**Título: Um modelo de definição de processos para sistemas de workflow flexíveis**

*Fábio de Lima Bezerra*

**Dissertação de Mestrado**

Instituto de Computação
Universidade Estadual de Campinas

# Título: Um modelo de definição de processos para sistemas de workflow flexíveis

**Fábio de Lima Bezerra**[1]

04 de Julho de 2003

**Banca Examinadora:**

- Prof. Dr. Jacques Wainer
  Instituto de Computação - UNICAMP (Orientador)
- Prof. Dr. Sofiane Labidi
  Centro de Tecnologia - UFMA
- Profa. Dra. Maria Beatriz Felgar de Toledo
  Instituto de Computação - UNICAMP
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação - UNICAMP (Suplente)

[1]**Trabalho com suporte da CAPES.**

**Substitua pela ficha catalográfica**

# Título: Um modelo de definição de processos para sistemas de workflow flexíveis

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fábio de Lima Bezerra e aprovada pela Banca Examinadora.

Campinas, 31 de Julho de 2003.

Prof. Dr. Jacques Wainer
Instituto de Computação - UNICAMP
(Orientador)

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

**Substitua pela folha com a assinatura da banca**

# Resumo

Em domínios de aplicação como a área médica e engenharia de software os sistemas de workflow existentes são inadequados, pois a definição dos processos nesses sistemas é inflexível. No exemplo da área médica, ninguém sabe quais as ações devem ser tomadas durante a abordagem de um paciente, ou seja, ninguém conhece um processo de tratamento para paciente no instante em que ele é admitido no hospital. Portanto, uma nova forma de interação com os sistemas de workflow se faz necessário. Nesta nova forma de interação um usuário pergunta ao sistema o que deve ser feito para que uma atividade (atividade objetivo) seja executada. Por exemplo, para um médico realizar uma cirurgia em um paciente que procedimentos devem ser executados antes ou mesmo depois? Nesta nova abordagem, o sistema de workflow cumpre, além do papel de despachante de atividades aos usuários, como nos sistemas de workflow atuais, o papel de ajudante do usuário, pois indica ao usuário quais atividades executar antes e depois de executar uma atividade objetivo.

Este trabalho apresenta um novo modelo de definição de processos em sistemas de workflow, o modelo PDBC. Apresentamos também um protótipo de um sistema de workflow que implementa o modelo PDBC, o servidor *Tucupi*. A proposta deste novo modelo é permitir que os sistemas de workflow manipulem workflows mais flexíveis. Para isso usaremos o que chamamos de *workflow parcial*, que é um workflow parcialmente definido cuja definição completa é realizada apenas durante a execução do workflow. O workflow parcial é definido como um conjunto de restrições e atividades.

Outra maneira de se obter flexibilidade em sistemas de workflow é através de um mecanismo de tratamento de exceções adequado. Neste trabalho propomos um mecanismo de tratamento de exceções através do uso de restrições violáveis, ou seja, restrições que podem ser violadas em determinadas circunstâncias. Para isso apresentamos o WRBAC, um modelo de controle de acesso para sistemas de workflow, utilizado para provê um mecanismo adequado de sobrecarga das restrições utilizadas na definição de um processo.

**Palavras-chave:** *sistemas de workflow, workflows flexíveis, tratamento de exceções, workflows baseados em regras, restrições violáveis*

# Abstract

This work presents a new model of process definition in workflow systems, the PDBC model. We also present a prototype of a workflow system which implements the PDBC model, the Tucupi server. The proposal of this new model is to allow flexibility in workflow systems. For this, we will use what we call partial workflow, which is a partially defined workflow whose complete definition is carried through during the execution of workflow. Partial Workflow is defined as a set of constraints and activities. Moreover, another way for a workflow system to get flexibility is through an exception handling mechanism. In this work we consider such mechanism through the use of overriden constraints, that is, constraints that can be violated in determined circumstances. For this we present the WRBAC, a model of access control for workflow systems, used to provide an adequate mechanism of violation of the overriden constraints used in the definition of a workflow.

**Keywords:** *workflow systems, flexible workflows, exception handling, rule based workflow and overriden constraints*

# Agradecimentos

Agradeço a Deus, nosso criador, por apresentar-me os caminhos que fizeram com que eu conseguisse concluir o mestrado. Além disso, agradeço a Deus pelo alimento espiritual que me proporcionou força para superar as dificuldades que encontrei durante o curso.

Agradeço aos meus pais, Claudionor e Aldenora, que me proporcionaram a formação cultural inicial e construíram os alicerces de minha personalidade moral e bons costumes. Agradecimento especial ao meu pai pelo financiamento de parte de meus estudos. A minha mãe, agradecimento especial pelo estímulo, amor e carinho.

Agradeço aos meus irmãos, Eduardo e Diego, pelo amor e carinho demonstrados quando chegava em Belém após muitos meses sem vê-los.

Agradeço aos meus avós maternos, Lourival (*in memoriam*) e Maria (*in memoriam*), e aos meus avós paternos, Francisco (*in memoriam*) e Elza.

Agradeço a família de minha mãe pela torcida em meus empreendimentos: tia Socorro e família, tia Elma e família, tia Leila e família, tia Zeca e família, tio Dori e família, tia Tetê (*in memoriam*) e família e tia Osmarina e família. Do mesmo modo a família de meu pai: tia Elza e família, tia Ilza e família, tio Pedro e família, tia Suely e família, tia Beth e família e tio Chico (*in memoriam*) e família.

Agradeço a minha noiva Erika, que por algum tempo morou próximo a Campinas e pôde fornecer-me amor e carinho. Entretanto, mesmo durante o tempo que passou distante de mim, conseguia dar-me força para concluir o curso. Agradeço também aos pais de minha noiva, Edimilson e Marilene.

Agradeço a avó de minha noiva, dona Carmita, pela torcida e muitas orações dedicadas a mim.

Agradeço ao meu orientador, o professor Jacques Wainer, pela orientação do trabalho e amizade. As dicas obtidas me proporcionaram maior amadurecimento no trabalho científico, que servirão para futuras contribuições à ciência.

Agradeço aos meus professores da graduação na UFPA: Cleidson, Elói, Rodrigo, Carla, Francisco Edson, Bechara, Mário Tavares, Aruanda, etc. Em especial ao Cleidson, o orientador de meu trabalho de conclusão de curso, que me apresentou o trabalho científico.

Agradeço a todos os meus amigos de trabalho da UNISYS, SYSDATA e da

ECOFLORESTAL.

Agradeço as meus amigos em Belém, que para não correr o risco de cometer a falta de esquecer algum nome, não indicarei nenhum nome especial.

Agradeço aos meus amigos de república, Charles e Daniel, oportunamente referenciados como "mulheques", pela excelente convivência e pela amizade que construímos. Por extensão, agradeço as amizades que formei com as amizades anexadas a Charles e Daniel.

Agradeço aos amigos da UNICAMP, que involuntariamente contribuíram para amenizar momentos de saudade. Dos muitos amigos que fiz não poderia deixar de fazer menção aos amigos do Pará, Fernando, Luciana e Schubert, que também fizeram a graduação comigo. Aos meus irmãos (de orientador) Alexandre, Cléo, Gregório, Márcio, Wanderley. Aos que ingressaram comigo no mestrado Adriane, Carlos, Daniele, Douglas, Lásaro, Tarcísio e Vanessa. A Amanda, Eduardo (Pará), Fábio, Fernando, Flavão, Flavinho, Glauber, Guido (Power), Guilherme, Gustavo (Baiano), Henrique, Leizza, Luiz, Marília, Nielsen, Rogério(Zafa e Messias), Silvana, Silvania e Zé, pela boa convivência e amizade.

Agradeço a CAPES pelo financiamento do curso.

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

A concepção da tecnologia de workflow teve como principal propósito o apoio à coordenação. Os sistemas de informação baseados em processos não ofereciam um mecanismo de controle das atividades que deveriam ser executadas, dos usuários que estão executando estas atividades e dos dados que estas atividades manipulam. Assim, o apoio limitado à coordenação, ou mesmo a falta de um mecanismo de coordenação desses sistemas, permitiu a rápida aceitação da tecnologia de workflow.

Um workflow é um conjunto de uma ou mais atividades ou procedimentos que juntas realizam um objetivo. Enquanto que um sistema de workflow (ou sistema de gerência de workflow) é um software que coordena a execução das atividades de um workflow. Os sistemas de workflow podem ser vistos como uma composição de dois componentes: *(i)* um componente de definição do workflow, onde o usuário define as atividades e ordem de execução destas atividades; *(ii)* um componente de execução, responsável pela execução e coordenação das atividades do workflow. Na execução de uma instância de workflow uma série de atividades é executada por pessoas ou automaticamente, de acordo com a definição do processo.

Em alguns domínios, a definição do processo é criada antes de seu uso, embora a maioria das definições utilize formalismos que permitem alguma flexibilidade, como por exemplo a execução condicional de caminhos diferentes (pré-definidos) baseados na escolha do usuário ou em alguns dos dados da instância do workflow. Neste caso, dizemos que o workflow é flexível porque uma única definição do workflow permite caminhos distintos de execução, apesar de previamente definidos.

Em domínios como a engenharia de software e medicina o projetista não conhece a estrutura (as atividades e a ordem de execução das atividades) de execução do processo antes de sua execução. Portanto existe a necessidade de se implementar nesses domínios uma forma alternativa de flexibilidade. Para isso usaremos o que chamamos de *workflow parcial*, que é um workflow condicional e parcialmente definido. Assim, consideraremos a

flexibilidade como a capacidade de um workflow ser dinamicamente definido.

Um workflow parcial é formado pelas atividades que podem ser executadas nesse workflow e pelas restrições entre as atividades desse workflow. Os workflows parciais podem incluir em sua definição atividades que devem ser executadas, mas admitem uma execução desordenada. Por exemplo, a construção de vários componentes independentes de um grande projeto de software. Outra característica dos workflows parciais é a existência de atividades cuja execução é condicionada à execução de outra(s) atividade(s) antes. Por exemplo, caso uma interveção cirúrgica seja requisitada o paciente deve assinar um termo de consentimento de transfusão de sangue no caso de haver a necessidade.

Apresentaremos neste trabalho uma abordagem nova de definir/modelar workflows. Nesta nova abordagem os workflows são definidos como um conjunto de restrições. O uso das restrições permite que o projetista do workflow não precise se preocupar com a definição das transições entre as atividades, permitindo, desta forma, um maior desacoplamento entre definição e execução de um workflow. Acreditamos que tal desacomplemento permite uma execução mais flexível do workflow. Um workflow também atende os requisitos de flexibilidade quando dispõe de um mecanismo de tratamento de exceções, pois acrescenta, na definição do workflow, caminhos alternativos de execução que são percorridos em situações excepcionais. Para atender este requisito de flexibilidade usaremos o conceito de *restrições violáveis*, ou seja, restrições que podem ser violadas em condições excepcionais.

O desenvolvimento deste trabalho proporcionou a publicação de um artigo no $9^o$ CRIWG, *International Workshop on Groupware*, que será realizado no mês de setembro deste ano, em Grenoble, França. Além desse trabalho, foram submetidos dois outros artigos que ainda aguardamos resposta.

Esta dissertação encontra-se estruturada como segue: no Capítulo 2 apresentaremos os conceitos iniciais e os termos fundamentais usados na dissertação; no Capítulo 3 apresentaremos o modelo WRBAC, um modelo de controle de acesso para sistemas de workflow; no Capítulo 4 apresentaremos o modelo PDBC, que é o modelo que contempla nossa proposta de definição de workflows baseados em restrições; no Capítulo 5 apresentaremos o Servidor Tucupi, um protótipo do modelo PDBC; por fim, no Capítulo 6 apresentaremos algumas conclusões sobre nosso trabalho, além de algumas sugestões para futuros trabalhos.

# Capítulo 2

# Primeiros Conceitos

Este capítulo apresenta os conceitos fundamentais para o entendimento da dissertação. Entretanto, o conteúdo deste capítulo não possui a pretensão de esgotar discussões sobre os conceitos, que podem ser explorados nas referências citadas, assim como em outras fontes.

A Seção 2.1 apresenta os sistemas de workflow, principal objeto de pesquisa desta dissertação. Uma definição dos sistemas de *workflow* é apresentada na Seção 2.1.2. A Seção 2.1.4 apresenta o modelo de referência arquitetural dos sistemas de workflow proposto pela WfMC. A Seção 2.2 apresenta o modelo de controle de acesso RBAC, que serviu de referência para extensão do modelo WRBAC. A Seção 2.2.2 apresenta uma família de modelos RBAC proposta em [24]. A Seção 2.2.1 apresenta uma descrição breve dos elementos que norteiam a definição do modelo RBAC. A Seção 2.2.3 compara o RBAC com as listas de controle de acesso (abreviação do inglês, ACL) e com o modelo mandatário de controle de acesso (abreviação do inglês, MAC).

## 2.1 Sistemas de Workflow

### 2.1.1 Apresentando o termo workflow

O termo workflow muitas vezes é confundido como um sistema computacional, software ou aplicação, mas apesar de ser utilizado com este sentido, o termo workflow diz respeito a um processo que suporta automação computacional. Em [32], workflow é a automação de um processo de negócios, no todo ou parte, durante o qual documentos, informação e tarefas são passados de um participante para outro por ações de acordo com um conjunto de regras procedimentais.

Um processo de negócio é um conjunto de uma ou mais atividades ou procedimentos que juntas realizam um objetivo de negócio ou objetivo político [32]. Um workflow

representa a parte do processo que pode ser automatizada. Entretanto, se considerarmos que todos os processos de negócios podem ser integralmente automatizados, os dois conceitos podem ser utilizados como sinônimos.

Também é possível apresentar uma definição para workflow como sendo um grafo orientado. Os vértices representam as atividades do processo e as arestas representam transições entre as atividades. A orientação das arestas indica a atividade origem e o destino da transição.

O desenvolvimento da tecnologia de workflow teve o propósito de minimizar os problemas relacionados à coordenação de processos [4]. Através do workflow é possível determinar os usuários que estão executando as atividades do processo, os dados e informações que estão sendo manipulados pelo processo e o tempo de execução das atividades ou do processo. Com estas informações um gerente do negócio pode identificar gargalos na execução do processo, podendo modificá-lo, ou seja, o gerente pode realizar a reegenharia do processo do negócio.

### 2.1.2 Definição

O sistema de workflow é um sistema que define, cria e gerencia a execução de workflows através do uso de software, executando sobre um ou mais *workflow engine*, o qual é capaz de interpretar a definição do processo, interagir com participantes do workflow e, quando solicitado, invocar o uso de ferramentas e aplicações [32].

A definição do workflow representa a descrição do conjunto de atividades que fazem parte do processo, além da relação de ordem entre as atividades. Os dados que serão manipulados pelo processo também fazem parte da definição do workflow. A função de definição do workflow faz parte do componente de modelagem do sistema de workflow.

A criação e gerência de execução do workflow é função do componente de execução do sistema de workflow. A criação representa uma instanciação de um workflow. Por exemplo, considere o processo de compra de material em uma indústria. Cada ordem de compra emitida pelo setor de compras da indústria representa uma instância do processo de compra.

O gerenciamento do workflow em execução é a característica principal dos sistemas de workflow. O gerente do processo coordena os workflows através de informações como: tempo de execução de uma atividade, usuário responsável pela execução de uma atividade, tempo de execução de um processo, usuários que participaram da execução de um processo, processos atrasados, entre outras. A identificação de gargalos do processo pode sugerir a reegenharia do mesmo.

### 2.1.3 Classificação

Os sistemas de workflow podem ser classificados em dois tipos: produção e *ad hoc*. Os workflows de produção são caracterizados pela habilidade de gerenciar processos complexos que possuem uma estrutura fixa de execução das tarefas e um conjunto de regras de roteamento entre as tarefas. Os workflows *ad hoc* não possuem uma estrutura fixa de execução, ou seja, a estrutura de execução pode ser modificada durante a execução do workflow.

Exemplo de processos que caracterizam o uso de sistemas de workflow de produção são: procedimento de empréstimo, reclamatório de seguros e aprovação de ordem de compras.

Neste trabalho estamos interessado em estudar os sistemas de workflow ad hoc, por possuirem em sua definição a característica de flexibilidade na execução dos workflows. Como exemplo de processos que caracterizam o uso de sistemas de workflow *ad hoc*, temos: desenvolvimento de software, programas de qualidade e atendimento hospitalar.

### 2.1.4 Arquitetura

A arquitetura de um sistema representa a descrição dos componentes do sistema, bem como a relação existente entre esses componentes. Os sistemas de workflow possuem dois componentes essenciais: componente de modelagem do processo e componente de execução do processo. A Figura 2.1, adaptada de [25], ilustra o modelo de referência proposto pela WFMC (*Workflow Management Coalition*). Neste modelo estão representados os principais componentes de um sistema de workflow e seus respectivos relacionamentos.

#### Componente de modelagem do workflow

O componente de modelagem do workflow, representado na Figura 2.1 como *definição de processo*, possui as funcionalidades para a criação do mesmo, que consiste em:

- definir as atividades;
- definir a ordem parcial entre as atividades;
- definir critérios de transição entre as atividades;
- definir os dados e informações que serão manipulados pelas atividades;
- definir os papéis e estrutura organizacional da instituição;
- associar os usuários aos papéis, atribuindo responsabilidades; e

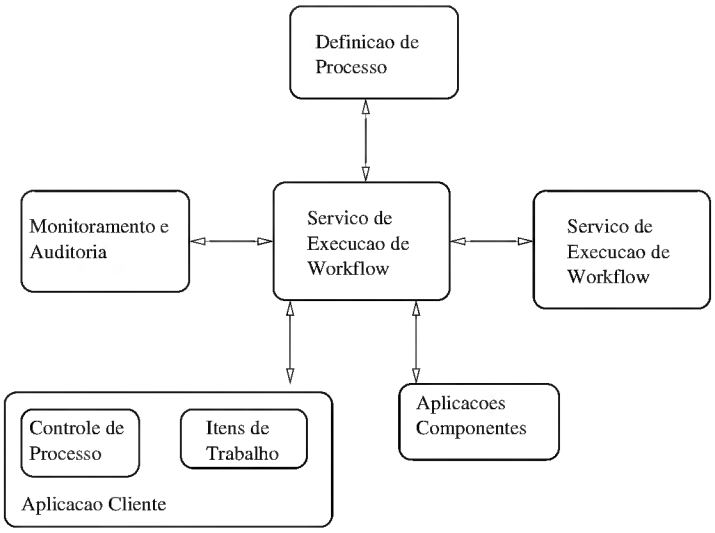


Figura 2.1: Arquitetura de um sistema de workflow.

- associar aos papéis as atividades, ou seja, definição das atribuições.

Em [7] o componente de definição do workflow é estruturado em componentes menores, cujas funcionalidades foram listadas acima. Os componentes menores são: *i)* modelagem do processo; *ii)* modelagem dos dados; *iii)* modelagem organizacional e *iv)* modelagem das aplicações de interação com o usuário.

### Componente de execução do workflow

A execução do processo diz respeito à instanciação do workflow definido no componente de modelagem do workflow. A partir da instância do workflow, o componente de execução coordena o fluxo de execução das atividades do workflow. A coordenação diz respeito à instanciação das atividades do processo, obediente às restrições de ordem e tempo, e gerenciamento do andamento do processo.

