open.org/committees/download.php/10350> (Acessado em Nov 2004).
- [Oasd] Oasis-Open (Organization for the Advancement of Structured Information Standards). WS-BPEL (plinkType) XML Schema. <http://www.oasis-open.org/committees/download.php/10352> (Acessado em Nov 2004).
- [OMGa] OMG. Object Management Group. www.omg.org (Acessado em Nov 2004).
- [OMGb] OMG. Unified Modeling Language. <http://www.uml.org/> (Acessado em Nov 2004).
- [OMGc] OMG. Unified Modeling Language 2.0 Superstructure Final Adopted specification. <http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.pdf> (Acessado em Nov 2004).

- [Ort84] L. Ortolano. *Environmental Planning and Decision Making*. John Wiley & Sons, EUA, 1984.
- [OWL] OWL-S (The DAML Program). OWL-based Web Service Ontology. <http://www.daml.org/services/owl-s/> (Acessado em Nov 2004).
- [Ple02] C. Plesums. *Workflow Handbook 2002*, chapter An Introduction to Workflow. Workflow Management Coalition (WfMC), 2002. http://www.wfmc.org/information/introduction_to_workflow02.pdf (Acessado em Nov 2004).
- [PMRR05] G. Z. Pastorello Jr., C. B. Medeiros, S. M. Resende, and H. A. Rocha. Interoperability for GIS Document Management in Environmental Planning. *Journal of Data Semantics*, 3(LNCS 3534):100–124, 2005.
- [Pos] PostgreSQL. PostgreSQL Database Management System. <http://www.postgresql.org/> (Acessado em Nov 2004).
- [Res03] S. M. Resende. Database-centered Documentation of Environmental Planning Activities. Master's thesis, IC–UNICAMP, Campinas–SP, 2003.
- [Roc00] A. Rockley. *Fundamental Concepts of Content Reuse*, chapter 2. New Riders, 2000.
- [Roc03] H. A. Rocha. Metadata for Scientific Workflows in Environmental Planning Support. Master's thesis, IC–UNICAMP, Campinas–SP, 2003.
- [SCP97] R. F. Santos, H. B. Carvalhais, and F. Pires. Planejamento Ambiental e Sistemas de Informações Geográficas. *Caderno de Informações Georeferenciadas*, 1(2), 1997. <http://www.cpa.unicamp.br/revista/cigv1n2a2.html> (Acessado em Nov 2004).
- [Sef98] L. Seffino. WOODSS - sistema espacial de apoio ao processo decisório baseado em workflows. Master's thesis, IC–UNICAMP, Campinas–SP, 1998.
- [SM04a] A. Santanchè and C. B. Medeiros. Geographic Digital Content Components. In *Proc. of VI Brazilian Symposium on GeoInformatics*, 2004.
- [SM04b] A. Santanchè and C. B. Medeiros. Managing Dynamic Repositories for Digital Content Components. In *Current Trends in Database Technology – EDBT 2004 Workshops: EDBT 2004 Workshops PhD, DataX, PIM*,

- P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004. Revised Selected Papers*, volume 3268/2004. Springer-Verlag Heidelberg, November 2004.
- [SMRY99] L. Seffino, C. B. Medeiros, J. Rocha, and B. Yi. WOODSS - A Spatial Decision Support System based on Workflows. *Decision Support Systems*, 27(1-2):125-123, 1999.
- [van03] W. M. P. van der Aalst. Patterns and XPD: A Critical Evaluation of the XML Process Definition Language. Technical report, Queensland University of Technology, 2003. QUT FIT-TR-2003-06.
- [VFM03] L. R. Venâncio, R. Fileto, and C. B. Medeiros. Aplicando ontologias de objetos geográficos para facilitar navegação em gis. In *Proc. V Simpósio Brasileiro de Geoinformática (GEOINFO)*, 2003.
- [vt02] W. M. P. van der Aalst and A. H. M. ter Hofstede. Workflow patterns: On the expressive power of (petri-net-based) workflow languages. In *Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, 2002.
- [vtKB00] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Advanced workflow patterns. In *Proc. 7th International Conference on Cooperative Information Systems (CoopIS 2000)*, 2000.
- [W3Ca] W3C. Semantic Web Services Interest Group. www.w3.org/2002/ws/swsig/ (Acessado em Nov 2004).
- [W3Cb] W3C. The World Wide Web Consortium. www.w3.org (Acessado em Nov 2004).
- [W3Cc] W3C. Web Service Choreography Interface (WSCI) 1.0. <http://www.w3.org/TR/wsci/> (Acessado em Nov 2004).
- [W3Cd] W3C. Web Services Activity. <http://www.w3.org/2002/ws/> (Acessado em Nov 2004).
- [W3Ce] W3C. Web Services Choreography Description Language 1.0. <http://www.w3.org/TR/ws-cdl-10/> (Acessado em Nov 2004).
- [WfMa] WfMC. Workflow Management Coalition. <http://www.wfmc.org> (Acessado em Nov 2004).