A Figura 2.2 ilustra de forma simplificada a interação dos dois principais componentes de um sistema de workflow. O componente de modelagem do workflow ilustra um workflow $A$ com cinco atividades, enquanto o componente de execução do workflow ilustra diferentes instâncias do workflow A. Cada instância do workflow $A$ se encontra em um estado diferente de execução. As atividades em execução estão destacadas com preenchimento cinza.

A atividade de gerenciamento do workflow pode ser considerada a principal motivação da adoção dos sistemas de workflow nas organizações. É através do gerenciamento do

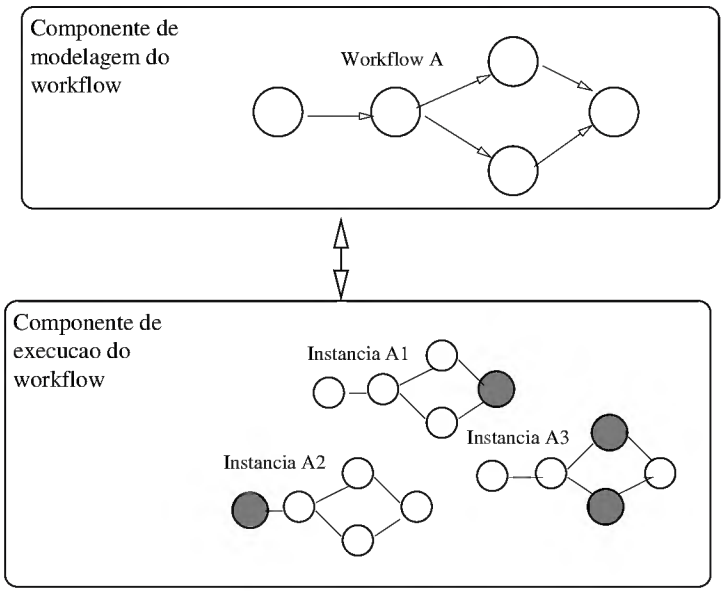


Figura 2.2: Relação dos componentes de modelagem e execução.

workflow que o usuário pode:

- identificar o estado da execução do workflow;
- identificar o tempo de execução das atividades e do workflow, podendo redefinir estes tempos;
- identificar os principais pontos de atraso da execução do workflow;
- medir a carga produtiva dos usuários participantes da execução do workflow;
- identificar as causas da improdutividade de um usuário participante da execução do processo, que pode ser ocasionada por falhas no escalonamento das atividades; e
- prever o tempo restante para conclusão do workflow.

## 2.2 RBAC: Controle de Acesso Baseado em Papéis

No RBAC (*Role Based Access Control*) um conjunto de privilégios é associado a um papel. A atribuição de um papel a um usuário, confere a este usuário todos os privilégios associados a este papel. Desta forma, usando o mecanismo de indireção dos papéis, o RBAC facilita a administração da segurança. Além disso, a associação de usuários aos

papéis tipicamente requer menos conhecimento técnico que a associação de permissões a papéis [24].

Muitas organizações realizam o controle de acesso baseado nos papéis que os usuários destas organizações desempenham [24]. Assim a mudança de função do usuário na organização não requerirá uma nova definição dos privilégios do usuário, pois ao associar o usuário ao conjunto dos novos papéis desempenhados na organização, o usuário adquire os privilégios definidos para estes papéis.

### 2.2.1 Definição

O RBAC [24, 19, 1] propõe que sistemas de controle de acesso possuam um nível de indireção entre usuários e permissões (ou privilégios), o **papel**. A idéia é que a relação entre papéis e privilégios é muito estável, e que os exemplos de mutabilidade de um sistema de permissão estão usualmente concentrados na relação entre usuários e papéis. Assim, quando um novo funcionário é contratado, não é preciso listar todas as suas novas permissões, mas apenas liga-lo aos papéis para os quais ele foi contratado.

Um papel pode representar competência para fazer algumas tarefas, ou pode incorporar autoridade e responsabilidade [24]. Por exemplo, um usuário pode ser competente para liderar vários departamentos, mas é associado para liderar apenas um deles. Os papéis podem ser confundidos com os grupos, mas os papéis além de representarem um conjunto de usuários, representam um conjunto de privilégios.

Além do papel, o RBAC possui as entidades *usuário* e *privilégio*. A atribuição de papéis aos usuários define uma relação entre as entidades papel e usuário. Nesta relação um usuário pode possuir vários papéis e um papel pode ser atribuído a mais de um usuário.

### 2.2.2 Modelo

Sandhu et al. [24] apresentam quatro modelos de referência para controle de acesso baseado em papéis: $RBAC_0$, modelo base, conforme ilustrado na Figura 2.3; $RBAC_1$; $RBAC_2$ e $RBAC_3$, modelo que agrega as propriedades dos modelos anteriores. A divisão em quatro modelos tem o propósito de servir como referência para comparações com outros modelos de controle de acesso, ou então, como guia de implementação de aplicações que apresentam o RBAC como modelo de controle de acesso.

O modelo base, $RBAC_0$, serve de referência para construção dos demais modelos e é compostos por três entidades: usuário, papel e privilégio. O *usuário* representa uma pessoa ou um grupo de pessoas. O *papel* pode representar uma função, cargo, nível de autoridade ou nível de competência. Normalmente o significado do papel é definido pela aplicação. O *privilégio* pode representar uma operação sobre um objeto do sistema ou uma ação no sistema. Novamente, como no caso do papel, a escolha depende da aplicação.

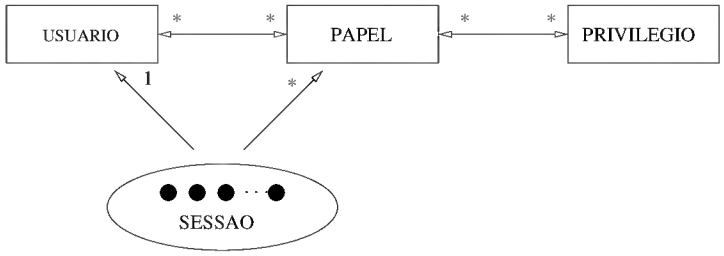


Figura 2.3: Modelo RBAC base.

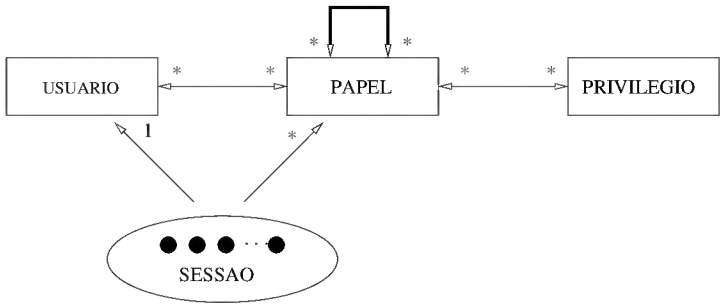


Figura 2.4: Modelo RBAC com suporte à herança de papel.

A Figura 2.3 mostra as entidades e os relacionamentos existentes entre elas. É possível observar na figura que o usuário não possui acesso direto aos privilégios. Para tanto, é preciso associar um privilégio $P$ a um papel $R$, relação muitos-para-muitos, e então associar um usuário $U$ ao papel $R$, relação muitos-para-muitos.

A *sessão*, representada na figura, é um mapeamento de um usuário para possivelmente muitos papéis, isto é, um usuário estabelece uma sessão durante a qual o usuário ativa um subconjunto de papéis que ele ou ela é membro.

Os modelos $RBAC_1$ e $RBAC_2$ são uma extensão do modelo base $RBAC_0$. O modelo $RBAC_2$ não é uma extensão do modelo $RBAC_1$, apesar das notas numéricas usadas nos nomes do modelo. O modelo $RBAC_1$ adiciona ao modelo base a relação hierárquica entre dois papéis, como pode ser observado na relação em destaque na Figura 2.4. Uma restrição intrínseca do modelo RBAC define a hierarquia de papel como uma ordem parcial.

O modelo $RBAC_2$ adiciona ao modelo base o conceito de restrição (Figura 2.5). As permissões no modelo RBAC são sempre positivas, ou seja, permitem ao usuário detentor da permissão o direito de executar alguma ação no sistema. A restrição é utilizada para representar uma permissão negativa, ou seja, a proibição de uma execução. As restrições são um importante aspecto do modelo RBAC, algumas vezes consideradas motivação do

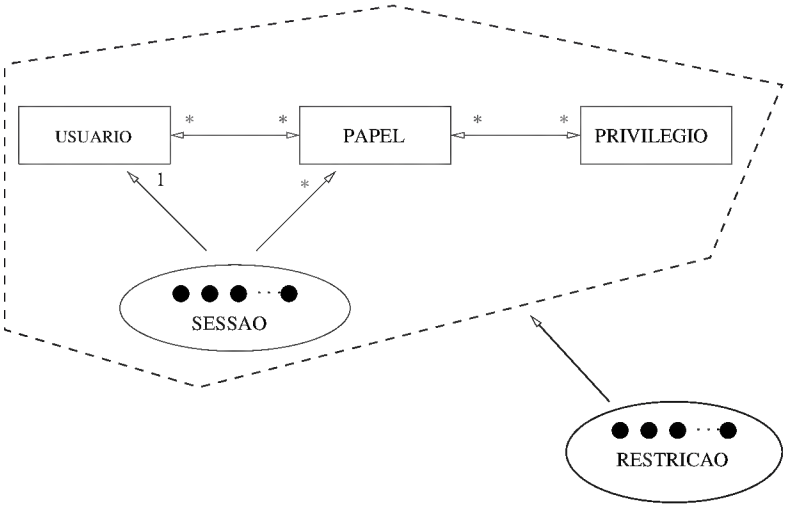


Figura 2.5: Modelo RBAC com suporte à definição de restrições.

RBAC.

A restrição pode referir-se às entidades usuário, papel e privilégio, assim como às relações entre as entidades. Uma restrição também pode ser aplicada a uma sessão, assim um usuário teria acesso restrito a um subconjunto mínimo de papéis que lhe conferisse os privilégios necessários à execução de uma atividade.

A Figura 2.6 ilustra o modelo $RBAC_3$. Este modelo representa a união dos modelos $RBAC_1$ e $RBAC_2$, suportando tanto a funcionalidade de hierarquia de papéis como a funcionalidade de definição de restrições. Neste modelo as restrições podem ser aplicadas também às hierarquias de papéis.

### 2.2.3 Comparações com outros modelos

Esta subseção apresenta uma breve descrição de dois outros modelos de controle de acesso: ACL (*Access Control List*) e MAC (*Mandatory Access Control*). A partir da descrição destes dois modelos será possível fazer algumas comparações com o modelo RBAC.

#### MAC - Controle de acesso mandatário

O controle de acesso mandatário baseia-se na idéia de classificar hierarquicamente os usuários e os objetos do sistema. Os requisitos de organizações militares são naturalmente atendidos com o uso desse modelo de controle de acesso.

Osborn [20] cita a seguinte definição para o controle de acesso mandatário:

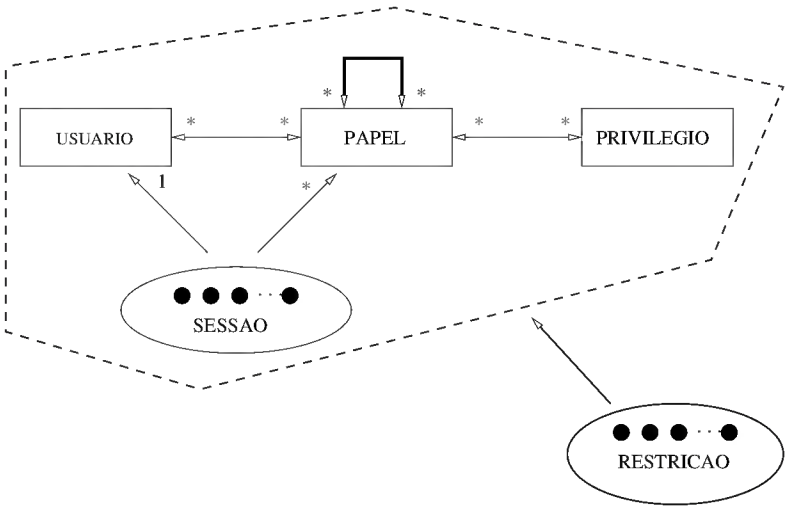


Figura 2.6: Modelo RBAC consolidado: restrições e hierarquia de papéis.

No controle de acesso mandatário, existe um conjunto de *classificações* (por exemplo, altamente secreto, secreto, confidencial, classificado e não classificado) que é totalmente ordenada. Além disso existe um conjunto de *categorias* que não é ordenada. A combinação de uma classificação com um subconjunto das categorias é chamado de *nível de segurança.*

É preciso atribuir um nível de segurança a cada usuário (sujeito) e objeto do sistema. Neste modelo temos duas regras mandatárias a serem seguidas:

- Propriedade da segurança simples. Um usuário tem a permissão de ler um objeto somente se o seu nível de segurança é igual ou superior ao do objeto.

- Propriedade $*$ (Estrela): Um usuário não confiável pode escrever em um objeto somente se o seu nível de segurança for menor ou igual ao do objeto.

### ACL - Listas de controle de acesso

As listas de controle de acesso são um mecanismo de definição de políticas de acesso a objetos pertencentes a um sistema. ACL é a sigla que designa tais lista, cujo significado em inglês é *Access Control List.*

Barkley [6] mostra que as funcionalidades do modelo de controle de acesso baseado em papéis são comparáveis às listas de controle de acesso. Entretanto, é importante destacar que o modelo RBAC comparado em [6] é mais simples que o modelo base proposto por Sandhu et al. [24].

Mecanismos de controle de acesso requerem que atributos de segurança dos usuários e dos objetos sejam guardados. Os atributos de segurança dos usuários consistem em coisas como os grupos dos quais o usuário pertence e os papéis autorizados para o usuário. Os atributos dos objetos geralmente consistem nas permissões necessárias para executar operações sobre o objeto [6].

ACLs tipicamente associam um objeto com uma lista de usuário e grupos. Um conjunto de operações que podem ser executadas sobre um objeto é associado a cada usuário ou grupo em uma ACL. Uma operação sobre um objeto pode ser executada por um usuário se este usuário ou grupo que este usuário pertence é listado na ACL associada com o objeto e esta operação está associada com o usuário ou o grupo [6].

No mecanismo de ACL, se um usuário é membro de um grupo, então este usuário é membro deste grupo em todas as sessões estabelecidas. Em outras palavras, um usuário é um membro de um grupo em todos os momentos e em todas as circunstâncias. No modelo $RBAC_0$ [24], cada sessão pode ser associada com um conjunto diferente de papéis ativos.

A capacidade de um papel herdar outro papel é uma característica comum dos modelos RBAC, por exemplo, $RBAC_1$ [24]. O conceito de herança de papéis pode ser implementado de forma semelhante com as ACLs quando um grupo é membro de outro grupo.

O conceito de restrição, como proposto no modelo $RBAC_2$ [24], não é definido no mecanismo ACL. As restrições podem ser consideradas como característica diferencial dos modelos de segurança RBAC [6].

# Capítulo 3

# Modelos WRBAC

O WRBAC é um modelo de controle de acesso para sistemas de workflow. Este modelo representa uma extensão do modelo RBAC apresentado no capítulo 2. As seções a seguir descreverão o par de modelos WRBAC. O primeiro desses modelos é o $W_0$RBAC que une o serviço de permissões do modelo RBAC com o componente de execução do workflow. O segundo modelo é o $W_1$RBAC, uma extensão do $W_0$RBAC, que incorpora a capacidade de tratar situações excepcionais através do conceito de sobrecarga de restrições.

Os dois modelos apresentam um serviço de permissão baseado em uma linguagem lógica. A partir dessa linguagem é possível selecionar os usuários autorizados a executar as tarefas do workflow seguindo uma ordenação pré-determinada.

O conteúdo deste capítulo é uma compilação do relatório técnico [29] e do trabalho [30], assim estaremos evitando a repetição de referências ao longo do texto. Este capítulo é organizado como segue: na Seção 3.1 apresentaremos a motivação para o uso de um mecanismo de controle de acesso específico para sistemas de workflow; nas Seções 3.2 e 3.3 apresentaremos os modelos $W_0$RBAC e $W_1$RBAC respectivamente; por fim, na Seção 3.4 apresentaremos uma proposta de implementação dos modelos WRBAC.

## 3.1 Introdução

Os sistemas de workflow permitem a definição e a execução de processos. A definição dos processos (ou workflow) possui em seu conteúdo as tarefas que compõem o processo, os usuários que executarão as tarefas e a estrutura das tarefas, ou seja, uma ordem parcial de execução das tarefas.

Tipicamente os workflows possuem tanto tarefas que requerem a execução por pessoas, como tarefas automatizadas. A partir da definição do workflow, o sistema de workflow cria ou instancia os workflows, comumente chamados de *caso* na literatura. Em um dado momento, uma ou mais instâncias de um mesmo workflow podem estar em execução.

Assim como os workflows, as tarefas que compõem o workflow são instanciadas. As instâncias das tarefas são criadas quando suas pré-condições são satisfeitas. Para as tarefas que exigem a intervenção de uma pessoa, o sistema de workflow seleciona um usuário para manipular a tarefa instanciada. Os modelos que serão apresentados neste capítulo são responsáveis pela seleção dos usuários que executarão uma tarefa do workflow.

Uma das principais responsabilidades dos sistemas de workflow é determinar quem, entre os usuários do sistema, é mais apropriado para executar uma dada atividade. Normalmente os sistemas de workflow utilizam um mecanismo simples de seleção dos usuários, por exemplo, através da associação de papéis aos usuário e papéis às tarefas do workflow. Neste caso, para um usuário executar uma tarefa, ele deve possuir o mesmo papel definido para a tarefa. Nestes sistemas de workflow a seleção dos usuários consiste em identificar o papel associado à tarefa que foi instanciada e adicionar tal tarefa na lista de trabalho dos usuários que possuem o papel identificado, então quando um dentre esses usuários assume a responsabilidade de execução da tarefa, esta tarefa é retirada das listas de trabalho dos outros usuários selecionados. Este mecanismo apesar de simples, não leva em consideração restrições como a separação de responsabilidades (do inglês, *separation of duties*) e a ligação de responsabilidades (do inglês, *biding of duties*).

A separação de responsabilidades é uma técnica que tenta minimizar a fraude através da distribuição de responsabilidades entre vários usuários para ações ou tarefas relacionadas. A ligação de responsabilidades é o inverso, neste caso é requerido que um mesmo usuário execute as ações ou tarefas relacionadas.

Considere o exemplo de reembolso apresentado na Figura 3.1. O processo de reembolso inicia-se com a tarefa de requisição do pedido de reembolso. Após o pedido de requisição o processo é encaminhado para auditoria do pedido. Após a execução da tarefa de auditoria o pedido é submetido a duas tarefas de aprovação, que dependendo dos resultados gerados nas tarefas de aprovação, o pedido pode ser encaminhado para a tarefa de rejeição, que justifica a negação do pedido de reembolso, ou para a tarefa de aprovação, que informa o usuário da aprovação do pedido. Além da definição das atividades e estrutura de execução destas atividades, o workflow pode incluir os seguintes requisitos:

1. A tarefa de requisição pode ser executada por qualquer usuário;

2. A tarefa de auditoria deve ser executada apenas por usuários que possuam o papel de auditor, contanto que este usuário não seja o requisitante do pedido de reembolso. Além disso, é preferível que tanto o requisitante quanto o auditor sejam membros da mesma unidade organizacional;

3. A primeira aprovação do pedido de reembolso deve ser feita pelo líder da unidade organizacional do requisitante e este líder não pode ser o requisitante;

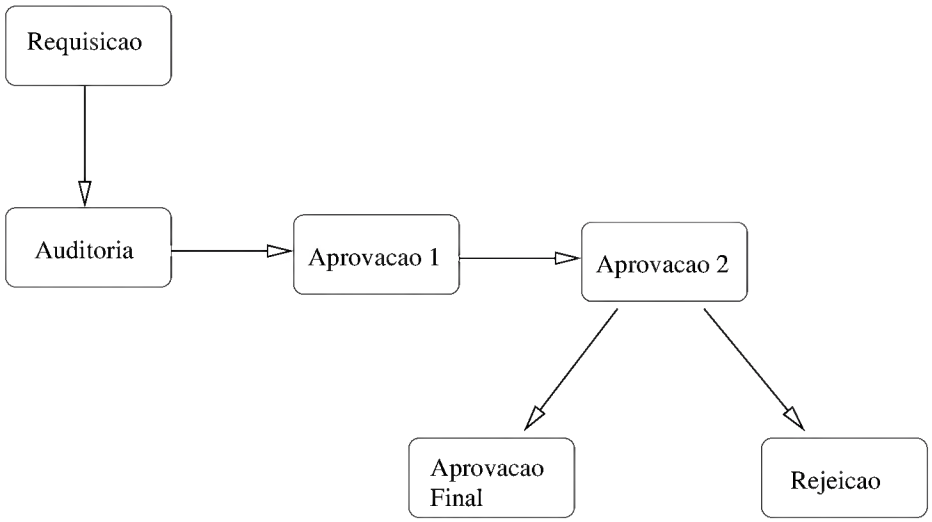


Figura 3.1: Exemplo de processo de reembolso.

4. A segunda aprovação do pedido deve ser feita por um usuário diferente do usuário da primeira aprovação e por um usuário diferente do usuário requisitante, além disso o usuário deve estar no mesmo nível hierárquico ou em um nível hierárquico acima.

A simples associação de papéis usada pelos sistemas de workflow não é expressiva o suficiente para atender os requisitos enunciados no exemplo acima. Portanto, faz-se necessário apresentar um modelo de controle de acesso que apresente um mecanismo de seleção de usuários, tanto em termos de segurança como em termos de expressividade.

## 3.2 $W_0$RBAC

O modelo $W_0$RBAC adiciona ao modelo RBAC as seguintes entidades: **caso**, que representa uma instância de workflow, e **unidade organizacional**, que representa um departamento, setor, instituto ou qualquer entidade de uma organização que é formada por pessoas, por exemplo, um projeto. Além dessas entidades, o modelo $W_0$RBAC restringe o significado da entidade privilégio, que no caso de um workflow representa o direito de executar uma atividade.

A figura 3.2 apresenta o modelo $W_0$RBAC através de um diagrama entidade/relacionamento. Através desta figura podemos observar as extensões realizadas sobre o modelo RBAC.

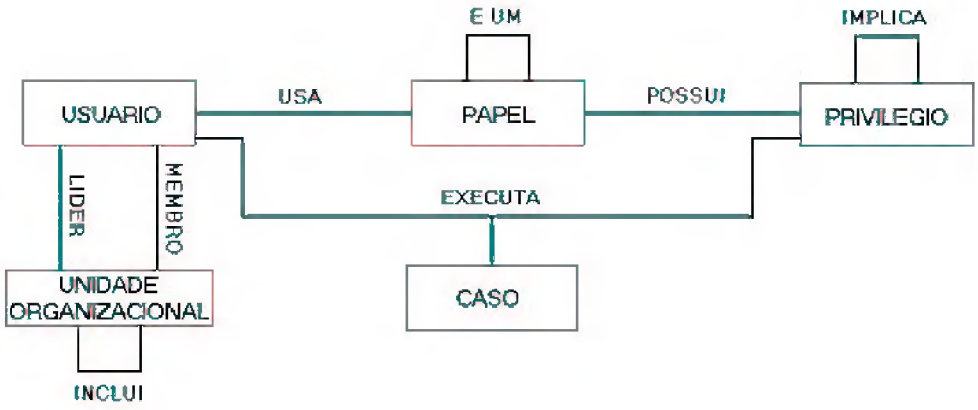


Figura 3.2: Modelo $W_0$RBAC.

### 3.2.1 Entidade *caso*

As restrições estáticas usadas no modelo RBAC são limitadas. Por exemplo, uma restrição estática pode proibir um usuário de possuir os papéis de piloto e navegador de um avião, entretanto isto não é desejado, pois um piloto pode precisar ser um navegador em um vôo e ser um piloto em outro vôo. Portanto, o que se deseja proibir é que a mesma pessoa seja um piloto e um navegador em um mesmo vôo. Para dar suporte a situações como essa, o RBAC implementa esta restrição no contexto de uma sessão, neste caso, um usuário não assumirá os papéis de piloto e navegador na mesma sessão, ou seja, a restrição é aplicada durante o intervalo de tempo da sessão.

Nos sistemas de workflow a execução de um processo não está associada a uma sessão, ou seja, o tempo de execução de um processo pode se estender entre várias sessões. De uma outra maneira, em uma mesma sessão vários processos podem ser executados. Como exemplo considere o caso de um reembolso, onde a política interna de uma organização pode proibir um usuário de REQUISITAR e APROVAR um mesmo pedido. Usando restrições estáticas o usuário nunca poderia fazer as duas tarefas, mesmo que fossem de pedidos distintos. Neste exemplo, o conceito de sessão também não se aplica, pois a política poderia ser violada. O usuário iniciaria uma sessão com o direito de REQUISITAR e mais tarde, numa outra sessão, poderia assumir o direito de APROVAR. Assim o usuário poderia assumir o direito de REQUISITAR e APROVAR o próprio pedido de reembolso.

A entidade *caso*, adicionada no modelo $W_0$RBAC, resolve o problema apresentado no exemplo acima, pois permite além do uso de restrições estáticas a definição de restrições dinâmicas, que são restrições que incluem em sua definição uma referência a uma instância do workflow. Para o exemplo do reembolso, a definição da restrição incluiria uma

referência a um caso, como segue:

$$\begin{aligned} \perp \leftarrow\ & doer(U1, requisitar, C1)\ and \\ & doer(U2, aprovar, C2)\ and \\ & U1 = U2\ and \\ & C1 = C2 \end{aligned}$$

A função *doer* apresentada acima indica que um usuário executou uma atividade em um caso. A Seção 3.2.3 explicará melhor esta função.

### 3.2.2 Entidade *unidade organizacional*

Nas organizações o conceito de hierarquia de papéis e de pessoas é bastante claro. As pessoas são posicionadas em uma ou mais unidades tais como, departamentos, divisões ou grupos. Tais pessoas podem ter diferentes chefes em diferentes níveis organizacionais. Assim a organização pode assumir a política de proibir que a aprovação de um pedido de reembolso seja feita por outra pessoa que não seja o líder da unidade organizacional do usuário que realizou o pedido.

O modelo $W_0$RBAC inclui o conceito de *unidade organizacional* bem como a relação de inclusão que pode existir entre diferentes unidades organizacionais. Por exemplo, uma universidade representa uma unidade organizacional, que inclui outras unidades organizacionais, ou seja, os centros (centro de exatas, centro de saúde, centro de engenharia, etc.). Por conseguinte os centros incluem outras unidades organizacionais, os departamentos. A figura 3.3 ilustra a relação de inclusão entre unidades organizacionais. Os pontos indicados na figura representam o papel do usuário que é líder na unidade organizacional.

A unidade organizacional possibilita a representação da relação de poder entre duas pessoas da organização. Por exemplo, o líder de uma unidade possui um poder maior que uma pessoa que é apenas membro da mesma unidade ou membro das unidades membro. No exemplo da figura 3.3, o usuário no papel de reitor é chefe de todos na universidade, mas um usuário que é apenas membro direto da universidade, ou seja, não é membro de nenhuma unidade membro da universidade (centro ou departamento), não pode ser considerado chefe dos usuários que são membros das unidades membro. De forma contrária, o usuário que é diretor não pode ser considerado chefe dos usuários que são membros direto da universidade.

Entretanto, é importante discutir as diferenças entre a hierarquia de papéis, definida no modelo RBAC, e a hierarquia de unidades organizacionais, do modelo $W_0$RBAC, quanto à definição da relação de poder. A relação *é um*, existente entre papéis, é uma relação de herança, útil para propósitos administrativos, algumas vezes confundida com

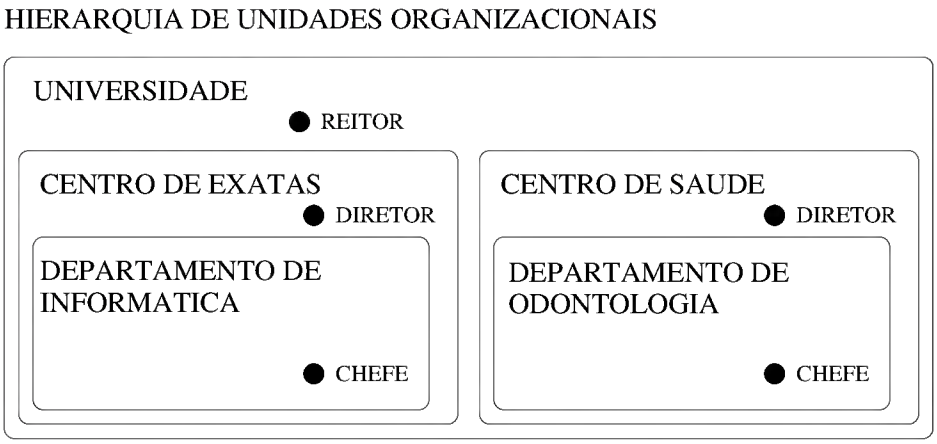


Figura 3.3: Exemplo de hierarquia organizacional.

a relação de poder. Por exemplo, quando representamos que um *analista de sistemas* é um *programador*, dizemos que o analista de sistemas possui inclusive os privilégios de programador, mas não indica que o analista de sistemas é o chefe do programador. A relação *é um* é confundida com a relação de poder nos casos em que o chefe acumula responsabilidades ou privilégios de seus subordinados. Entretanto isto não é comum, por exemplo, o diretor de um hospital embora chefe de todos os médicos, não possui necessariamente o direito de operar um paciente.

### 3.2.3 Relações do modelo

O modelo $W_0$RBAC define além das entidades, um conjunto de relacionamentos entre as entidades que são úteis nas definições das restrições e políticas da organização. A seguir tais relações são listadas:

**include(UO1, UO2).** Representa a relação hierárquica existente entre duas unidades organizacionais. UO1 representa a unidade organizacional que inclui a unidade organizacional UO2.

**member(U, UO).** Representa a relação existente entre as entidades usuário e unidade organizacional. Indica que o usuário U pertence à unidade organizacional UO.

**head(U, UO).** Representa também uma relação existente entre as entidades usuário e unidade organizacional. Indica poder, pois o usuário U representa o líder ou chefe da unidade organizacional UO.

**can_play(U, R).** Representa a relação existente entre as entidades usuário e papel. Indica que o usuário U tem o papel R na organização.

**hold(R, P).** Relação existente entre as entidades privilégio e papel. Indica que o papel R tem privilégio de execução P.

**is_a(R1, R2).** Relação de herança existente entre dois papeis. Indica que o papel R1 é um papel R2, ou seja, R1 possui o conjunto de privilégios definidos para si e mais os privilégios definidos em R2.

**imply(P1, P2).** Relação existente entre dois privilégios. Indica uma relação de inclusão, tal que ter o privilégio P1 implica ter o privilégio P2.

**doer(U, P, C).** Relação ternária existente entre as entidades usuário, privilégio e caso. Indica que o usuário U fez uso do privilégio P no caso C. Através desta relação é possível definir as políticas e restrições dinâmicas.

### 3.2.4 Funções do modelo

Apresentamos abaixo uma lista com duas funções derivadas a partir das relações e entidades do modelo. O desenvolvimento dessas funções permite uma definição mais simples das restrições e políticas da organização.

**boss(U1, U2).** Função que indica a relação de poder existente entre dois usuários. O usuário U1 é chefe do usuário U2, quando o usuário U1 é líder de uma unidade organizacional UO e o usuário U2 é membro da mesma unidade organizacional UO ou de qualquer unidade organizacional que UO inclui.

**same_level(U1, U2).** Função que indica se dois usuários encontram-se no mesmo nível hierárquico. O usuário U1 encontra-se no mesmo nível do usuário U2 se ambos possuem um mesmo chefe X e um mesmo número de chefes intermediários entre o chefe X e cada usuário.

### 3.2.5 Restrições estáticas e dinâmicas

As restrições no $W_0$RBAC são proposições que indicam uma limitação ou proibição. As restrições são escritas numa linguagem lógica e são falsas. Uma restrição é verdadeira quando a sua definição não é respeitada. Por exemplo, considere a seguinte restrição:

$$\bot \leftarrow hold(P1, requisitar)\ and\ hold(P2, aprovar)\ and\ P1 = P2.$$

Tal proposição indica que é uma restrição o mesmo papel executar as tarefas de requisitar e aprovar. A restrição acima é falsa até o momento que tentarem atribuir ao mesmo papel, o direito de execução das tarefas requisitar e aprovar.

No modelo $W_0$RBAC as restrições podem ser estáticas ou dinâmicas. A restrição estática representa uma construção sobre qualquer entidade do modelo $W_0$RBAC, exceto a entidade CASO. A execução de uma instância de workflow não precisa existir para as restrições estáticas serem aplicadas, por isso a entidade CASO não faz parte das construções dessas restrições.

As restrições dinâmicas são aplicadas em uma instância de workflow. Tais restrições são chamadas de dinâmicas pois dependem da dinâmica de execução de uma instância de workflow. Portanto, diferente das restrições estáticas, as construções das restrições dinâmicas fazem referência à entidade CASO. Abaixo apresentamos dois exemplos de restrições que ilustram a diferença existente entre as restrições estáticas e as restrições dinâmicas.

$$\begin{aligned} \bot_{estatica} \leftarrow\ & hold(P1, contratar)\ and \\ & hold(P2, definir\_salario)\ and \\ & P1 = P2. \end{aligned}$$

$$\begin{aligned} \bot_{dinamica} \leftarrow\ & doer(U, contratar, C1)\ and \\ & doer(U, definir\_salario, C2)\ and \\ & C1 = C2. \end{aligned}$$

A primeira restrição proíbe que um mesmo papel assuma os privilégios de contratar e definir o salário, enquanto a segunda restrição proíbe que em um mesmo caso o usuário possa contratar e definir o salário. A segunda restrição possui, claramente, mais flexibilidade, pois permite que um usuário possa tanto contratar quanto definir o salário, mas tais tarefas devem ser executadas em casos diferentes.

As restrições estáticas proíbem a introdução de relações indesejáveis entre as entidades usuário, papel, privilégio e unidade organizacional. Tais restrições são chamadas de estáticas pois não dependem de nenhuma execução de workflow. O controle da segurança independe de qualquer comportamento dinâmico. Existem duas grandes razões para o uso das restrições estáticas: prevenir o agrupamento de usuários incompatíveis e prevenir que os usuários acumulem muito poder, ou seja, assumam o direito de execução de privilégios incompatíveis. Assim um usuário estaria proibido de adquirir uma combinação de privilégios que poderiam provocar algum dano no sistema.

As restrições dinâmicas, do contrário, dependem da execução de um workflow. As restrições dinâmicas podem bloquear a execução de duas ações no sistema ou requerer

suas execuções. Este comportamento dependerá das ações anteriores executadas no caso. As restrições dinâmicas admitem:

**Separação dinâmica de responsabilidades.** Esta ação previne que um usuário execute uma ação conflitante com uma que ele já executou no mesmo caso.

**Ligação dinâmica de responsabilidades.** Esta ação, contrária da anterior, requer que o usuário execute uma ação no futuro por ter executado uma ação relacionada no passado. A razão para tal é que, por ter executado a primeira ação, o usuário adquiriu conhecimento para execução de alguma atividade no futuro.

## 3.3 $W_1$RBAC

$W_1$RBAC estende o modelo $W_0$RBAC através da inclusão de um mecanismo de sobrecarga de restrições, que é realizado de forma controlada. A execução de um workflow é sujeita a desvios do fluxo de execução como definido no modelo. Isto é um problema bastante conhecido na literatura de workflow como *exceções*. O modelo $W_1$RBAC tem a capacidade de tratar tais exceções de uma forma controlada e sistemática. Este mecanismo associa um nível de prioridade para cada restrição e define um privilégio que viole tal restrição.

### 3.3.1 Sobrecarga de restrições

A sobrecarga de restrições é um mecanismo, definido no modelo $W_1$RBAC, que permite a violação de uma restrição. Esta propriedade de violação de restrição é importante, pois claramente algumas restrições são mais importantes que outras. Portanto, em certas situações, pode ser aceitável violar restrições menos importantes.

Normalmente, as restrições dinâmicas que expressam ligação de responsabilidades são menos importantes que as restrições dinâmicas que expressam separação de responsabilidades. Por exemplo, a política de uma organização pode requerer que as tarefas de receber uma pergunta de um cliente e responder a pergunta sejam realizadas pela mesma pessoa, proporcionando ao cliente um sentimento de resposta personalizada (exemplo de ligação de responsabilidades). Considere o usuário *João* recebendo uma pergunta do cliente *Alimentos Ltda.* O usuário *João* inicia a preparação da resposta do cliente para no dia seguinte entregá-la ao cliente, mas excepcionalmente fica doente no dia em que a resposta deveria ser entregue. Neste caso, um outro usuário é alocado para entregar a resposta à *Alimentos Ltda*, pois neste caso é melhor que o cliente não tenha o sentimento de resposta personalizada, ao invés de ter uma impressão de ineficiência na produção da resposta.

As restrições dinâmicas que expressam separação de responsabilidades implementam políticas de segurança mais severas. Por exemplo, considere novamente o exemplo do processo de reembolso. Neste exemplo, o usuário que requisita não pode ser o mesmo que aprova o pedido de reembolso. Portanto, considerando este exemplo de separação de responsabilidades, quando esta política poderá ser violada? Isto mostra claramente que umas restrições são mais importantes que outras.

A literatura de workflow admite que em situações reais a especificação de workflow precise ser violada de forma a permitir que o workflow seja executado [29, 30]. Tal ação é conhecida como tratamento de exceção. Por exemplo, considere um workflow de pedido de empréstimo de uma instituição bancária. Um cliente muito importante pode pedir mais agilidade na obtenção da resposta de seu pedido de empréstimo, assim o workflow para este caso pode suprimir a realização de algumas atividades ou permitir que usuários não autorizados para executar determinadas tarefas ganhem a permissão.

Existem outras formas de tratar uma situação excepcional em um workflow. Umas das formas que não foi apresentada ainda refere-se à mudança na ordem de execução das tarefas do workflow. Entretanto, qualquer alteração da estrutura do workflow, seja com a mudança de ordem de tarefas, seja com a supressão de tarefas, referem-se à definição do workflow quanto à execução das tarefas.

O modelo $W_1$RBAC suporta o mecanismo de tratamento de exceções no contexto da seleção dos usuários para execução de uma tarefa. Assim, em uma situação excepcional um usuário, antes proibido de executar uma tarefa do workflow, pode ser selecionado para executar tal tarefa.