- [WfMb] WfMC – Workflow Management Coalition. TC-1025 — XPD L XML Schema (XSD). http://www.wfmc.org/standards/docs/TC-1025_schema_10_xpdl.xsd (Acessado em Nov 2004).
- [WfM95] WfMC – Workflow Management Coalition. The Workflow Reference Model. Technical report, Workflow Management Coalition, 1995. TC-1003.
- [WfM98] WfMC – Workflow Management Coalition. Workflow Management Application Programming Interface (Interfaces 2 & 3) Specification. Technical report, Workflow Management Coalition, 1998. TC-1009.
- [WfM02] WfMC – Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language (XPDL). Technical report, Workflow Management Coalition, 2002. TC-1025.
- [WP] Workflow Patterns. tmitwww.tm.tue.nl/research/patterns/ (Acessado em Nov 2004).
- [WvdADtH03] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *Proc. of the 29th EUROMICRO Conf.*, pages 298–305, 2003.
- [WWVM96] J. Wainer, M. Weske, G. Vossen, and C. B. Medeiros. Scientific workflow systems. In *Proc. of the NSF Workshop on Workflow and Process Automation Information Systems*, 1996.

Apêndice A

Script para Criação do Banco de Dados em PostgreSQL

```
CREATE TABLE Role (  
  role_id SERIAL,  
  role_name VARCHAR(80) UNIQUE,  
  role_description VARCHAR(255),  
  PRIMARY KEY (role_id)  
);
```

```
CREATE TABLE Agent (  
  agent_id SERIAL,  
  agent_name VARCHAR(80) UNIQUE,  
  agent_description VARCHAR(255),  
  agent_application VARCHAR(80),  
  agent_application_uri VARCHAR(255),  
  agent_application_call TEXT,  
  PRIMARY KEY (agent_id)  
);
```

```
CREATE TABLE DataType (  
  data_type_id SERIAL,  
  data_type_name VARCHAR(80) UNIQUE,  
  data_type_description VARCHAR(255),  
  data_type_application VARCHAR(255),  
  PRIMARY KEY (data_type_id)  
);
```



```

CREATE TABLE Data (
  data_id SERIAL,
  data_type INTEGER,
  data_uri TEXT,
  data_version VARCHAR(80),
  PRIMARY KEY (data_id),
  FOREIGN KEY (data_type) REFERENCES DataType(data_type_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);

```

```

CREATE TABLE ActivityType (
  activity_type_id SERIAL,
  activity_type_name VARCHAR(80) UNIQUE,
  activity_type_description VARCHAR(255),
  activity_type_operation VARCHAR(80),
  activity_type_application VARCHAR(80),
  PRIMARY KEY (activity_type_id)
);

```

```

CREATE TABLE ParameterSpecifier (
  ps_id SERIAL,
  activity_type INTEGER NOT NULL,
  data_type INTEGER NOT NULL,
  ps_parameter_order INTEGER NOT NULL,
  ps_fixed_parameter INTEGER,
  ps_in_out CHAR(1) CHECK (ps_in_out IN ('i','o')),
  PRIMARY KEY (ps_id),
  FOREIGN KEY (activity_type) REFERENCES ActivityType(activity_type_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (data_type) REFERENCES DataType(data_type_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (ps_fixed_parameter) REFERENCES Data(data_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);

```

```

CREATE TABLE Workflow (
  workflow_id SERIAL,
  workflow_name VARCHAR(80) UNIQUE,
  workflow_description VARCHAR(255),
  workflow_type INTEGER, -- This is an ActivityType
  workflow_creation_date TIMESTAMP WITH TIME ZONE,
  workflow_state CHAR(1) CHECK (workflow_state IN ('c','i')),
  workflow_onto_uri VARCHAR(255),
  workflow_onto_cover TEXT,
  PRIMARY KEY (workflow_id),
  FOREIGN KEY (workflow_type) REFERENCES ActivityType(activity_type_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);

```

```

CREATE TABLE Activity (
  activity_id SERIAL,
  activity_name VARCHAR(80) UNIQUE,
  activity_description VARCHAR(255),
  activity_state CHAR(1) CHECK (activity_state IN ('c','i')),
  activity_workflow INTEGER NOT NULL,
  activity_type INTEGER,
  activity_class VARCHAR(17) CHECK (activity_class IN
    ('WorkflowConnector','BasicActivity','SubWorkflow')),
  activity_sub_workflow INTEGER,
  activity_workflow_connector_type CHAR(1) CHECK
    (activity_workflow_connector_type IN ('i','o')),
  activity_pre_condition VARCHAR(80),
  activity_post_condition VARCHAR(80),
  activity_in_control_mode VARCHAR(3) CHECK
    (activity_in_control_mode IN ('AND','OR','XOR')),
  activity_out_control_mode VARCHAR(3) CHECK
    (activity_out_control_mode IN ('AND','OR','XOR')),
  activity_visual_position_x INTEGER,
  activity_visual_position_y INTEGER,
  activity_onto_cover TEXT,
  PRIMARY KEY (activity_id),
  FOREIGN KEY (activity_workflow) REFERENCES Workflow(workflow_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (activity_type) REFERENCES ActivityType(activity_type_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (activity_sub_workflow) REFERENCES Workflow(workflow_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);

```

```

CREATE TABLE ActivityConnector (
  ac_id SERIAL,
  activity INTEGER NOT NULL,
  parameter_specifier INTEGER NOT NULL,
  data INTEGER,
  ac_connector_control_mode CHAR(1) CHECK
    (ac_connector_control_mode IN ('M','D')),
  PRIMARY KEY (ac_id),
  FOREIGN KEY (activity) REFERENCES Activity(activity_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (parameter_specifier) REFERENCES ParameterSpecifier(ps_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (data) REFERENCES Data(data_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);

```

```

CREATE TABLE Transition (
  transition_id SERIAL,
  transition_origin_ac INTEGER NOT NULL,
  transition_destin_ac INTEGER NOT NULL,
  transition_name VARCHAR(80) UNIQUE,
  transition_activation_condition VARCHAR(80),
  transition_expiration INTEGER,
  transition_origin_position_x INTEGER,
  transition_origin_position_y INTEGER,
  transition_destin_position_x INTEGER,
  transition_destin_position_y INTEGER,
  PRIMARY KEY (transition_id),
  FOREIGN KEY (transition_origin_ac) REFERENCES ActivityConnector(ac_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (transition_destin_ac) REFERENCES ActivityConnector(ac_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);

```

```

CREATE TABLE AgentPlaysRole (
  agent INTEGER,
  role INTEGER,
  PRIMARY KEY (agent,role),
  FOREIGN KEY (agent) REFERENCES Agent(agent_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (role) REFERENCES Role(role_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);

```

```
CREATE TABLE ATypeMayHaveRole (  
  activity_type INTEGER,  
  role INTEGER,  
  PRIMARY KEY (activity_type,role),  
  FOREIGN KEY (activity_type) REFERENCES ActivityType(activity_type_id)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
  FOREIGN KEY (role) REFERENCES Role(role_id)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Apêndice B

Estruturas em XMLSchema WS-BPEL

B.1 XMLSchema para *partnerLinkTypes*

Disponível em (<http://schemas.xmlsoap.org/ws/2003/05/partner-link/>):

```
<?xml version='1.0' encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  targetNamespace="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  elementFormDefault="qualified">

  <element name="partnerLinkType" type="plnk:tPartnerLinkType"/>

  <complexType name="tPartnerLinkType">
    <sequence>
      <element name="role" type="plnk:tRole" minOccurs="1" maxOccurs="2"/>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
  </complexType>

  <complexType name="tRole">
    <sequence>
      <element name="portType" minOccurs="1" maxOccurs="1">
        <complexType>
          <attribute name="name" type="QName" use="required"/>
        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
  </complexType>
</schema>
```

B.2 XMLSchema para WSDL

Disponível em (<http://schemas.xmlsoap.org/wSDL/>):

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
```

Copyright 2001-2003 International Business Machines Corporation,
Microsoft Corporation. All rights reserved.

The presentation, distribution or other dissemination of the information contained herein by Microsoft is not a license, either expressly or impliedly, to any intellectual property owned or controlled by Microsoft.

This document and the information contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, Microsoft provides the document AS IS AND WITH ALL FAULTS, and hereby disclaims all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence, all with regard to the document. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE DOCUMENT.