Um usuário do workflow pode ser proibido de executar uma tarefa por dois motivos: *(i)* o usuário não possui o papel que lhe concede o privilégio de executar a tarefa; *(ii)* o usuário possui o papel que tem o privilégio de execução da tarefa, mas uma restrição dinâmica o proíbe de executar a tarefa para o caso em execução. A sobrecarga de restrição ocorre no segundo caso.

### 3.3.2 Níveis de importância das restrições

Para indicar um nível de importância às restrições o modelo $W_1$RBAC define que a cada restrição um valor numérico deve ser associado. Este valor numérico indica que quanto maior for o valor associado à restrição, mais importante é a restrição. Abaixo apresentamos um exemplo de uma restrição que possui nível de importância cinco:

$$\begin{aligned} \bot_5 \leftarrow\ & doer(U, contratar, C1)\ and \\ & doer(U, definir\_salario, C2)\ and \\ & C1 = C2. \end{aligned}$$

A sobrecarga de uma restrição é implementada como um privilégio, ou seja, para um usuário violar uma restrição ele deve pertencer a um papel que possui tal privilégio. Assim como a importância das restrições é indicada por um valor, o privilégio de violação de uma restrição também é indicado por um valor. Este valor numérico indica que quanto maior for o valor indicado pelo privilégio, maior o poder do papel em violar restrições. O privilégio que indica o poder de sobrecarga de uma restrição é mostrado abaixo:

$$override(N).$$

N é um valor numérico que indica o poder de sobrecarga.

## 3.4 Uma proposta de implementação

Esta seção foi escrita com o intuito de apresentar uma implementação para os modelos citados acima. Os requisitos utilizados na implementação foram do modelo $W_1$RBAC. Como o modelo $W_1$RBAC é uma extensão do modelo $W_0$RBAC estaremos atendendo também os requisitos do modelo $W_0$RBAC.

A implementação dos modelos WRBAC foi realizada em *Prolog* por acreditarmos que a definição das restrições, principal característica do modelo, pudesse ser feita de forma mais simples. Além disso em prolog o *backtrack* é realizado de forma automatizada, excluindo o desenvolvedor da tarefa de pesquisar a existência de restrições para um workflow em execução.

A implementação foi realizada com o propósito de servir como um componente ou biblioteca reutilizável, assim qualquer sistema de workflow implementado em Prolog poderia utilizar os mecanismos de controle de acesso do modelo WRBAC. Para exportar as funcionalidades deste componente para outros sistemas não implementados em Prolog, propomos a utilização do mecanismo de comunicação em sockets.

### 3.4.1 Interface pública

A implementação do modelo WRBAC vista como um componente reutilizável possui como interface pública os seguintes serviços:

**who(O, T, C).** Esta função é executado por um sistema de workflow que recebe como resposta uma lista com os usuários habilitados para executar a tarefa indicada pela variável T na instância de workflow indicada pela variável C. A ordem de preferência na seleção dos usuários é indicada pela variável O.

**doer(U, T, C).** Esta função registra no componente WRBAC que o usuário indicado pela variável U executou a tarefa indicada pela variável T na instância de workflow indicada pela variável C.

**done(C).** Esta função registra no componente WRBAC que a instância de workflow indicada pela variável C foi concluída.

**override(Ct, C).** Para indicar que uma restrição foi sobrecarregada, ou seja violada, o sistema de workflow utiliza esta função. A variável Ct indica a restrição, que possui um identificador único para cada instância de workflow. A variável C indica a instância de workflow.

Um usuário U é considerado hábil para executar uma tarefa T quando ele possui um papel R que tem o direito de executar a tarefa T, ou seja $can\_play(U, R)$ e $hold(R, T)$, e a adição da relação $doer(U, P, C)$ não deriva a existência de uma inconsistência entre as declarações das restrições.

A sobrecarga de uma restrição pode deixar a definição do workflow inconsistente. Para evitar isso, adicionamos a cada restrição do workflow a seguinte cláusula:

$$not\ override(Ct, C)$$

Assim, teríamos como exemplo de definição de uma restrição:

$$\bot \leftarrow not\ override(Ct, C)\ and\ doer(U, contratar, C1)\ and\ doer(U, definir\_salario, C2)\ and\ C1 = C2.$$

Portanto, uma restrição é verdadeira se ainda não foi violada e o restante de sua definição é verdadeiro.

### 3.4.2 Ordenação da resposta da função *who*

Uma característica importante dos modelos WRBAC é a de ordenar os usuários que podem executar uma tarefa. Esta característica é importante porque o número de usuários listados para execução de uma atividade pode ser muito alto.

Uma prática comum em sistemas de workflow baseados em RBAC é a de indicar o papel menos específico para executar uma tarefa. Por exemplo, considere dois usuários U1 e U2, sendo o usuário U1 com o papel R1 e o usuário U2 com o papel R2. O papel R2 é um papel R1, ou seja existe uma relação de herança entre os papeis R1 e R2. Supondo que o privilégio de executar a tarefa T pertence aos dois papéis, o usuário U1 tem preferência na indicação para executar a tarefa T por possuir um papel menos específico que o usuário U2.

Em particular nosso modelo sugere três formas de ordenação:

- Entre dois papéis, que pode ser maior (mais específico) ou menor (mais geral). Neste caso, prefere-se o usuário que possui o menor papel, ou seja, o papel mais geral.

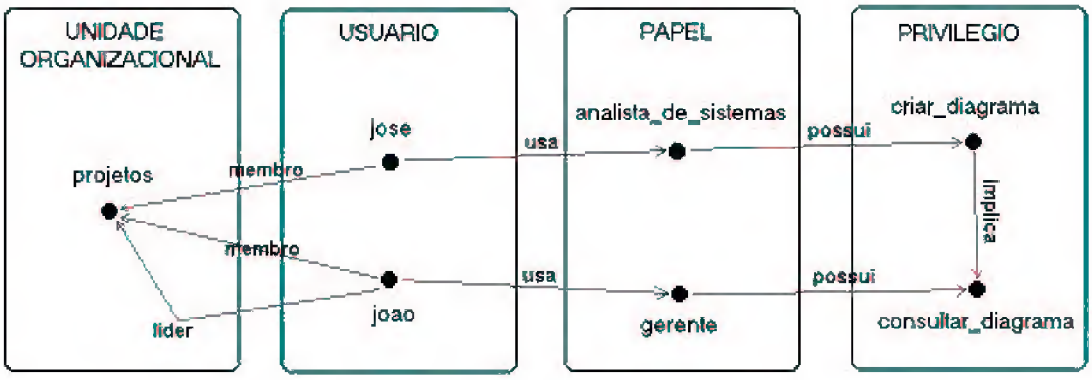


Figura 3.4: Exemplo de relação entre as entidades dos modelos WRBAC.

- Entre dois privilégios, que pode ser mais forte ou mais fraco. Neste caso, prefere-se o usuário que possui o papel que tem o direito de execução do privilégio mais fraco.

- Entre dois usuários, que pode ser chefe ou subordinado. Neste caso, utiliza-se a função *boss* entre os usuários e prefere-se o usuário que é um subordinado.

Para as relações mostradas na figura 3.4, suponhamos que o sistema de workflow deseja encontrar os usuários que podem executar a tarefa *consultar diagrama*, teríamos as seguintes preferências entre os usuários:

- Sobre o esquema de preferência entre dois papéis, não existe diferença entre os usuários *José* e *João*, pois não existe relação entre os papéis executados por estes usuários. O papel *analista de sistemas* não **é um** *gerente*.

- Sobre o esquema de preferência entre dois privilégios, o usuário *João* tem preferência na seleção para execução da tarefa, pois *José* pode executar a tarefa *criar diagrama* que implica no direito de execução da tarefa *consultar diagrama*, portanto a tarefa *consultar diagrama* é mais fraca que a tarefa *criar diagrama*.

- Sobre o esquema de preferência entre dois usuários, ou seja, sobre a relação de poder entre dois usuários, o usuário *José* tem preferência na seleção para execução da tarefa, pois *João* é o líder da unidade organizacional *projetos* e o usuário *José* é apenas membro da unidade organizacional *projetos*. Portanto, por ser um subordinado do usuário *João*, *José* tem preferência na seleção para executar a tarefa.

### 3.4.3 Arquitetura

A implementação do componente WRBAC foi dividida em três estruturas: hierarquia organizacional (papéis, usuários e privilégios); restrições de execução; e implementação dos métodos de manipulação das restrições e hierarquia organizacional (Funções do WRBAC). Cada estrutura representa um arquivo manipulado pelo interpretador prolog. A Figura 3.5 ilustra como estas estruturas estão relacionadas. Esta figura também ilustra com que estrutura do componente WRBAC uma aplicação cliente se relaciona.

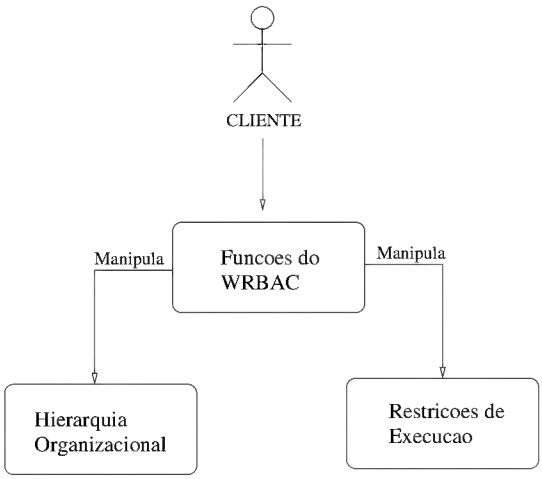


Figura 3.5: Arquitetura do componente WRBAC.

# Capítulo 4

# PDBC: um modelo para workflows flexíveis

Apresentaremos neste capítulo uma nova proposta de definição de processos em sistemas de workflow, o modelo PDBC (***P**rocess **D**efinition **B**ased on **C**onstraints*). Acreditamos que os workflows definidos através do uso das restrições são mais flexíveis, pois não definem uma estrutura rígida de execução.

A flexibilidade dos workflows definidos com as restrições é alcançada de duas maneiras. A primeira maneira é através da escolha mais livre da estrutura de execução, ou seja, o usuário determina qual atividade executar e quando, desde que obedeça às restrições do workflow. A segunda maneira é através de um mecanismo de tratamento de exceções. No modelo PDBC isto é possível graças ao conceito de restrições violáveis.

O conteúdo deste capítulo encontra-se organizado da seguinte maneira: a Seção 4.1 apresenta a motivação do desenvolvimento deste trabalho; a Seção 4.2 apresenta o modelo PDBC, destacando as restrições (Seção 4.2.2) e o mecanismo de sobrecarga (violação) das restrições como solução para as exceções dos workflows (Seção 4.2.4); e por fim a Seção 4.4 apresenta os trabalhos correlatos.

## 4.1 Introdução

Em domínios como o de processos de qualidade, engenharia de software e área médica existe uma necessidade para o que chamaremos de **workflows parciais**, ou seja, workflows que são apenas parcialmente definidos e são condicionais. No caso da área médica, quando um paciente é admitido em um hospital, ninguém tem as informações para definir, nesse instante, quais as atividades serão executadas sobre o paciente. Entretanto, quando o paciente recebe alta e é liberado do hospital, é fácil notar que este paciente fez parte da execução de um workflow, com um conjunto de atividades e ordem de execução entre

essas atividades. Em nenhum momento do tratamento existe a definição da estrutura de execução do caso do paciente.

Portanto, na área médica e em áreas como engenharia de software e de processos de qualidade, os planos de execução são definidos a medida que o processo é executado, ou seja, não existe uma estrutura rígida de execução. A determinação da próxima atividade que será executada no processo depende da própria execução do processo. Assim o usuário que participa da execução do processo tem a liberdade de determinar qual atividade executar, desde que obedeça às restrições do processo. Ou seja, existem regras ou restrições que precisam ser obedecidas e que: (*i*) impõem a execução de atividades não requisitadas, (*ii*) proíbem algumas atividades de serem executadas ou (*iii*) impedem algumas atividades de serem executadas de imediato.

Nos domínios que o plano de execução é definido a medida que o processo é executado, os workflows apresentam uma mistura de liberdade na escolha de qual atividade executar e rigidez na definição de qual atividade precisa ser executada. A parte menos flexível refere-se às restrições, que obrigam a execução de algumas atividades antes da execução da atividade objetivo. Por exemplo, considere o caso em que uma intervenção cirúrgica é necessária (isto é um exemplo de procedimento que normalmente não é determinado no momento em que o paciente é admitido no hospital), então o paciente ou responsável deve assinar um termo de consentimento para realização de uma transfusão de sangue, caso esta seja necessária durante a cirurgia. Assim, a condição para cirurgia acontecer é que o médico tenha o termo de consentimento do paciente para realização de uma transfusão de sangue.

Portanto, em domínios de processos fracamente definidos, geralmente existe uma ou mais pessoas que determinam que atividade precisa ser executada, ou seja, a atividade objetivo (*target*). Chamaremos essa pessoa de *controlador*. No exemplo do hospital, o médico que decide por realizar a intervenção cirúrgica - atividade objetivo - no paciente é o controlador.

No cenário em que o sistema de workflow funciona como um ajudante, o sistema de workflow ajuda o controlador a decidir sobre as implicações de se executar a atividade objetivo. As implicações envolvem a necessidade de executar atividades antes ou depois de se executar a atividade objetivo. Tais atividades são chamadas de *atividades obrigatórias*. Além das atividades obrigatórias, as restrições podem fazer referência a tempo, o que induz a definição de outras restrições sobre a execução da atividade objetivo. Por exemplo, em uma empresa pode existir a política de que as reuniões precisam ser agendadas pelo menos dois dias antes da data da reunião. Assim quando um usuário do sistema de workflow tentar fazer uma chamada para reunião nesta empresa, este usuário será notificado da restrição de que o mais cedo que esta reunião pode ocorrer é em dois dias.

### Sistemas de workflow tradicionais

Os sistemas de workflow tradicionais não são adequados para modelar e executar os exemplos de workflow apresentados. Os sistemas de workflow, como utilizado no ambiente de negócios, são adequados para processos bem definidos. Nestes processos, a definição do workflow é total e completa: o workflow possui em sua definição todas as atividades que serão executadas bem como a ordem de execução dessas atividades. Nestes workflows, quando uma atividade é concluída o sistema de workflow é capaz de determinar exatamente quais atividades devem iniciar.

Os sistemas de workflow para estes processos funcionam como um despachante de atividades aos usuários. No final da execução de uma atividade, o sistema de workflow determina quais são as atividades que devem ser iniciadas e despacha tais atividades à lista de trabalho dos usuários que serão os executantes destas atividades. Portanto, o controle de qual atividade será executada é do sistema de workflow, mas o controle de quando a atividade é executada fica a cargo do executante.

## 4.2 O modelo PDBC

A descrição do processo é o coração da tecnologia de workflow [11]. Acreditando nisso propomos o modelo de definição de processos baseado em restrições, o PDBC (***P**rocess **D**efinition **B**ased on **C**onstraints*). Neste modelo o processo é definido a partir de um conjunto de restrições e atividades. Acreditamos que a definição a partir das restrições permite um modelo de execução menos rígido do processo.

Para definir um processo o usuário terá que indicar quais atividades compõem o processo, bem como quais restrições fazem parte da definição deste processo. Este modelo de definição é diferente do tradicional, que a estrutura de execução das atividades faz parte da definição do processo. Ou seja, no modelo tradicional o usuário indica na definição dos processos as atividades, a ordem de execução das atividades e as transições entre as atividades.

A Figura 4.1 apresenta o modelo PDBC como um diagrama de classes em UML [8]. Um processo é formado por pelo menos uma atividade. Além da atividade, o processo possui zero ou mais restrições sobre as atividades. A restrição define uma condição de execução de uma atividade objetivo, portanto cada restrição possui pelo menos um requisito de execução da atividade objetivo. Assim, uma restrição deve possuir pelo menos uma atividade obrigatória, ou seja, uma pré-condição ou pós-condição.

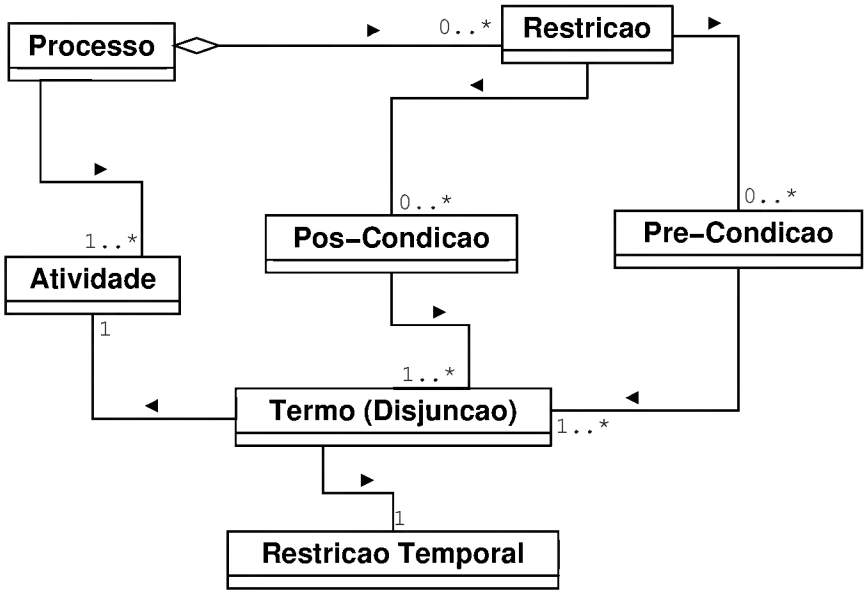


Figura 4.1: Diagrama de classes UML para o modelo PDBC.

### 4.2.1 Flexibilidade: algumas definições

A flexibilidade é considerada um tópico de pesquisa de grande relevância na área de gerenciamento de workflows. Problemas distintos relacionados à flexibilidade nos sistemas de workflow são discutidos na literatura, que pode causar confusão quanto à definição de flexibilidade. Para elucidar os diferentes contextos em que o termo flexibilidade é usado, apresentamos a seguir algumas definições encontradas na literatura para o termo em pesquisas sobre workflow.

Heinl et al. em [15] classifica a flexibilidade em dois grupos: as obtidas por seleção e as obtidas por adaptação. No primeiro caso um workflow é flexível se podemos derivar vários caminhos de execução a partir de uma definição de workflow. No segundo caso um workflow é flexível se o workflow pode ser modificado. A necessidade de modificação ocorre quando a seleção dos caminhos de execução disponíveis não são suficientes para atender uma situação.

Os trabalhos [17, 18, 23] descrevem a flexibilidade como a habilidade do workflow de executar sobre um modelo parcialmente definido, tal que a completa especificação é feita em tempo de execução e pode ser única para cada instância de workflow.

Voorhoeve e Aalst em [27] apresentam o termo workflow ad-hoc. Workflows ad-hoc são os workflows cuja instância é derivada a partir de modelos (*templates*) de workflows que podem ser modificados quando necessário, ou seja, para atender necessidades específicas.

Tais modelos não descrevem em detalhe como o caso deve ser executado.

Neste trabalho, usaremos o termo flexibilidade para indicar a capacidade que um workflow parcialmente definido possui em permitir múltiplos caminhos de execução, tal que a completa definição é realizada pelo usuário durante a execução do workflow.