IN NO EVENT WILL MICROSOFT BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS DOCUMENT, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

```
-->
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
           targetNamespace="http://schemas.xmlsoap.org/wSDL/"
           elementFormDefault="qualified" >

  <xs:complexType mixed="true" name="tDocumentation" >
    <xs:sequence>
      <xs:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="tDocumented" >
    <xs:annotation>
      <xs:documentation>
        This type is extended by component types to allow them to be documented
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="documentation" type="wSDL:tDocumentation"
        minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
```

```

    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="tExtensibleAttributesDocumented" abstract="true" >
    <xs:complexContent>
      <xs:extension base="wsdl:tDocumented" >
        <xs:annotation>
          <xs:documentation>
            This type is extended by component types to allow attributes
            from other namespaces to be added.
          </xs:documentation>
        </xs:annotation>
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="tExtensibleDocumented" abstract="true" >
    <xs:complexContent>
      <xs:extension base="wsdl:tDocumented" >
        <xs:annotation>
          <xs:documentation>
            This type is extended by component types to allow elements
            from other namespaces to be added.
          </xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
            processContents="lax" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="definitions" type="wsdl:tDefinitions" >
    <xs:key name="message" >
      <xs:selector xpath="wsdl:message" />
      <xs:field xpath="@name" />
    </xs:key>
    <xs:key name="portType" >
      <xs:selector xpath="wsdl:portType" />
      <xs:field xpath="@name" />
    </xs:key>
    <xs:key name="binding" >
      <xs:selector xpath="wsdl:binding" />
      <xs:field xpath="@name" />
    </xs:key>
    <xs:key name="service" >
      <xs:selector xpath="wsdl:service" />
      <xs:field xpath="@name" />
    </xs:key>
    <xs:key name="import" >
      <xs:selector xpath="wsdl:import" />
      <xs:field xpath="@namespace" />
    </xs:key>
  </xs:element>

  <xs:group name="anyTopLevelOptionalElement" >

```

```

<xs:annotation>
  <xs:documentation>
    Any top level optional element allowed to appear more then once -
    any child of definitions element except wsdl:types. Any
    extensibility element is allowed in any place.
  </xs:documentation>
</xs:annotation>
<xs:choice>
  <xs:element name="import" type="wsdl:tImport" />
  <xs:element name="types" type="wsdl:tTypes" />
  <xs:element name="message" type="wsdl:tMessage" >
    <xs:unique name="part" >
      <xs:selector xpath="wsdl:part" />
      <xs:field xpath="@name" />
    </xs:unique>
  </xs:element>
  <xs:element name="portType" type="wsdl:tPortType" />
  <xs:element name="binding" type="wsdl:tBinding" />
  <xs:element name="service" type="wsdl:tService" >
    <xs:unique name="port" >
      <xs:selector xpath="wsdl:port" />
      <xs:field xpath="@name" />
    </xs:unique>
  </xs:element>
</xs:choice>
</xs:group>

<xs:complexType name="tDefinitions" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:sequence>
        <xs:group ref="wsdl:anyTopLevelOptionalElement"
          minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="targetNamespace" type="xs:anyURI"
        use="optional" />
      <xs:attribute name="name" type="xs:NCName" use="optional" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tImport" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleAttributesDocumented" >
      <xs:attribute name="namespace" type="xs:anyURI" use="required" />
      <xs:attribute name="location" type="xs:anyURI" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tTypes" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" />
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tMessage" >
  <xs:complexContent>

```



```

    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:sequence>
        <xs:element name="part" type="wsdl:tPart" minOccurs="0"
          maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tPart" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleAttributesDocumented" >
      <xs:attribute name="name" type="xs:NCName" use="required" />
      <xs:attribute name="element" type="xs:QName" use="optional" />
      <xs:attribute name="type" type="xs:QName" use="optional" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tPortType" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleAttributesDocumented" >
      <xs:sequence>
        <xs:element name="operation" type="wsdl:tOperation"
          minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tOperation" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:sequence>
        <xs:choice>
          <xs:group ref="wsdl:request-response-or-one-way-operation" />
          <xs:group ref="wsdl:solicit-response-or-notification-operation" />
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required" />
      <xs:attribute name="parameterOrder" type="xs:NMTOKENS"
        use="optional" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:group name="request-response-or-one-way-operation" >
  <xs:sequence>
    <xs:element name="input" type="wsdl:tParam" />
  </xs:sequence>
  <xs:sequence minOccurs="0" >
    <xs:element name="output" type="wsdl:tParam" />
  </xs:sequence>
  <xs:element name="fault" type="wsdl:tFault" minOccurs="0"
    maxOccurs="unbounded" />
</xs:sequence>
</xs:group>