### 4.2.2 Restrições

Como parte do modelo PDBC apresentaremos a seguir uma linguagem simplificada de definição de restrições. As restrições são representadas através de uma linguagem própria. Esta linguagem fornece os atributos necessários para representar uma restrição, que são:

**Rule**
: Este atributo indica o nome da restrição;

**Target**
: Este atributo indica a atividade cuja execução está sujeita a restrição e que o usuário do sistema deseja executar;

**Precondition**
: Este atributo indica as atividades que devem ser executadas antes de executar a atividade indicada pelo atributo *Target*;

**Postcondition**
: Este atributo indica as atividades que devem ser executadas após a execução da atividade indicada pelo atributo *Target*.

Considere o exemplo abaixo.

```
rule: C
   target: X
   precondition: Y
   postcondition: Z
```

Esta restrição possui como identificador único o nome $C$. A atividade objetivo é $X$, e sua pré-condição é a atividade $Y$. Após a conclusão da atividade $X$, a atividade $Z$ deve ser executada.

A declaração da restrição indica que a execução de uma atividade (*target*) implica a execução de pelo menos uma atividade obrigatória (*precondition* ou *postcondition*). No caso da declaração acima, representada pela pré-condição $Y$ e pós-condição $Z$. A pós-condição é representada quando não existe uma restrição entre a pós-condição e a atividade indicada no atributo *target*.

O usuário ao definir o processo deve listar as restrições que compõem o processo, declarando para cada restrição, a atividade que provoca o teste da restrição (*target*) e as implicações da execução desta atividade, ou seja, quais atividades devem ser executadas antes (*precondition*) e quais atividades devem ser executadas depois (*postcondition*).

Outras construções podem ser utilizadas na linguagem, como segue:

**Disjunção de pré-condições e pós-condições**

As pré-condições formam uma conjunção de restrições, mas para representarmos uma disjunção de pré-condições, basta incluir na definição de uma pré-condição as atividades que formam a disjunção de pré-condições separadas pelo conectivo OR. O mesmo se aplica para as pós-condições, conforme ilustra o exemplo abaixo:

```
rule: c01
   target: E
   precondition: A or B
   precondition: C or D
   postcondition: F or G
```

Esta regra indica que para E ser executada A ou B foi executada antes e C ou D também foi executada antes. Após E ser executada F ou G deve ser executada.

**Restrição de não execução de uma atividade**

As pré e pós-condições indicam que a atividade deve ser executada. Para indicar uma restrição de que a atividade não deve ser executada, basta utilizar o operador NOT, conforme exemplo abaixo:

```
rule: c01
   target: B
   precondition: not A
```

Esta regra indica que para B ser executada A não deve ter sido executada antes.

**Atributo *parcondition***

Este atributo indica as atividades que devem ser executadas antes ou depois da execução da atividade indicada pelo atributo *Target*. Também representa as atividades obrigatórias.

```
rule: c01
   target: B
   parcondition: A
```

Esta regra indica que para B ser executada A deve ter sido executada antes ou deve ser executada depois.

O uso das restrições permite uma definição de workflow mais flexível porque não apresenta ao usuário uma definição rígida de execução. Por exemplo, considere um workflow com quatro atividades $Atividades = \{A, B, C, D\}$ e a restrição abaixo:

```
rule: C01
   target: C
   precondition: A
```

Esta restrição indica que a atividade $A$ deve ser executada antes da atividade $C$ ser executada. Dizemos que esta definição é flexível porque o usuário que executar o workflow com esta definição não será obrigado a executar a atividade $C$ logo após a atividade $A$ ser concluída, ou seja, o usuário pode executar uma outra atividade, $B$ por exemplo. O usuário deve apenas obedecer a restrição de executar $A$ antes de $C$. Além disso, como as atividades $A$, $B$ e $D$ não possuem restrições de execução, podem ser executadas em qualquer ordem, ou seja, o usuário determina a ordem que julgar melhor para executar estas atividades.

### 4.2.3 Representação de workflows totais

As restrições que apresentamos acima também podem ser usadas na representação de um workflow total. Chamamos de workflow total o workflow que possui uma definição completa do processo. Assim, poderemos considerar que a linguagem que apresentamos tenha um poder de representação tão bom quanto as linguagens usadas na representação de workflows totais. Considere o workflow da Figura 4.2. Este workflow utiliza os seguintes construtores: *AND-SPLIT*, *AND-JOIN*, *OR-SPLIT*, *OR-JOIN* e seqüência.

A seguir apresentamos uma representação do workflow da Figura 4.2 utilizando a linguagem de definição de restrições. Para simplificar o entendimento do uso das restrições na representação do workflow, seguiremos a definição das restrições observando o workflow do fim para o início.

- A atividade $H$ deve ser executada antes do final do workflow, portanto é pré-condição da atividade $FIM$:

```
rule: c01
   target: FIM
   precondition: H
```

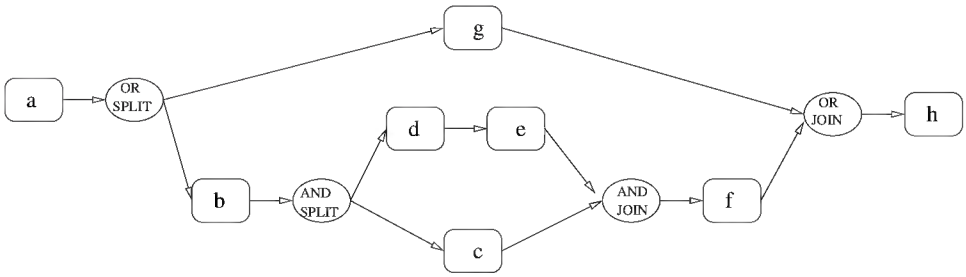


Figura 4.2: Exemplo tradicional de definição de workflow.

- A pré-condição de execução da atividade $H$ é a atividade $F$ ou a atividade $G$:

```
rule: c02
   target: H
   precondition: F or G
```

- A pré-condição de execução da atividade $G$ é a atividade $A$:

```
rule: c03
   target: G
   precondition: A
```

- A pré-condição de execução da atividade $F$ são as atividades $E$ e $C$:

```
rule: c04
   target: F
   precondition: E
   precondition: C
```

- A pré-condição de execução da atividade $E$ é a atividade $D$:

```
rule: c05
   target: E
   precondition: D
```

- A pré-condição de execução da atividade $D$ é a atividade $B$:

```
rule: c06
   target: D
   precondition: B
```

- A pré-condição de execução da atividade $C$ é a atividade $B$:

```
rule: c07
   target: C
   precondition: B
```

- A pré-condição de execução da atividade $B$ é a atividade $A$:

```
rule: c08
   target: B
   precondition: A
```

- Como a atividade $A$ não possui pré-condição, pode ser executada logo que o workflow inicia. Portanto, não é necessário apresentarmos uma restrição para a atividade $A$. Mas se desejarmos que a atividade $A$ seja habilitada para execução logo após o início da execução do workflow, podemos definir a seguinte restrição para a atividade

```
rule: c09
   target: INICIO
   postcondition: A
```

### 4.2.4 Restrições violáveis

As exceções nos sistemas de workflow podem ser de dois tipos, as provocadas por mal funcionamento de algum componente de software ou hardware do sistema; e as provocadas por desvios do fluxo de execução planejado pelo projetista do workflow (do processo). O primeiro tipo de exceção é chamado no trabalho de Wainer [7] como exceção de *infraestrutura*. As exceções de infraestrutura não fazem parte de nosso trabalho, por isso não apresentamos uma discussão sobre essas exceções aqui. As exceções que estamos

interessados são as que forçam um novo fluxo de execução nos sistemas de workflow. Portanto, tais sistemas necessitam de um mecanismo de tratamento desses desvios de fluxo de execução.

Para entendermos o que são estes desvios consideremos como exemplo o workflow ilustrado na Figura 4.3. Este workflow refere-se a um processo simplificado de venda de livros pela internet. Considere uma instância deste workflow para um cliente especial, que possui alta prioridade. Este cliente precisa esperar a aprovação do crédito para realizar a compra? Ou de outra forma, a entrega da compra deste cliente é condicionada ao rigor de outras consultas realizadas para outros clientes (consulta cliente, consulta estoque e consulta crédito)? Este é um exemplo típico da situação excepcional que pode interferir na execução do workflow, provocando um desvio do fluxo de execução da instância do workflow.

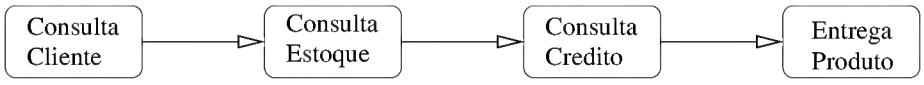


Figura 4.3: Exemplo simplificado de workflow: comércio pela internet.

Apresentamos o uso das restrições violáveis como uma solução para o tratamento de situações excepcionais como apresentadas acima. As restrições são consideradas violáveis porque podem ser violadas em situações excepcionais por usuários autorizados. O direito de sobrecarga (violação) de uma restrição está representado numa estrutura que implemente o modelo WRBAC. Por isso consideramos importante a integração entre os modelos PDBC e WRBAC. Enquanto o modelo PDBC mantém informações sobre workflow, ou seja, atividades e restrições entre as atividades, o modelo WRBAC mantém informações sobre segurança e políticas de acesso, ou seja, os usuários, papéis destes usuários e privilégios destes usuários.

Para a sobrecarga de uma restrição ocorrer, duas condições devem ser satisfeitas: primeiro, é preciso que uma restrição da definição do workflow seja realizada. Isto ocorre quando um usuário tentar realizar uma operação que desobedeça a definição da restrição. Segundo, é preciso que o usuário que provocou a ocorrência da restrição tenha o direito de violar a restrição. Sugerimos duas formas de implementar o direito do usuário violar uma restrição:

### Sobrecarga baseada na indicação de importância

Este mecanismo de sobrecarga é semelhante ao utilizado pelo modelo WRBAC. Atribuímos a cada restrição um valor numérico que indica o quão importante é a restrição.

O direito de sobrecarga da restrição é representado por um privilégio combinado com um valor numérico que indica o nível máximo de importância de restrição que pode ser violado. Esta proposta de sobrecarga de restrição está apoiada no fato de acharmos que algumas restrições são mais importantes que outras, da mesma forma que achamos que o direito de violar restrições é condicionado à importância organizacional de quem está violando uma restrição. Entretanto acreditamos que esta abordagem, apesar de simples, possui uma limitação quando a estrutura organizacional não representa um mapeamento direto para uma estrutura de papéis.

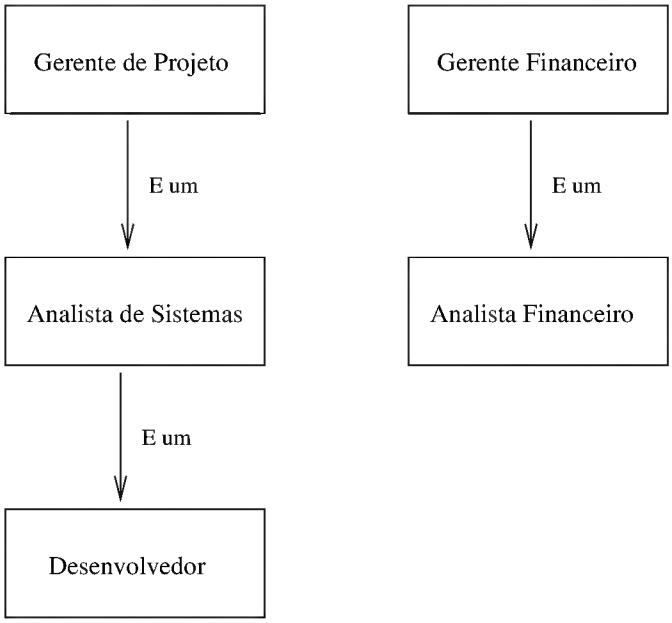


Figura 4.4: Exemplo de hierarquias de papéis desconexas.

Dizemos que a estrutura organizacional não representa um mapeamento direto quando os papéis existentes na empresa não pertencem à mesma hierarquia de papéis, ou seja, quando temos duas ou mais hierarquias de papéis desconexas (ver Figura 4.4). Por exemplo, é comum nas empresas o presidente pertencer a um papel que não possui relação com o papel de administrador de banco de dados, assim o presidente não possui privilégios que um administrador de banco de dados possui. Entretanto, mesmo como presidente da empresa este usuário não terá o direito de executar os privilégios de um administrador de banco de dados em uma situação excepcional. Ou seja, temos um conflito: o presidente deve possui um privilégio máximo de violação de restrição por ser o presidente da empresa, mas não deve violar uma restrição relacionada a um privilégio que um presidente não pode possuir por não ter habilidades para tal. Este conflito portanto, representa uma

limitação deste mecanismo de sobrecarga. A estratégia a seguir sugere uma solução para este problema.

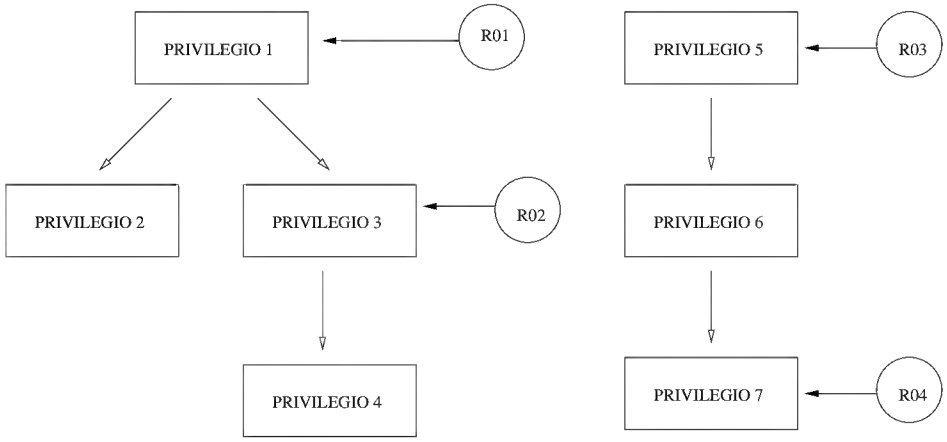


Figura 4.5: Herança de privilégios.

### Sobrecarga baseada na indicação direta do direito de violar uma restrição

Este mecanismo de sobrecarga utiliza a definição de um privilégio que indica diretamente qual restrição o usuário tem o direito de violar. Este mecanismo demanda mais trabalho do projetista do mecanismo de controle de acesso, pois terá que definir para cada restrição um privilégio e depois associar este privilégio aos papéis que possuem o direito de violar as restrições indicadas pelos privilégios. Entretanto, com este mecanismo de representação de privilégios, podemos resolver a limitação de atribuirmos um valor numérico às restrições.

A principal vantagem de atribuirmos um valor numérico às restrições está na simplicidade de representarmos uma relação de importância entre duas restrições. Entretanto, apresentamos nas Figuras 4.5 e 4.6 duas propostas de representação de relação de importância deste mecanismo. Na primeira proposta utilizamos a relação de herança de privilégios do modelo WRBAC, ou seja, a relação "**imply**". Na segunda proposta utilizamos a relação de herança de papéis do modelo WRBAC, ou seja, a relação "**is a**".

A Figura 4.5 ilustra duas hierarquias de privilégios. Cada hierarquia define uma relação de importância entre os privilégios, tal que quanto mais acima na hierarquia o privilégio estiver, maior é a sua importância (na Figura 4.5 o privilégio 1 é mais importante que o privilégio 4). Entretanto, dado dois privilégios, cada um pertencente a uma hierarquia de privilégio diferente, não podemos definir uma relação de importância entre esses privilégios (na Figura 4.5, os privilégio 3 e 5).

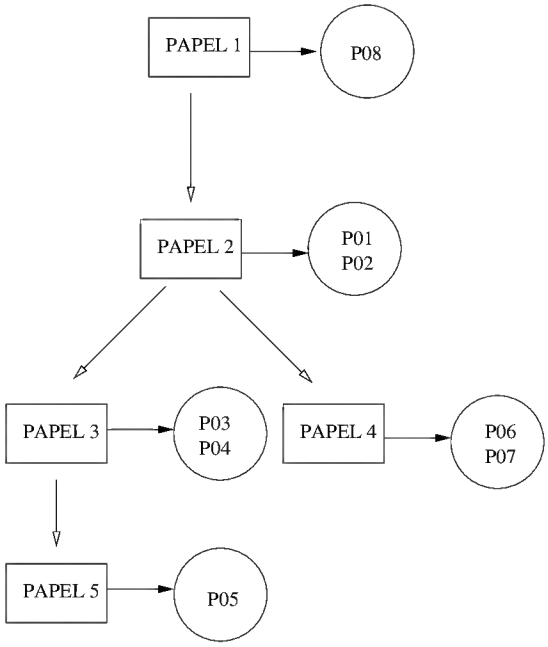


Figura 4.6: Herança de papéis.

A Figura 4.6 ilustra uma hierarquia de papéis. Nesta hierarquia a representação da relação de importância entre os papéis é realizada de forma indireta. Assim, um privilégio $P1$ é mais importante que um privilégio $P2$, quando o privilégio $P1$ é definido para um papel $R1$ ($hold(R1, P1)$) que é mais importante que um papel $R2$ que define o privilégio $P2$ ($hold(R2, P2)$). A importância de um papel está relacionada à posição deste papel na hierarquia de papéis, tal que quanto mais acima na hierarquia o papel estiver, maior é a sua importância.

### 4.2.5 Restrições temporais

Em alguns domínios as restrições temporais relacionadas às atividades obrigatórias podem adicionar restrições na execução da atividade objetivo. Por exemplo, uma organização pode definir uma política de que as reuniões devem ser precedidas pela distribuição de um material preparatório ao menos dois dias antes da reunião. Desta forma, o workflow como um ajudante pode informar o controlador que, para invocar uma reunião com a equipe de desenvolvimento, ele precisa gerar o material preparatório e que o data mais breve que a reunião pode ocorrer são dois dias após a invocação da reunião.

Adicionamos as restrições temporais na definição de uma restrição a fim de atender requisitos temporais como no exemplo acima. Considere o exemplo de restrição abaixo:

```
rule: c01
   target: B
   precondition: A[2, 5]
   postcondition: C[3]
```

Esta restrição indica que para executar a atividade B a atividade A deve ter sido executada ao menos 2 unidades de tempo ou no máximo 5 unidades de tempo antes. A atividade C deve ser executada pelo menos 3 unidades de tempo após a atividade B ter sido executada.

As restrições temporais, usadas na definição da restrição, permitem a definição dos limites do instante em que a atividade deve ser executada. As restrições temporais não são usadas para indicar os limites do tempo de execução de uma atividade.

## 4.3 Um exemplo de workflow baseado em restrições

Apresentaremos nesta seção um exemplo de workflow definido como um conjunto de restrições. Inicialmente apresentamos a definição do workflow (atividades e restrições que compõem o workflow) utilizando a linguagem de restrições. A partir da definição exploraremos alguns fluxos de execução que a definição do workflow permite, ilustrando através de exemplos o conceito que flexibilidade que utilizamos neste trabalho. Também mostraremos com este exemplo, situações em que o usuário consulta as pré-condições de uma atividade, agenda a execução de uma atividade e viola uma restrição da definição do workflow.