```

```

<xs:group name="solicit-response-or-notification-operation" >
  <xs:sequence>
    <xs:element name="output" type="wsdl:tParam" />
    <xs:sequence minOccurs="0" >
      <xs:element name="input" type="wsdl:tParam" />
    <xs:element name="fault" type="wsdl:tFault" minOccurs="0"
      maxOccurs="unbounded" />
    </xs:sequence>
  </xs:sequence>
</xs:group>

<xs:complexType name="tParam" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleAttributesDocumented" >
      <xs:attribute name="name" type="xs:NCName" use="optional" />
      <xs:attribute name="message" type="xs:QName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tFault" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleAttributesDocumented" >
      <xs:attribute name="name" type="xs:NCName" use="required" />
      <xs:attribute name="message" type="xs:QName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tBinding" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:sequence>
        <xs:element name="operation" type="wsdl:tBindingOperation"
          minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required" />
      <xs:attribute name="type" type="xs:QName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tBindingOperationMessage" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:attribute name="name" type="xs:NCName" use="optional" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tBindingOperationFault" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:attribute name="name" type="xs:NCName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

<xs:complexType name="tBindingOperation" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:sequence>
        <xs:element name="input" type="wsdl:tBindingOperationMessage"
          minOccurs="0" />
        <xs:element name="output" type="wsdl:tBindingOperationMessage"
          minOccurs="0" />
        <xs:element name="fault" type="wsdl:tBindingOperationFault"
          minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tService" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:sequence>
        <xs:element name="port" type="wsdl:tPort" minOccurs="0"
          maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="tPort" >
  <xs:complexContent>
    <xs:extension base="wsdl:tExtensibleDocumented" >
      <xs:attribute name="name" type="xs:NCName" use="required" />
      <xs:attribute name="binding" type="xs:QName" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:attribute name="arrayType" type="xs:string" />
<xs:attribute name="required" type="xs:boolean" />
<xs:complexType name="tExtensibilityElement" abstract="true" >
  <xs:attribute ref="wsdl:required" use="optional" />
</xs:complexType>

</xs:schema>

```

B.3 XMLSchema para WS-BPEL

Disponível em (<http://schemas.xmlsoap.org/ws/2003/03/business-process/>):

```

<?xml version='1.0' encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  targetNamespace="http://schemas.xmlsoap.org/ws/2003/03/business-process/"

```

```

    elementFormDefault="qualified">

<import namespace="http://schemas.xmlsoap.org/wsdl/"
        schemaLocation="http://schemas.xmlsoap.org/wsdl/" />

<complexType name="tExtensibleElements">
    <annotation>
        <documentation>
            This type is extended by other component types
            to allow elements and attributes from
            other namespaces to be added.
        </documentation>
    </annotation>
    <sequence>
        <any namespace="##other" minOccurs="0" maxOccurs="unbounded"
            processContents="lax" />
    </sequence>
    <anyAttribute namespace="##other" processContents="lax" />

</complexType>

<element name="process" type="bpws:tProcess" />
<complexType name="tProcess">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <sequence>
                <element name="partnerLinks" type="bpws:tPartnerLinks"
                    minOccurs="0" />
                <element name="partners" type="bpws:tPartners"
                    minOccurs="0" />
                <element name="variables"
                    type="bpws:tVariables"
                    minOccurs="0" />
                <element name="correlationSets"
                    type="bpws:tCorrelationSets" minOccurs="0" />
                <element name="faultHandlers" type="bpws:tFaultHandlers"
                    minOccurs="0" />
                <element name="compensationHandler"
                    type="bpws:tCompensationHandler" minOccurs="0" />
                <element name="eventHandlers"
                    type="bpws:tEventHandlers" minOccurs="0" />
                <group ref="bpws:activity" />
            </sequence>
            <attribute name="name" type="NCName"
                use="required" />
            <attribute name="targetNamespace" type="anyURI"
                use="required" />
            <attribute name="queryLanguage" type="anyURI"
                default="http://www.w3.org/TR/1999/REC-xpath-19991116" />
            <attribute name="expressionLanguage" type="anyURI"
                default="http://www.w3.org/TR/1999/REC-xpath-19991116" />
            <attribute name="suppressJoinFailure" type="bpws:tBoolean"
                default="no" />
            <attribute name="enableInstanceCompensation"
                type="bpws:tBoolean" default="no" />
            <attribute name="abstractProcess" type="bpws:tBoolean"
                default="no" />
        </extension>
    </complexContent>
</complexType>

```

```

    </complexContent>
  </complexType>

  <group name="activity">
    <choice>
      <element name="empty" type="bpws:tEmpty"/>
      <element name="invoke" type="bpws:tInvoke"/>
      <element name="receive" type="bpws:tReceive"/>
      <element name="reply" type="bpws:tReply"/>
      <element name="assign" type="bpws:tAssign"/>
      <element name="wait" type="bpws:tWait"/>
      <element name="throw" type="bpws:tThrow"/>
      <element name="terminate" type="bpws:tTerminate"/>
      <element name="flow" type="bpws:tFlow"/>
      <element name="switch" type="bpws:tSwitch"/>
      <element name="while" type="bpws:tWhile"/>
      <element name="sequence" type="bpws:tSequence"/>
      <element name="pick" type="bpws:tPick"/>
      <element name="scope" type="bpws:tScope"/>
    </choice>
  </group>

  <complexType name="tPartnerLinks">
    <complexContent>
      <extension base="bpws:tExtensibleElements">
        <sequence>
          <element name="partnerLink" type="bpws:tPartnerLink"
            minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tPartnerLink">
    <complexContent>
      <extension base="bpws:tExtensibleElements">
        <attribute name="name" type="NCName" use="required"/>
        <attribute name="partnerLinkType" type="QName"
          use="required"/>
        <attribute name="myRole" type="NCName"/>
        <attribute name="partnerRole" type="NCName"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tPartners">
    <complexContent>
      <extension base="bpws:tExtensibleElements">
        <sequence>
          <element name="partner" type="bpws:tPartner"
            minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

```

```

<complexType name="tPartner">
  <complexContent>
    <extension base="bpws:tExtensibleElements">
      <sequence>
        <element name="partnerLink" minOccurs="1"
          maxOccurs="unbounded">
          <complexType>
            <complexContent>
              <extension base="bpws:tExtensibleElements">
                <attribute name="name" type="NCName"
                  use="required"/>
              </extension>
            </complexContent>
          </complexType>
        </element>
      </sequence>
      <attribute name="name" type="NCName" use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="tFaultHandlers">
  <complexContent>
    <extension base="bpws:tExtensibleElements">
      <sequence>
        <element name="catch" type="bpws:tCatch"
          minOccurs="0" maxOccurs="unbounded"/>
        <element name="catchAll"
          type="bpws:tActivityOrCompensateContainer"
          minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="tCatch">
  <complexContent>
    <extension base="bpws:tActivityOrCompensateContainer">
      <attribute name="faultName" type="QName"/>
      <attribute name="faultVariable" type="NCName"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="tActivityContainer">
  <complexContent>
    <extension base="bpws:tExtensibleElements">
      <sequence>
        <group ref="bpws:activity"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="tActivityOrCompensateContainer">
  <complexContent>
    <extension base="bpws:tExtensibleElements">

```

```

        <choice>
            <group ref="bpws:activity"/>
            <element name="compensate" type="bpws:tCompensate"/>
        </choice>
    </extension>
</complexContent>
</complexType>

<complexType name="tEventHandlers">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <sequence>
                <element name="onMessage" type="bpws:tOnMessage"
                    minOccurs="0" maxOccurs="unbounded"/>
                <element name="onAlarm" type="bpws:tOnAlarm"
                    minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tOnMessage">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <sequence>
                <element name="correlations" type="bpws:tCorrelations"
                    minOccurs="0"/>
                <group ref="bpws:activity"/>
            </sequence>
            <attribute name="partnerLink" type="NCName" use="required"/>
            <attribute name="portType" type="QName" use="required"/>
            <attribute name="operation" type="NCName" use="required"/>
            <attribute name="variable" type="NCName"
                use="optional"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tOnAlarm">
    <complexContent>
        <extension base="bpws:tActivityContainer">
            <attribute name="for" type="bpws:tDuration-expr"/>
            <attribute name="until" type="bpws:tDeadline-expr"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tCompensationHandler">
    <complexContent>
        <extension base="bpws:tActivityOrCompensateContainer"/>
    </complexContent>
</complexType>

<complexType name="tVariables">
    <complexContent>
        <extension base="bpws:tExtensibleElements">

```

```

        <sequence>
            <element name="variable"
                type="bpws:tVariable"
                maxOccurs="unbounded"/>
        </sequence>
    </extension>
</complexContent>

</complexType>

<complexType name="tVariable">

    <!-- variable does not allow extensibility elements
    because otherwise its content model would be non-deterministic -->
    <attribute name="name" type="NCName" use="required"/>
    <attribute name="messageType" type="QName" use = "optional"/>
    <attribute name="type" type="QName" use = "optional"/>
    <attribute name="element" type="QName" use = "optional"/>
    <anyAttribute namespace="##other" processContents="lax"/>

</complexType>

<complexType name="tCorrelationSets">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <sequence>
                <element name="correlationSet"
                    type="bpws:tCorrelationSet"
                    maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tCorrelationSet">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <attribute name="properties" use="required">
                <simpleType>
                    <list itemType="QName"/>
                </simpleType>
            </attribute>
            <attribute name="name" type="NCName" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tActivity">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <sequence>
                <element name="target" type="bpws:tTarget"
                    minOccurs="0" maxOccurs="unbounded"/>
                <element name="source" type="bpws:tSource"
                    minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