O código abaixo representa a definição do workflow exemplo. Tal workflow é formado por oito atividades ($A$, $B$, $C$, $D$, $E$, $F$, $G$ e $H$) e por três restrições ($c01$, $c02$ e $c03$).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Workflow: "Workflow Exemplo"
Activities: A, B, C, D, E, F, G, H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rule: c01
   target: C
   precondition: A[0, 10]
   precondition: B
   postcondition: D
   postcondition: F[0, 5]

rule: c02
```

```
   target: F
   precondition: E
   postcondition: G[2, 5]

rule: c03
   target: B
   postcondition: F[15]
   postcondition: G
```

A partir dessa definição de workflow podemos gerar o grafo ilustrado na Figura 4.7. As arestas foram diferenciadas para indicarem pré e pós-condições. A orientação das arestas indica o sentido da relação entre as atividades. Assim, a aresta que conecta as atividades $A$ e $C$ indica que a atividade $A$ é pré-condição para a atividade $C$, enquanto que a aresta que conecta as atividades $B$ e $G$ indica que a atividade $G$ é pós-condição da atividade $B$. A atividade $H$ não possui restrição e nem é restrição para nenhuma outra atividade do workflow, por isso não existe nenhuma aresta ligando a atividade $H$.

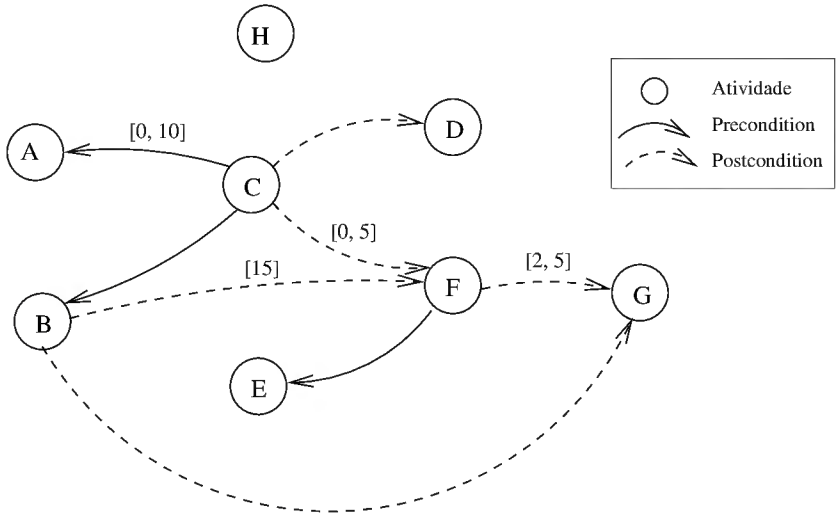


Figura 4.7: Grafo do workflow exemplo.

Um fluxo de execução para o *workflow exemplo* pode ser sugerido a partir do grafo da Figura 4.7. Para isto bastaria inverter a orientação das arestas que indicam a relação de pré-condição entre duas atividades.

Uma execução serial da definição do workflow acima é ilustrada na Figura 4.8. A ordem das atividades $A$ e $B$, assim como das atividades $D$ e $E$, pode ser trocada que a execução do workflow continuará serial e consistente com a definição. A atividade

$H$, representada na figura como última atividade da execução do workflow, poderia ser executada em qualquer ordem.


Figura 4.8: Exemplo de execução serial do workflow exemplo.

Além das execuções seriais, diferentes estruturas de execução poderiam ser derivadas a partir da definição do *workflow exemplo*. Cada estrutura de execução é dinamicamente definida pelo controlador durante a execução do workflow. Entretanto, não apresentamos essas diferentes estruturas de execução porque seria complicado.

O controlador do workflow é a pessoa responsável por determinar que atividade executar durante a execução do workflow, entretanto tal liberdade é limitada pela definição das restrições do mesmo. Por exemplo, caso o controlador desejasse executar a atividade $F$ o sistema de workflow verificaria se a atividade $E$ foi executada antes. Além disso, para as atividades que possuem restrição temporal é necessário realizar a verificação dessa restrição também. Nestes casos não é suficiente garantir que a atividade que é pré-condição foi executada antes, pois a restrição temporal também precisa ser testada. É o caso da atividade $A$ que é pré-condição da atividade $C$, e que deve ser executada no máximo dez unidades de tempo antes.

No cenário em que o sistema de workflow funciona como um ajudante, o sistema de workflow ajuda o controlador a decidir sobre as implicações de se executar a atividade objetivo. As implicações envolvem a necessidade de executar atividades antes ou depois de se executar a atividade objetivo. Por exemplo, considere o controlador requisitando do sistema as atividades que precisam ser executadas antes para que a atividade $C$, do exemplo acima, seja executada. A seguir apresentamos uma lista com as possíveis respostas para este exemplo, indicadas pelo conjunto $R$. Além das respostas apresentamos em que condições tais respostas são produzidas.

- $\mathbf{R = \{\ \}}$. Esta resposta é produzida quando as atividades $A$ e $B$ foram executadas antes e a execução da atividade $C$ no instante da consulta não viola a restrição temporal existente entre as atividades $A$ e $C$.
- $\mathbf{R = \{A[0,\ 10]\}}$. Esta resposta é produzida quando apenas a atividade $B$ foi executada antes.
- $\mathbf{R = \{B\}}$. Esta resposta é produzida quando apenas a atividade $A$ foi executada antes e a execução da atividade $C$ no instante da consulta não viola a restrição temporal existente entre as atividades $A$ e $C$.

- **R = {A[0, 10], B}**. Esta resposta é produzida quando nenhuma das pré-condições de $C$ foi executada antes. Entretanto, esta resposta também é produzida se a atividade $A$ foi executada antes, mas a execução da atividade $C$ no instante da consulta viola a restrição temporal existente entre as atividades $A$ e $C$.

Novamente no cenário do workflow como um ajudante, o usuário pode, ao invés de requisitar quais atividades precisam ser executadas antes de uma atividade dada, agendar a execução de uma atividade. Por exemplo, considere o controlador agendando a atividade $C$. Neste caso o sistema de workflow iria executar algum dos procedimentos apresentados na lista abaixo.

- **Agendar apenas a atividade** $C$. Este procedimento é executado quando as atividades $A$ e $B$ foram executadas antes e a execução da atividade $C$ no instante do agendamento não viola a restrição temporal existente entre as atividades $A$ e $C$;

- **Agendar as atividades** $A$ **e** $C$. Este procedimento é executado quando as atividades $A$ e $B$ foram executadas antes, mas a execução da atividade $C$ no instante do agendamento viola a restrição temporal existente entre as atividades $A$ e $C$. Neste caso a atividade $A$ será re-executada. Este procedimento também é executado se a atividade $A$ não foi executada no instante do agendamento;

- **Agendar as atividades** $B$ **e** $C$. Este procedimento é executado quando apenas a atividade $A$ foi executada antes e a execução da atividade $C$ no instante do agendamento não viola a restrição temporal existente entre as atividades $A$ e $C$;

- **Agendar as atividades** $A$, $B$ **e** $C$. Este procedimento é executado quando nenhuma das pré-condições de $C$ foi executada antes. Entretanto, este procedimento também é executado se a atividade

  $A$ foi executada antes, mas a execução da atividade $C$ no instante do agendamento viola a restrição temporal existente entre as atividades $A$ e $C$. Neste caso a atividade $A$ será re-executada.

Os procedimentos listados acima referem-se apenas às pré-condições da atividade $C$ e a própria atividade $C$. Entretanto, para cada atividade agendada, o sistema também deve agendar a execução das pós-condições. Ou seja, no caso da atividade $C$, as atividades $D$ e $F$ também devem ser agendadas. O mesmo deve ser feito para as atividades agendadas que são pré-condição da atividade $C$. Entretanto, como no exemplo a atividade $F$ é agendada por ser pós-condição da atividade $C$, não precisará ser agendada novamente para a atividade $B$.

Os exemplos apresentados até agora não consideraram a possibilidade de violação de uma restrição da definição do workflow. Como apresentado na Seção 4.2.4, em situações excepcionais um usuário autorizado poderá violar a definição da restrição. O mecanismo de tratamento dos desvios de execução provocados por situações excepcionais permite que caminhos alternativos de execução possam ser utilizados. Assim, consideramos a utilização desses mecanismos como uma forma de apoio à flexibilidade. Como exemplo de violação de restrição, considere o seguinte cenário:

- a execução da atividade $B$ foi concluída no instante 5;
- a execução da atividade $C$ foi concluída no instante 20;
- um usuário deseja executar a atividade $F$ de imediato, no instante 24.

A partir da descrição do cenário, podemos concluir que a execução da atividade $F$ no instante requisitado pelo usuário (instante 24), apesar de satisfazer a restrição entre as atividades $C$ e $F$, viola a restrição temporal definida entre as atividades $B$ e $F$. Neste caso, para um usuário poder executar a atividade $F$, o mesmo terá que violar esta restrição.

A violação de uma restrição é definida no modelo PDBC como um privilégio. No exemplo acima, para o usuário violar a restrição (temporal) identificada, o usuário terá que possuir um papel que lhe conceda o direito de violar a restrição, do contrário o usuário não poderá executar a atividade.

## 4.4 Trabalhos relacionados

A literatura de workflow ainda considera a flexibilidade um problema. Muitos trabalhos relacionados ao provimento de flexibilidade nos sistemas de workflow encontram-se em desenvolvimento, seguindo linhas de pesquisas distintas. Entre as abordagens utilizadas podemos citar as que tratam o problema da mudança dinâmica na definição de um processo e as conseqüências dessas mudanças nas instâncias de workflow ativas. Uma segunda abordagem refere-se ao entendimento e desenvolvimento de um mecanismo de tratamento de exceções, ou seja, refere-se ao comportamento do sistema de workflow durante um desvio excepcional do fluxo de execução de uma instância de workflow. Essas duas abordagens adotadas na literatura difere da abordagem utilizada neste trabalho. Apesar de propormos um mecanismo de tratamento de exceções, estendemos a nossa solução através da definição de uma nova forma de modelagem de workflow. Nós propomos o uso de restrições como instrumento de definição de workflow. Acreditamos que o workflow definido como um conjunto de restrições possibilita uma execução menos rígida. Conhecemos poucos trabalhos que usam as restrições como mecanismo de definição de workflows.

Os trabalhos de Mangan e Sadiq [17, 18] são os que mais se aproximam de nossa proposta. Nestes trabalhos ou autores apresentam a flexibilidade nos sistemas de workflow como a propriedade que estes sistemas possuem de executar processos parcialmente definidos, deixando a especificação completa dos processos durante a execução dos mesmos. O workflow completo é construído através da junção de pequenos segmentos de workflow. As restrições são usadas durante a especificação do processo e podem ser:

- **Restrição de seleção**, que indica quais segmentos de workflow podem ser usados na construção de um workflow em execução, ou seja, um segmento de workflow é disponibilizado para construção de um processo quando não viola nenhuma restrição de seleção;
- **restrição de terminação**, que indica uma condição de terminação de um processo, ou seja, um processo pode ser finalizado quando não viola nenhuma restrição de terminação;
- e as **restrições de construção** que disciplinam a construção de uma instância de workflow, ou seja, tais restrições são verificadas no instante que o usuário tenta colar um segmento de workflow.

Nesses trabalhos Mangan e Sadiq modelam o sistema de cursos de uma universidade. Uma disciplina é considerada como uma tarefa de um workflow, portanto um segmento de workflow. O curso é considerado um processo, assim para o aluno - usuário do sistema de workflow - matricular-se em uma disciplina, o mesmo deve ter completado os pré-requisitos dessa disciplina, ou seja, as restrições de seleção. Para um aluno concluir um curso, o mesmo deve completar um número mínimo de créditos, ou seja, uma restrição de terminação. A restrição de construção indica o número máximo de alunos por disciplina, assim uma disciplina pode não satisfazer uma restrição de construção mesmo que satisfaça as restrições de seleção.

Uma limitação dos trabalhos de Mangan e Sadiq [17, 18] refere-se à ausência de um mecanismo de ativação. As restrições não possuem a propriedade de indicar o momento em que uma atividade deve iniciar. Esta limitação não chega a ser um problema para o exemplo do sistema de cursos de uma universidade, pois este não necessita de um mecanismo de ativação. Outra limitação relacionada ao mecanismo de ativação refere-se à definição de pós-condições. Por exemplo, uma atividade $B$ pode ser requisitada para execução após a atividade $A$ ser executada.

O trabalho reportado em [28] apresenta uma linguagem lógica temporal de definição de workflow. Esta linguagem é utilizada para representar a relação de ordem temporal entre as atividades de um workflow. A pré-condição de nosso modelo é semelhante com a relação ***before*** desse trabalho. Outra semelhança desse trabalho com o nosso modelo

refere-se ao mecanismo de ativação de atividades. Entretanto, o mecanismo de definição de pós-condição não é contemplado neste trabalho.

Paul Dourish et al. em [11] descrevem o *Freeflow*, o protótipo de um sistema de workflow que utiliza restrições em seu modelo de definição de workflow. Neste trabalho Dourish apresenta as restrições como uma forma declarativa de definir dependências entre as atividades de um workflow.

Além disso o Freeflow adota um modelo de execução do processo diferente do tradicional. No modelo tradicional de execução, as ações do usuário são orientadas ou guiadas pela definição do processo, enquanto que no Freeflow as ações do usuário são apenas comparadas com a definição. Neste modelo o usuário tem total controle sobre as ações que deseja tomar, mesmo no momento de violar uma restrição. O sistema Freeflow apenas alerta o usuário sobre a existência da restrição e questiona-o sobre prosseguimento da ação na presença da restrição. No Freeflow isto é possível porque não existe integrado ao sistema nenhum mecanismo de controle de acesso semelhante ao WRBAC. Outra limitação do Freeflow refere-se à inexistência de um mecanismo de ativação de tarefas.

A área de engenharia de software também desenvolveu alguns trabalhos relacionados à especificação de processos usando regras [22, 5, 14]. O trabalho de Reis et al. [22] apresenta um modelo para definição de políticas estáticas usadas na definição de processos em ambientes de desenvolvimento de software. Tais políticas são úteis na verificação dos processos definidos pelos usuários. A política estática descreve uma condição lógica que deve ser satisfeita para que um processo seja verificado.

As políticas estáticas apresentadas em [22] são aplicadas na modelagem do processo, diferente da abordagem adotada pelo modelo PDBC, que aplica as restrições durante a execução do workflow.

Outro trabalho da área de engenharia de software que podemos citar é o ambiente Marvel [5, 14], que é um ambiente de desenvolvimento baseado em regras (abreviação do inglês, RBDE). No Marvel, o projetista cria um ambiente através da definição do modelo de dados e do modelo de processo. O modelo de processo é definido através de regras que especificam uma condição e os efeitos da execução da regra. A condição é uma declaração lógica que precisa ser satisfeita para a regra ser executada, enquanto que os efeitos são declarações que indicam as mudanças sobre o banco de dados de objetos. Semelhante ao modelo PDBC, as regras no ambiente Marvel são aplicadas durante a execução do processo. Entretanto, diferente do modelo PDBC, os usuários no ambiente Marvel são livres para escolher qualquer ação do sistema, independente das pré-condições existentes.

# Capítulo 5

# Servidor Tucupi: uma implementação para o modelo PDBC

Este capítulo apresentará o servidor Tucupi, um protótipo de um sistema de workflow que implementa o modelo PDBC. A característica de servidor do Tucupi é limitada à propriedade de execução dos sistemas de workflow. Para atender aos requisitos de definição do processo, foi desenvolvida uma aplicação componente ao servidor Tucupi. Esta aplicação será descrita na seção 5.3.

Este capítulo encontra-se organizado da seguinte forma: a Seção 5.1 descreverá a interface pública do servidor Tucupi, ou seja, a lista de métodos que podem ser executados por uma aplicação cliente; a Seção 5.2 descreverá a arquitetura destacando os componentes que fazem parte do servidor; por fim, a Seção 5.3 descreverá como é realizada a definição do processo através das regras, bem como a derivação destas regras em uma definição interpretável pelo servidor.

## 5.1 Interface pública

O servidor Tucupi é um protótipo de um sistema de workflow que suporta o uso de restrições na definição de um workflow. O servidor Tucupi foi implementado em Prolog e funciona como um serviço que utiliza sockets como mecanismo de comunicação. Assim, o servidor escuta uma porta TCP/IP onde clientes podem se conectar e executar os serviços que o servidor disponibiliza através de sua interface pública. Um cliente deste servidor precisa conhecer o formato das mensagens de interação com o servidor Tucupi. Com o intuito de apresentar como as mensagens devem ser enviadas ao servidor Tucupi, as seções a seguir descreverão quatro grupos diferentes destas mensagens. O primeiro grupo de mensagens refere-se à criação e destruição (encerramento) de um processo. O segundo grupo de mensagens refere-se ao início e fim da execução de uma atividade do processo. O

terceiro grupo refere-se às mensagens de consulta e o quarto grupo refere-se à mensagem de agendamento de execução de uma atividade.

### 5.1.1 Operações para criar e destruir um processo

Os sistemas de workflow podem controlar a existência de várias instâncias de processo. Cada instância de processo representa um caso particular de execução de um processo. Para criar e destruir uma instância de processo a aplicação servidora utiliza duas funções: **create** e **destroy**. Estas duas funções são utilizadas para determinar o tempo de execução de um processo, tal que a função **create** indica o instante em que foi criada a instância do processo e a função **destroy** indica o instante em que foi destruída a instância do processo.

Para iniciar e encerrar um processo, as mensagens devem ser enviadas pela aplicação cliente conforme representada na lista abaixo.

**create(Workflow, Case)**

A função **create** possui dois argumentos: *Workflow* que identifica o nome do processo e *Case* que identifica uma instância do processo do tipo identificado em Workflow.

**destroy(Workflow, Case)**

A função **destroy** possui dois argumentos: *Workflow* que identifica o nome do processo e *Case* que identifica uma instância do processo do tipo identificado em Workflow.

### 5.1.2 Operações para iniciar e terminar uma atividade

As atividades são elementos de um processo que indicam uma ação, ou seja, representa o que o usuário do sistema de workflow deve fazer para atingir o objetivo do processo. A execução de uma atividade está relacionada à execução de um processo, ou seja, é preciso existir uma instância de processo para que uma atividade seja iniciada/executada.

A aplicação servidora utiliza duas funções para representar o início e o fim da execução de uma atividade. As funções são **begin** e **end** respectivamente. Estas funções são também utilizadas para determinar o tempo de execução da atividade, tal que a função **begin** representa o instante em que a atividade foi iniciada e a função **end** representa o instante em que a atividade terminou a execução.

A aplicação servidora registra o instante em que essas funções foram executadas. Assim é possível implementar funções que manipulem restrições temporais. Por exemplo, o intervalo entre o fim da atividade A e o início da atividade B deve ser de no máximo 3 horas.

Alguns workflows possuem atividades cujo o tempo médio de execução é muito pequeno, sendo portanto desnecessário utilizar as funções que delimitam o tempo de execução de uma atividade. Entretanto, como a utilização destas funções não comprometem os workflows com atividades cujo o tempo médio de execução é muito pequeno, resolvemos manter a utilização das funções *begin* e *end*.

Para iniciar e terminar uma atividade, as mensagens devem ser enviadas pela aplicação cliente conforme representada na lista abaixo.

**begin(User, Activity, Workflow, Case)**

A função **begin** possui quatro argumentos: *User* que identifica o usuário que inicializou a atividade no processo, *Workflow* que identifica o nome do processo e *Case* que identifica uma instância do processo, identificado em Workflow, onde a atividade foi inicializada. Esta função utiliza o Componente WRBAC, através da execução da função *doer*. A execução da função *doer* permite que a execução da atividade *Activity* seja consistente quanto ao direito de execução da atividade pelo usuário *User*.

**end(Activity, Workflow, Case)**

A função **end** possui três argumentos: *Activity* que indica a atividade que foi concluída, *Workflow* que identifica o nome do processo e *Case* que identifica uma instância do processo, identificado em Workflow, onde a atividade foi terminada.

### 5.1.3 Operações de consulta

A coordenação dos processos e das atividades dos processos é característica essencial dos sistemas de workflow. As operações de consulta foram implementadas na aplicação servidora Tucupi para oferecer recursos de coordenação dos processos em execução.

A integração dos modelos WRBAC e PDBC fornece os requisitos para implementar funções de consulta aos dados organizacionais, que estão representados no modelo WRBAC, e de processos, que estão representados no modelo PDBC.

As mensagens devem ser enviadas pela aplicação cliente conforme representada na lista abaixo.

**next_activities(Workflow, Case)**

A função **next_activities** é utilizada para consultar quais as atividades que podem ser executadas na instância de processo indicada pelo argumento *Case*. O argumento *Workflow* indica o nome do processo. Uma atividade pode ser executada em uma instância de processo quando suas precondições forem concluídas.

**missing_activities(Activity, Workflow, Case)**

A função **missing_activities** é utilizada para consultar quais as atividades que ainda não foram concluídas (ou iniciadas) e que são pre-condição da atividade indicada pelo argumento *Activity*, na instância de processo indicada pelo argumento *Case*. O argumento *Workflow* indica o nome do processo.

**what_if(Activity, Workflow, Case)**

A resposta desta consulta é uma cadeia de atividade formada recursivamente pelas precondições de *Activity*. Ou seja, além das precondições, a resposta inclui as precondições das precondições, recursivamente, até não existir mais precondições. O argumento *Case* indica a instância de processo consultada e o argumento *Workflow* indica o nome do processo. A resposta não inclui as atividades que já foram concluídas.

**pending_activities(Workflow, Case)**

A função **pending_activities** é utilizada para consultar quais as atividades pendentes na instância de processo indicada pelo argumento *Case*. Atividades pendentes são atividades que não foram terminadas. Estas atividades podem ter sido iniciadas ou não. O argumento *Workflow* indica o nome do processo.

**finished_activities(Workflow, Case)**

A função **finished_activities** é utilizada para consultar quais as atividades foram terminadas na instância de processo indicada pelo argumento *Case*. O argumento *Workflow* indica o nome do processo.

**who(Activity, Workflow, Case)**

A função **who** é utilizada para consultar quais usuários podem executar a atividade indicada pelo argumento Activity na instância de processo indicada pelo argumento **Case**. O argumento *Workflow* indica o nome do processo. A execução desta função é delegada ao componente WRBAC, responsável pela manipulação e consulta de dados relacionados à estrutura organizacional e mecanismos de controle de acesso.

**can_do(User, Activity)**

A função *can_do* é utilizada para consultar se o usuário indicado pelo argumento User, pode executar a atividade indicada pelo argumento *Activity*. Assim como a função *who*, esta função é delegada ao componente WRBAC.

### 5.1.4 Operação de agendamento de um atividade

A aplicação servidora utiliza a função **schedule** para agendar a execução de uma atividade (normalmente as que possuem pré-condições). A execução do agendamento de uma

atividade **A** (*schedule* em A), implica no agendamento das atividades que formam a cadeia de pré-condições da atividade **A** que ainda não foram executadas na instância de workflow em questão.

Para determinar a cadeia de pré-condições de uma atividade **A** usamos a equação de ponto fixo abaixo, tal que *pre* indica a função que determina a cadeia de pré-condições de uma atividade e *predir* indica a função que determina as pré-condições diretas de uma atividade (atividades indicadas no atributo *precondition* na restrição) menos as atividades que já foram executadas. As atividades agendadas serão disponibilizadas para execução a medida que suas pré-condições forem sendo satisfeitas satisfeitas.

$$pre(A) = predir(A) \cup \bigcup_{a_i \in predir(A)} pre(a_i)$$

Para agendar a execução de uma atividade, a mensagem deve ser enviada pela aplicação cliente da forma indicada abaixo.

**schedule(Activity, Workflow, Case)**

A função schedule possui três argumentos: *Activity* que indica a atividade para agendar a execução, *Workflow* que identifica o nome do processo e *Case* que identifica uma instância do processo, identificado em Workflow, onde a atividade deve ser agendada.

## 5.2 Arquitetura

O servidor Tucupi foi implementado em prolog [9] (SWI-Prolog, Versão 5.0.10) como um servidor sockets. A implementação foi modularizada através da concepção de três componentes principais. A figura 5.1 ilustra os componentes que compõem o servidor Tucupi e a relação entre eles. Os componentes apresentados na figura seguem o formato da linguagem UML [8]. A integração entre os componente está esquematicamente representada através da implementação e utilização das interfaces dos componentes. A seguir uma breve apresentação de cada um dos componentes:

**Componente *Workflow Engine*.** Refere-se ao componente que realiza a instanciação, execução e consulta dos processos definidos como um conjunto de restrições. A interface pública do servidor Tucupi refere-se à interface deste componente.

**Componente de operações com tempo e data.** Refere-se ao componente que realiza as operações com data. O servidor Tucupi precisa realizar algumas

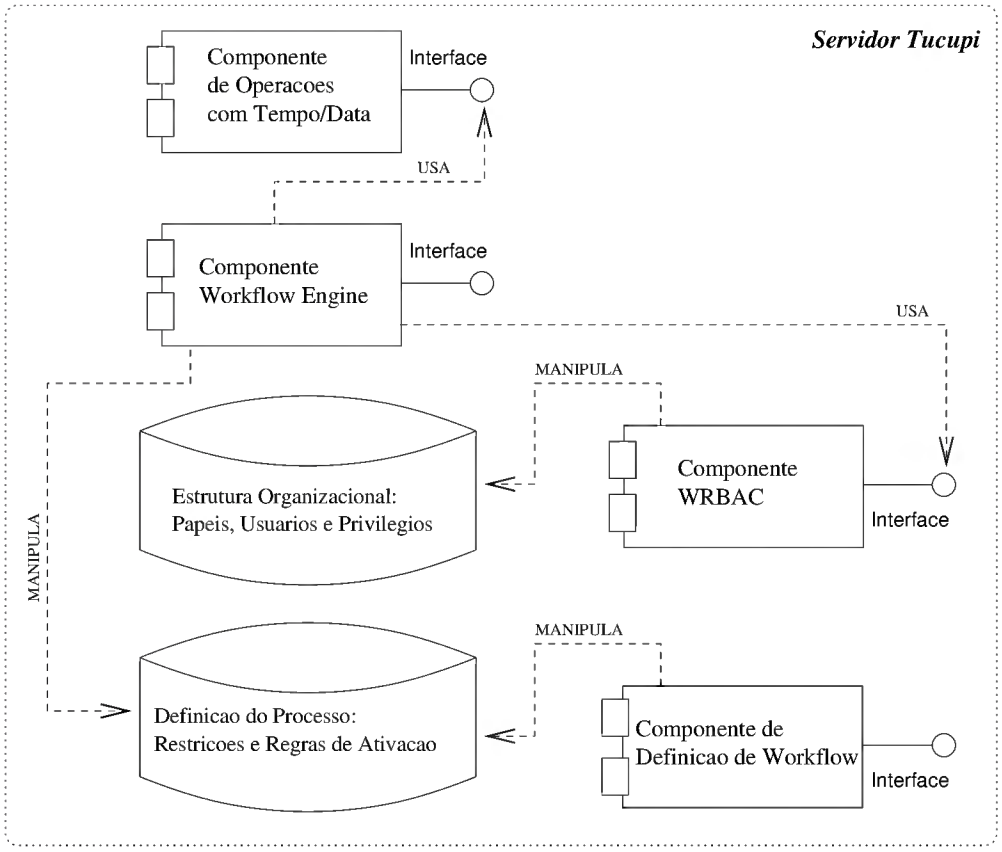


Figura 5.1: Arquitetura do servidor *Tucupi*.

operações com datas durante a execução de algumas funções do componente *Workflow Engine*. Como exemplo de operações com datas podemos citar: comparação entre tempos e datas, adição entre tempos e datas, subtração entre tempos e datas, etc.

**Componente *WRBAC Engine*.** Refere-se ao componente que realiza o controle e verificação de acesso à execução das atividades do processo. Este componente também realiza o controle da sobrecarga das restrições que fazem parte da definição do workflow.

### 5.2.1 Integração dos modelos WRBAC e PDBC

O componente *WRBAC Engine* é uma implementação do modelo WRBAC (***W**orkflow - **R**ole **B**ased **A**ccess **C**ontrol* [29]). O modelo WRBAC propõe um mecanismo de

controle de acesso para sistemas de workflow baseado em permissões atribuídas a papéis. O componente *Workflow Engine* é uma implementação do modelo PDBC (***P**rocess **D**efinition **B**ased on **C**onstraints*) que apresenta uma nova proposta de definição e execução de processos, o uso de restrições. As funções do componente *WRBAC Engine* são acessadas pelo componente *Workflow Engine* (figura 5.1). Entretanto, a aplicação cliente do servidor Tucupi conhece apenas a interface pública do componente *Workflow Engine*. As funções do componente *WRBAC Engine* são implicitamente utilizadas pela aplicação cliente.

O componente Workflow Engine é limitado a resolver consultas e execução de workflows. Este componente não possui em seu núcleo funções de controle de acesso dos usuários participantes da execução dos workflows. Para suprir esta limitação, o servidor Tucupi incorporou o componente WRBAC Engine, que realiza o controle de acesso dos workflows em execução, restringindo a execução das atividades somente aos usuários autorizados. Entende-se por usuário autorizado, aquele que possui o direito de executar uma atividade na instância de workflow.

Além disso, o componente WRBAC é integrado ao componente Workflow Engine por apresentar uma solução para o problema da execução pelo mesmo usuário de atividades incompatíveis em uma instância de workflow, ou seja, um usuário que executou uma atividade não pode executar na mesma instância de workflow uma atividade conflitante com a atividade executada antes (ex.: requisitar e aprovar compra). No WRBAC, este problema é resolvido através de restrições que definem separação de responsabilidades. Outra funcionalidade do componente WRBAC usada pelo componente Workflow Engine é a seleção dos usuários habilitados para executar uma atividade do workflow.

## 5.3 Definição do processo

O modelo PDBC propõe uma forma alternativa de definição de processo, o uso de restrições sobrecarregáveis (ou violáveis). Esta seção mostrará a declaração das restrições e descreverá como as regras geradas a partir de uma restrição são implementadas na aplicação servidora Tucupi. No final, uma ferramenta para definição de processo será apresentada.

Na ferramenta apresentada, o usuário realiza a declaração das regras e ao final do processo de definição destas regras, o processo de derivação é iniciado, produzindo como resultado um conjunto de predicados em prolog representando as restrições e as regras de ativação.

A propriedade de sobrecarga das restrições é implementada como permissões do modelo WRBAC. Assim, para um usuário violar uma restrição $R1$, este usuário terá que possui o privilégio de sobrecarga da restrição $R1$. Portanto, durante a execução do processo

o componente *Workflow Engine*, numa situação excepcional, requisitará ao componente *WRBAC Engine* a permissão para violação da restrição.

### 5.3.1 Restrição

A proposta do modelo PDBC é a de fornecer uma definição de processo para os sistemas de workflow através do uso de restrições. O uso das restrições permite a definição de workflows mais flexíveis. As restrições já fazem parte do universo do projetista de processos, pois quando o projetista modela a atividade $B$ sendo executada após a atividade $A$ é que *possivelmente* existe uma restrição que obriga a atividade $B$ ser executada apenas quando a atividade $A$ terminar.

Os workflows definidos com restrições são mais flexíveis porque o projetista do workflow ao definir a restrição de que a atividade $A$ deve ser executada antes da atividade $B$ *não* indica que após a execução da atividade $A$ a atividade $B$ deve ser executada em seguida. Assim a execução é flexível o suficiente para permitir que uma outra atividade, $C$ por exemplo, seja executada antes da atividade $B$. Ou seja, o usuário não segue um fluxo rígido de execução e possui a liberdade de escolher quais atividades executar, desde que respeite as restrições definidas.

Portanto, dizemos que os workflows definidos como um conjunto de restrições são mais flexíveis porque podem permitir diferentes estruturas de execução para uma mesma definição do workflow. Com o uso das restrições violáveis, outras estruturas de execução podem ser geradas. Como exemplo, considere o caso das atividades $A$ e $B$ citado acima. Poderíamos violar esta restrição e executar a atividade $B$ antes da atividade $A$ ou até não executar a atividade $A$.

### 5.3.2 Derivação de uma restrição

Uma restrição indica o que *não* é permitido fazer, portanto podemos dizer que as restrições são regras negativas [28]. Entretanto, os sistemas de workflow precisam, além das restrições, de regras que indicam o que fazer. Podemos chamar estas regras que dizem o que fazer de regras positivas, por possuirem uma natureza oposta às regras negativas.

As restrições, isoladamente, fornecem ao workflow apenas uma característica reativa. A derivação de uma restrição é um mecanismo de transformação de uma restrição em regras que: mantém o significado semântico da restrição, ou seja, indicam que ações durante a execução de um workflow o usuário é proibido de executar; e de regras proporcionam um mecanismo de ativação das atividades que formam a definição da restrição.

A ativação de uma atividade representa permissão de execução da atividade e a inclusão da atividade em um mecanismo de distribuição de tarefas de um sistema de

workflow.

O modelo PDBC propõe uma forma de derivação de uma restrição em até três novas regras: regras de ativação e restrições de ordem e/ou temporais. A figura 5.2 ilustra esquematicamente, a derivação de uma regra. A derivação apresentada na figura é em código prolog. Outras linguagens de programação poderiam ser utilizadas como resultado do processo de derivação.

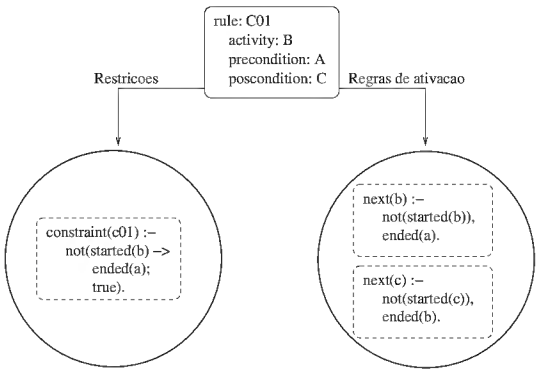


Figura 5.2: Esquema de derivação de uma regra da definição de um workflow.

A Figura 5.3 ilustra esquematicamente a derivação de uma restrição que possui na definição restrições temporais. Novamente a linguagem prolog é utilizada na derivação.

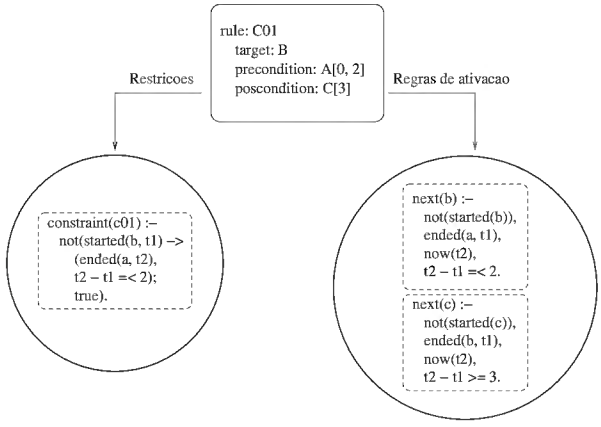


Figura 5.3: Esquema de derivação de uma restrição com restrições temporais.

Um processo, como definido pelo modelo PDBC, possui um conjunto de restrições, e a aplicação servidora mantém a derivação destas restrições em um banco de dados para

cada processo. Para ilustrar bem esta derivação considere o exemplo abaixo de definição de restrição com as atividades $A$ e $B$ quaisquer. Esta restrição indica que $B$ não deve começar antes de $A$ acabar. Além disso, como restrição temporal, $B$ não deve começar antes de duas unidades de tempo ou depois de oito unidades de tempo, considerando o fim da execução da atividade $A$. Como pós-condição, a atividade $C$ deve ser executada após a atividade $B$ ser concluída. Esta pós-condição declara como restrição temporal que o início da atividade $C$ não deve ser anterior a uma unidade de tempo ou superior a cinco unidades de tempo, considerando o fim da execução da atividade $B$.