```



```

        <attribute name="name" type="NCName"/>
        <attribute name="joinCondition"
            type="bpws:tBoolean-expr"/>
        <attribute name="suppressJoinFailure"
            type="bpws:tBoolean" default="no"/>
    </extension>
</complexContent>

</complexType>

<complexType name="tSource">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <attribute name="linkName" type="NCName" use="required"/>
            <attribute name="transitionCondition"
                type="bpws:tBoolean-expr"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tTarget">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <attribute name="linkName" type="NCName" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tEmpty">
    <complexContent>
        <extension base="bpws:tActivity"/>
    </complexContent>
</complexType>

<complexType name="tCorrelations">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <sequence>
                <element name="correlation" type="bpws:tCorrelation"
                    minOccurs="1" maxOccurs="unbounded" />
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tCorrelation">
    <complexContent>
        <extension base="bpws:tExtensibleElements">
            <attribute name="set" type="NCName" use="required"/>
            <attribute name="initiate" type="bpws:tBoolean"
                default="no"/>
        </extension>
    </complexContent>
</complexType>

```

```

<complexType name="tCorrelationsWithPattern">
  <complexContent>
    <extension base="bpws:tExtensibleElements">
      <sequence>
        <element name="correlation"
          type="bpws:tCorrelationWithPattern"
          minOccurs="1"
          maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="tCorrelationWithPattern">
  <complexContent>
    <extension base="bpws:tCorrelation">
      <attribute name="pattern">
        <simpleType>
          <restriction base="string">
            <enumeration value="in" />
            <enumeration value="out" />
            <enumeration value="out-in" />
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </complexContent>
</complexType>

<complexType name="tInvoke">
  <complexContent>
    <extension base="bpws:tActivity">
      <sequence>
        <element name="correlations"
          type="bpws:tCorrelationsWithPattern"
          minOccurs="0" maxOccurs="1"/>
        <element name="catch" type="bpws:tCatch"
          minOccurs="0" maxOccurs="unbounded"/>
        <element name="catchAll"
          type="bpws:tActivityOrCompensateContainer"
          minOccurs="0"/>
        <element name="compensationHandler"
          type="bpws:tCompensationHandler" minOccurs="0"/>
      </sequence>
      <attribute name="partnerLink" type="NCName" use="required"/>
      <attribute name="portType" type="QName" use="required"/>
      <attribute name="operation" type="NCName" use="required"/>
      <attribute name="inputVariable"
        type="NCName" use="optional"/>
      <attribute name="outputVariable" type="NCName"
        use="optional"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="tReceive">

```

```

    <complexContent>
      <extension base="bpws:tActivity">
        <sequence>
          <element name="correlations"
            type="bpws:tCorrelations" minOccurs="0"/>
        </sequence>
        <attribute name="partnerLink" type="NCName" use="required"/>
        <attribute name="portType" type="QName" use="required"/>
        <attribute name="operation" type="NCName" use="required"/>
        <attribute name="variable" type="NCName" use="optional"/>
        <attribute name="createInstance" type="bpws:tBoolean"
          default="no"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tReply">
    <complexContent>
      <extension base="bpws:tActivity">
        <sequence>
          <element name="correlations"
            type="bpws:tCorrelations" minOccurs="0"/>
        </sequence>
        <attribute name="partnerLink" type="NCName" use="required"/>
        <attribute name="portType" type="QName" use="required"/>
        <attribute name="operation" type="NCName" use="required"/>
        <attribute name="variable" type="NCName"
          use="optional"/>
        <attribute name="faultName" type="QName"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tAssign">
    <complexContent>
      <extension base="bpws:tActivity">
        <sequence>
          <element name="copy" type="bpws:tCopy"
            minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tCopy">
    <complexContent>
      <extension base="bpws:tExtensibleElements">
        <sequence>
          <element ref="bpws:from"/>
          <element ref="bpws:to"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <element name="from" type="bpws:tFrom"/>
  <complexType name="tFrom">
    <complexContent>