```
rule: C01
   target: B
   precondition: A[2, 8]
   poscondition: C[1, 5]
```

A lista abaixo descreve e exemplifica as regras derivadas a partir da restrição acima. A derivação das regras utiliza a sintaxe de declaração de predicados do *prolog* [9], que foi a linguagem utilizada para implementar a aplicação servidora Tucupi. Neste exemplo incluímos restrições temporais, a fim de ilustrar como tais restrições são representadas na derivação.

A divisão em restrições de ordem e restrições temporais aumenta a granularidade da definição das restrições. Apesar da ocorrência da restrição temporal implicar na ocorrência da restrição de ordem, a divisão em duas restrições pode permitir a violação da restrição temporal sem que a violação da restrição de ordem ocorra. No exemplo de restrição listado acima, poderíamos tolerar a violação da restrição temporal permitindo que a atividade $B$ seja executada mesmo depois de oito unidades de tempo após o fim da execução da atividade $A$, entretanto a ordem de execução, ou seja, primeiro a atividade $A$ e depois a atividade $B$, deve ser garantida.

A seguir, apresentamos a derivação do exemplo de restrição listado acima. A regras apresentadas a seguir são para um workflow exemplo, chamado *workflow_exemplo*. A variável *Case* representa uma instância de workflow.

**Regras de ativação**

As regras de ativação indicam o que fazer e quando fazer.

Ex.:

```
next(b, workflow_exemplo, Case) :-
   not(begin(_, b, workflow_exemplo, Case, T2)),
   end(a, workflow_exemplo, Case, T1),
   atLeast(T2, T1, 2, minute),
```

```
   atMost(T2, T1, 8, minute).

next(c, workflow_exemplo, Case) :-
   not(begin(_, c, workflow_exemplo, Case, T2)),
   end(b, workflow_exemplo, Case, T1),
   atLeast(T2, T1, 1, minute),
   atMost(T2, T1, 5, minute).
```

Neste exemplo duas regras de ativação foram derivadas. A primeira delas indica que a próxima atividade que pode ser executada será $B$ se $B$ ainda não começou e suas pre-condições foram satisfeitas. A segunda delas indica que a próxima atividade que pode ser executada será $C$ se $C$ ainda não começou e a atividade $B$ foi concluída satisfazendo as restrições temporais.

**Restrições de ordem**

As restrições de ordem indicam qual a ordem de execução entre duas atividades.

Ex.:

```
constraint(workflow_exemplo, Case, c01) :-
   not(override(c01, workflow_ab, Case)),
   not( beginx(b, workflow_exemplo, Case, T2) ->
      (end(a, workflow_exemplo, Case, T1),
       compare_time(<, T1, T2));
       true).
```

Neste exemplo a restrição $c01$ ocorrerá se ainda não foi violada e a atividade $A$ não terminou antes do início da atividade $B$.

**Restrições temporais**

As restrições temporais podem ter o mesmo significado que as restrições de ordem, mas adicionam a semântica de tempo, pois indicam em que momento uma atividade deve começar em relação a uma outra atividade.

Ex.:

```
constraint(workflow_ab, Case, c01) :-
   not(override(c01, workflow_ab, Case)),
   not( beginx(b, workflow_ab, Case, T2) ->
```