```

```

        <extension base="bpws:tExtensibleElements">
            <attribute name="variable" type="NCName"/>
            <attribute name="part" type="NCName"/>
            <attribute name="query" type="string"/>
            <attribute name="property" type="QName"/>
            <attribute name="partnerLink" type="NCName"/>
            <attribute name="endpointReference" type="bpws:tRoles"/>
            <attribute name="expression" type="string"/>
            <attribute name="opaque" type="bpws:tBoolean"/>
        </extension>
    </complexContent>
</complexType>

<element name="to">
    <complexType>
        <complexContent>
            <restriction base="bpws:tFrom">
                <attribute name="expression" type="string"
                    use="prohibited"/>
                <attribute name="opaque" type="bpws:tBoolean"
                    use="prohibited"/>
                <attribute name="endpointReference" type="bpws:tRoles"
                    use="prohibited"/>
            </restriction>
        </complexContent>
    </complexType>
</element>

<complexType name="tWait">
    <complexContent>
        <extension base="bpws:tActivity">
            <attribute name="for"
                type="bpws:tDuration-expr"/>
            <attribute name="until"
                type="bpws:tDeadline-expr"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tThrow">
    <complexContent>
        <extension base="bpws:tActivity">
            <attribute name="faultName" type="QName" use="required"/>
            <attribute name="faultVariable" type="NCName"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tCompensate">
    <complexContent>
        <extension base="bpws:tActivity">
            <attribute name="scope" type="NCName"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tTerminate">

```

```

    <complexContent>
      <extension base="bpws:tActivity"/>
    </complexContent>
  </complexType>

  <complexType name="tFlow">
    <complexContent>
      <extension base="bpws:tActivity">
        <sequence>
          <element name="links" type="bpws:tLinks" minOccurs="0"/>
          <group ref="bpws:activity" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tLinks">
    <complexContent>
      <extension base="bpws:tExtensibleElements">
        <sequence>
          <element name="link"
            type="bpws:tLink"
            maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tLink">
    <complexContent>
      <extension base="bpws:tExtensibleElements">
        <attribute name="name" type="NCName" use="required"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="tSwitch">
    <complexContent>
      <extension base="bpws:tActivity">
        <sequence>
          <element name="case" maxOccurs="unbounded">
            <complexType>
              <complexContent>
                <extension base="bpws:tActivityContainer">
                  <attribute name="condition"
                    type="bpws:tBoolean-expr"
                    use="required"/>
                </extension>
              </complexContent>
            </complexType>
          </element>
          <element name="otherwise"
            type="bpws:tActivityContainer"
            minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

```

```

<complexType name="tWhile">
  <complexContent>
    <extension base="bpws:tActivity">
      <sequence>
        <group ref="bpws:activity"/>
      </sequence>
      <attribute name="condition"
        type="bpws:tBoolean-expr"
        use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="tSequence">
  <complexContent>
    <extension base="bpws:tActivity">
      <sequence>
        <group ref="bpws:activity" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="tPick">
  <complexContent>
    <extension base="bpws:tActivity">
      <sequence>
        <element name="onMessage"
          type="bpws:tOnMessage"
          maxOccurs="unbounded"/>
        <element name="onAlarm"
          type="bpws:tOnAlarm" minOccurs="0"
          maxOccurs="unbounded"/>
      </sequence>
      <attribute name="createInstance"
        type="bpws:tBoolean" default="no"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="tScope">
  <complexContent>
    <extension base="bpws:tActivity">
      <sequence>
        <element name="variables"
          type="bpws:tVariables"
          minOccurs="0"/>
        <element name="correlationSets"
          type="bpws:tCorrelationSets"
          minOccurs="0"/>
        <element name="faultHandlers"
          type="bpws:tFaultHandlers"
          minOccurs="0"/>
        <element name="compensationHandler"
          type="bpws:tCompensationHandler"
          minOccurs="0"/>
        <element name="eventHandlers"

```

```

        type="bpws:tEventHandlers"
        minOccurs="0"/>
      <group ref="bpws:activity"/>
    </sequence>
    <attribute name="variableAccessSerializable"
      type="bpws:tBoolean"
      default="no"/>
  </extension>
</complexContent>
</complexType>

<simpleType name="tBoolean-expr">
  <restriction base="string"/>
</simpleType>

<simpleType name="tDuration-expr">
  <restriction base="string"/>
</simpleType>

<simpleType name="tDeadline-expr">
  <restriction base="string"/>
</simpleType>

<simpleType name="tBoolean">
  <restriction base="string">
    <enumeration value="yes"/>
    <enumeration value="no"/>
  </restriction>
</simpleType>

<simpleType name="tRoles">
  <restriction base="string">
    <enumeration value="myRole"/>
    <enumeration value="partnerRole"/>
  </restriction>
</simpleType>
</schema>

```