```
(end(a, workflow_ab, Case, T1),
(atMost(T2, T1, 8, minute);
atLeast(T2, T1, 2, minute)));
true).
```

As funções *compare_time*, *atMost* e *atLeast* utilizadas nos exemplos acima não são funções próprias do prolog. Estas funções pertencem ao componente de operações com tempo e data desenvolvido para a aplicação servidora Tucupi. Este componente possui um conjunto de funções que realizam operações temporais, entre as quais podemos listar:

**add_time**
: Adiciona a uma variável $T$ de tempo dada, um valor $V$ em unidades de tempo $U$ dado, ou seja, incrementa a variável $T$.

**sub_time**
: Subtrai de uma variável $T$ de tempo dada, um valor $V$ em unidades de tempo $U$ dado, ou seja, decrementa a variável $T$.

**compare_time**
: Compara duas variáveis de tempo *T1* e *T2*, indicando se as variáveis são menores, maiores ou iguais uma com relação a outra.

**atLeast**
: Indica se uma variável de tempo *T1* é pelo menos um valor $V$ em unidades de tempo $U$ menor a uma variável de tempo *T2*.

**atMost**
: Indica se uma variável de tempo *T1* é no máximo um valor $V$ em unidades de tempo $U$ maior a uma variável de tempo *T2*.

É importante observar que uma restrição temporal pode não ser derivada. Neste caso a restrição original, cuja derivação criaria a restrição temporal, não teria no corpo de sua declaração cláusulas que indiquem unidades de tempo. Para o exemplo acima teríamos:

```
rule: C01
   target: B
   precondition: A
   poscondition: C
```

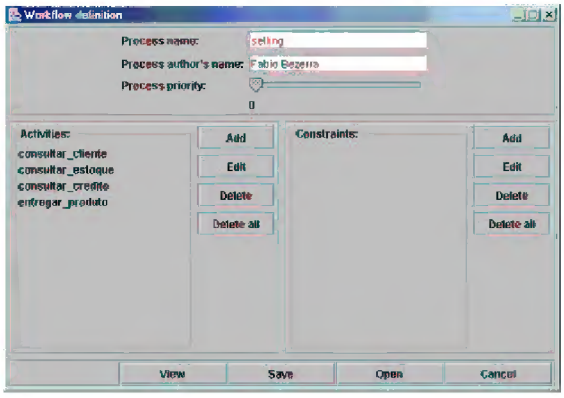


Figura 5.4: Interface principal da definição de um workflow.

### 5.3.3 Ferramenta de definição de processos

A ferramenta de definição de processos representa um componente independe do servidor Tucupi. Desenvolvida em Java, a ferramenta oferece ao usuário projetista do processo uma interface simples de definição de restrição. O usuário cria um processo através da definição das atividades e restrições que fazem parte deste processo (ver Figura 5.4).

A Figura 5.4 ilustra os atributos que fazem parte da definição de um workflow. O usuário deve fornecer algumas informações como nome, autor e prioridade do workflow. Além destes atributos o usuário deve fornecer a lista de atividades e restrições que compõem o workflow.

A definição das atividades é simples e requer apenas a definição dos nomes, enquanto que a definição das restrições requer do projetista definições relacionadas à atividade objetivo, às atividades obrigatórias (pré e pós-condições) e respectivas restrições temporais se houverem (ver Figura 5.5).

A Figura 5.5 ilustra os atributos que fazem parte da definição de uma restrição. A definição da restrição contempla a definição dos atributos especificados na linguagem de definição de restrição. Durante a definição da restrição o usuário deve selecionar a atividade objetivo e definir as atividades obrigatórias para execução da atividade objetivo. A definição de uma atividade obrigatória é realizada através da seleção da atividade, da configuração das restrições temporais se houverem e da definição do tipo de atividade forçada (*preconditions*, *postconditions* e *parconditions*). Após a definição, a atividade forçada é incluída em uma das listas de atividades obrigatórias que ficam na parte inferior da interface.

As Figuras 5.6 e 5.7 ilustram a interface que é ativada quando o usuário solicita a visualização do processo na interface principal da ferramenta de definição de processos

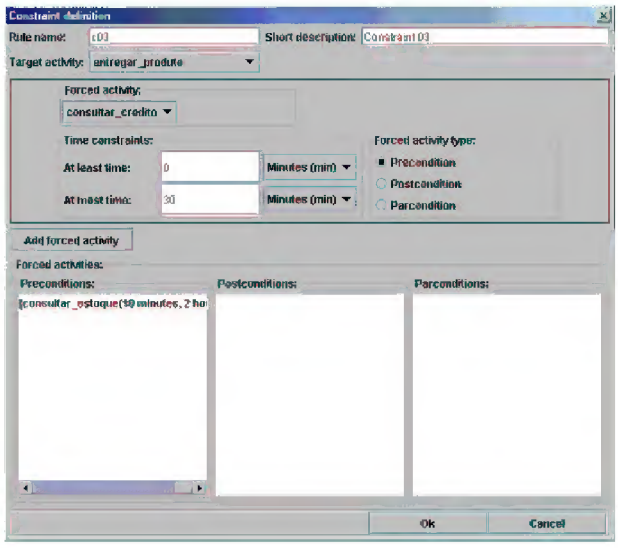


Figura 5.5: Interface de definição de uma restrição.

(botão *View*, Figura 5.4). A Figura 5.6 mostra a representação do processo como um conjunto de restrições, enquanto a Figura 5.7 mostra as regras em prolog derivadas a partir das restrições que fazem parte da definição do processo.

Outra maneira de definir o processo de forma interpretável pelo servidor Tucupi é através da edição direta em um arquivo texto das regras em prolog. Entretanto, as regras geradas em prolog por esta ferramenta simplificam o trabalho do projetista, que não precisa conhecer a sintaxe da linguagem prolog para editar as restrições e regras de ativação que fazem parte da definição do workflow.

## 5.4 Raciocínio temporal

Um sistema que possui suporte ao raciocínio temporal (do inglês, *temporal reasoning*) é capaz de executar a satisfação de restrições temporais. Embora o servidor Tucupi não execute a satisfação de restrições temporais de forma completa, um suporte limitado foi implementado a fim de operar com limites temporais entre as atividades.

O suporte ao raciocínio temporal implementado aqui refere-se à verificação da consistência das restrições temporais durante a execução de uma atividade. Por exemplo, quando um usuário tenta executar uma atividade que possui restrições temporais o servidor Tucupi verifica se essa execução viola alguma das restrições definidas. Durante a verificação algumas operações com data são executadas.

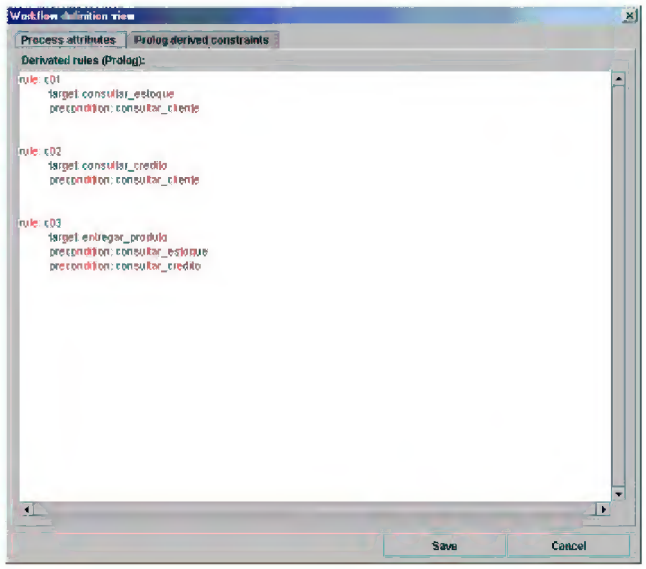


Figura 5.6: Visão do processo como um conjunto de restrições.

Outra característica de raciocínio temporal implementada no servidor Tucupi refere-se ao workflow visto como um ajudante, ou seja, durante a execução da função *what-if*.

### 5.4.1 Função *what_if*

Apresentaremos nesta seção algumas considerações a respeito da função *what-if* como implementada no servidor Tucupi. Para ilustrar a funcionalidade da função, considere o exemplo de restrição a seguir:

```
rule: c01
   target: B
   precondition: A [0, 5]

rule: c02
   target: C
   precondition: B [0, 10]
```

A restrição $c01$ indica que a atividade $A$ precisa ser executada até 5 unidades de tempo antes do início da atividade $B$, enquanto que a restrição $c02$ indica que a atividade $B$ precisa ser executada até 10 unidades de tempo antes do início da atividade $C$.

Esta informação é apresentada para o controlador do workflow como resposta à requisição $what\_if(c, caso_2, workflow\_exemplo)$. Entretanto, se durante o cálculo das

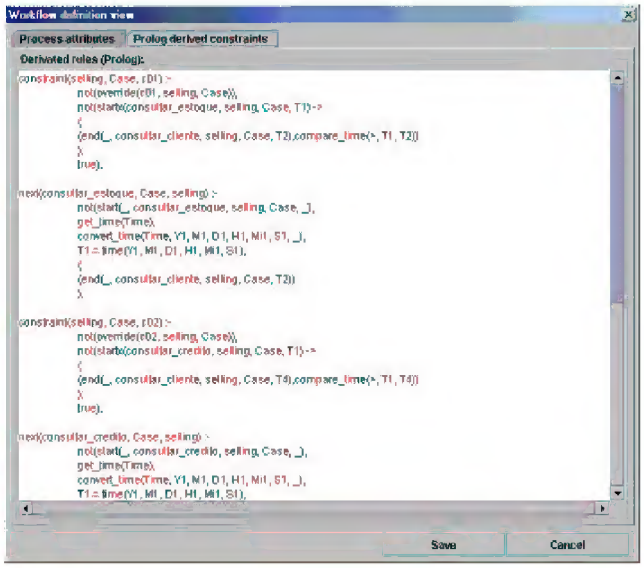


Figura 5.7: Visão das regras derivadas em prolog.

atividades obrigatórias para a atividade $C$ a atividade $B$ foi executada mais de 10 horas antes, a resposta será diferente. Neste caso, o sistema determinará que a atividade $B$ precisa ser executada novamente. Além disso, caso a atividade $B$ precise ser reiniciada, possivelmente as pré-condições de $B$ com limites temporais também precisem ser reiniciadas. Então, neste exemplo, o sistema incluirá na cadeia de pré-condições de $B$ apenas a atividade $A$.

A composição do conjunto de atividades que devem ser executadas antes da atividade indicada na função *what-if* é definida inicialmente através da determinação cadeia de pré-condições da atividade indicada na função *what-if*. Tal cadeia de pré-condições pode ser obtida através da equação de ponto fixo abaixo, tal que $predir(A)$ indica a função que determina as pré-condições direta da atividade $A$ que não foram executadas.

$$cadeia(A) = predir(A) \cup \bigcup_{a_i \in predir(A)} cadeia(a_i)$$

Entretanto, como mostrado no exemplo acima, é importante considerar as atividades concluídas que possuem restrições temporais, pois pode ser preciso executar algumas dessas atividades novamente. Isto ocorre quando a execução de uma atividade viola uma restrição temporal de sua pré-condição.

No caso da equação apresentada acima, bastaria modificar a função *predir*, que incluiria na resposta as atividades concluídas que são pré-condição da atividade indicada na função e que a execução da atividade indicada na função (no instante da consulta da função $what\_if$) viola alguma restrição temporal dessas atividades.

Entretanto a função what-if possui uma limitação. A limitação da solução implementada na função *what-if* refere-se à determinação da cadeia de pré-condições. Para a função ser executada a cadeia de pré-condições deve ser uma árvore. Por exemplo, considere o workflow abaixo definido como um conjunto de restrições:

```
rule: c01
   target: B
   precondition: A
   precondition: D[5]  % Pelo menos 5 unidades de tempo antes

rule: c02
   target: C
   precondition: B[0, 5] % No máximo 5 unidades de tempo antes
   precondition: D[10]   % Pelo menos 10 unidade de tempo antes
```

No workflow exemplo acima, adicionamos a cada pré-condição com restrição temporal, um comentário com o significado da restrição temporal representada.

A definição do workflow exemplo acima gera o grafo de dependência ilustrado na Figura 5.8. Os nós do grafo representam as atividades e as arestas orientadas indicam o sentido da dependência. Além disso, as arestas foram marcadas com as restrições temporais da definição acima.

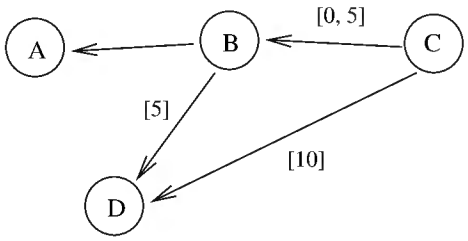


Figura 5.8: Grafo de dependência com restrições temporais.

Neste exemplo, caso assumíssemos que o tempo de execução de cada atividade do workflow fosse nulo, a atual implementação da função *what-it* seria capaz de determinar uma resposta para a atividade $C$, entretanto é difícil determinar o tempo de execução de

uma atividade em um workflow. O servidor Tucupi não é responsável pelo instante em que a atividade deve ser iniciada ou terminada, pois isto é função do usuário do sistema.

Portanto, satisfazer as restrições da definição tal que o intervalo de tempo entre as atividades $D$ e $C$ coincida com o intervalo de tempo T como definido pela equação abaixo, não é possível, pois não conhecemos o valor da função $tempo(B)$, que é o tempo de execução da atividade $B$. Nesta equação, a função *inicio* indica o instante que a atividade começou e a função $fim$ indica o instante que a atividade foi concluída.

$$T = (inicio(B) - fim(D)) + tempo(B) + (inicio(C) - fim(B))$$

Entretanto, se neste mesmo exemplo, não houvesse a dependência indicada na Figura 5.8 pela aresta entre as atividades $C$ e $D$ (ou seja, o grafo formado fosse uma árvore), não seria necessário considerar o tempo de execução entre as atividades para satisfazer as restrições temporais. Por isso afirmamos que a função *what-if* possui a execução limitada a grafos de dependência que são uma árvore.

# Capítulo 6

# Conclusão

A tecnologia de workflow é considerada atualmente como um campo promissor de pesquisas. Isto se deve pela promessa de automatização de processos de negócios que tais tecnologias fazem. Entretanto, os sistemas de workflow atuais não são adequados para processos que exigem maior flexibilidade na execução do workflow. A engenharia de software e a medicina são exemplos de domínios cujos processos não são modelados de forma adequada nos sistemas de workflow atuais.

Apresentamos neste trabalho o uso das restrições como uma proposta diferente de definir/modelar um workflow. Percebemos que o uso das restrições proporciona uma definição de workflow mais flexível, permitindo a uma única definição de workflow, diferentes estruturas de execução. Para tanto, definimos o modelo PDBC (Process Definition Based on Constraints).

Apresentaremos na Seção 6.1, a seguir, uma lista com as contribuições proporcionadas pelo desenvolvimento deste trabalho. Além das contribuições, incluimos neste capítulo uma análise crítica do trabalho (Seção 6.2), assim apresentamos as limitações do trabalho desenvolvido que também podem servir como sugestão para o desenvolvimento de outros trabalhos.

## 6.1 Contribuições

Entre as contribuições dessa dissertação podemos citar:

- a apresentação de uma solução para o problema da flexibilidade nos workflows através da especificação de uma linguagem de definição de workflows baseada em restrições;
- a definição de um mecanismo de tratamento de exceções nos workflows baseados em restrições através do conceito de restrições violáveis e a apresentação deste

mecanismo de tratamento de exceções como uma forma de se obter flexibilidade nos sistemas de workflow;

- a implementação de uma ferramenta de definição de workflows baseados em restrições;
- a especificação de restrições temporais nas restrições usadas na definição de um workflow;
- a implementação do servidor Tucupi, um protótipo de um sistema de workflow que utiliza workflows baseados em restrições como modelo de execução;
- e a implementação de um componente do servidor Tucupi que apoia de forma simplicada o raciocínio temporal, ou seja, a execução de restrições temporais.

## 6.2 Análise crítica e sugestões

A inclusão desta seção tem o intuito de apresentar as limitações de nosso trabalho. A partir da apresentação destas limitações futuras extensões podem ser discutidas a fim de promover futuras pesquisas na área de workflow.

A seguir apresentamos uma lista com as limitações do trabalho e sugestões para o desenvolvimento de outros trabalhos.

- A ferramenta de definição de processos apresentada não permite a definição de todas as construções possíveis definidas na liguagem de definição de restrição do modelo PDBC.
- A implementação usada para o servidor Tucupi não permite que mais de uma sessão seja criada simultaneamente. Ou seja, em um instante qualquer, apenas uma sessão é mantida com um cliente. Além disso, como apenas uma seção é considerada, nenhum mecanismo de sicronização de recursos foi implementado.
- Nenhuma avaliação de desempenho quanto às operações de atualização dos arquivos manipulados pelo servidor Tucupi foi realizada. Entretanto acreditamos que o mecanismo utilizado na atual implementação é ineficiente se comparado com um mecanismo que utilize algum banco de dados convencional.
- As atividades obrigatórias das restrições definidas no modelo PDBC fazem menção apenas a conclusão (pré-condição) ou ao início de uma atividade (pós-condição). Entretanto, acreditamos que outros atributos da atividade podem ser considerados, como por exemplo, o dado manipulado pela atividade. Assim poderíamos definir

restrições como: "o reembolso poderá ser realizado apenas se o pedido foi aprovado". Tal restrição considera que a execução da atividade *reembolsar* deve ser precedida pela atividade *aprovar*, mas o resultado da atividade *aprovar* deve ser *aprovado*.

- Nenhuma avaliação prática foi realizada com o modelo PDBC de forma a permitir uma melhor aferição dos ganhos com flexibilidade.

- A linguagem de restrição apresentada tem o objetivo apenas de especificar que atributos são importantes para definir uma restricão, portanto podemos considerar a linguagem apresentada como uma meta-linguagem. Assim, sugerimos o desenvolvimento de uma linguagem, com sintaxe e formalismos, para definição de workflows baseados em restrições.

- O suporte implementado pelo servidor Tucupi ao raciocínio temporal é limitado. Entretanto, ainda não sabemos como trabalhar com as restrições temporais nos workflows, ou seja, precisamos elaborar mais os requisitos de raciocínio temporal que devem ser implementados no servidor Tucupi.

- Tanto o servidor Tucupi como a ferramenta de definição de processos não utilizam nenhum mecanismo de verificação da corretude da definição de um workflow. Citamos abaixo um exemplo simples de declaração incorreta que não é detectado pelo servidor Tucupi ou pela ferramenta de definição de processos:

```
rule: c1
  target: B
  precondition: A

rule: c2
  target: A
  precondition: B
```

# Referências Bibliográficas


[1] *Role Based Access Control: Features and Motivations*, IEEE Computer Society Press, New Orleans, Louisianna, December 1995.

[2] *Workflows in Dynamic Environments - Can they be managed?*, Woollongong, Australia, March 1999.

[3] W. M. P. Van Der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317, 2001.

[4] Vinícius Amaral, Anderson Santos Grala e José Valdini Lima. Workflow e gerência de documentos. Em *XIV Jornada de Atualização em Informática*, Brasília, DF, agosto 1997. SBC.

[5] Naser S. Barghouti. Supporting cooperation in the marvel process-centered sde. Em Herbert Weber, editor, *Fifth ACM SIGSOFT Symposium on Software Development Environments*, volume 17 de *Special issue of Software Engineering Notes*, páginas 21–31, Tyson's Corner VA, December 1992.

[6] John Barkley. Comparing simple role-based access control models and access control lists. páginas 127–134, 1997.

[7] P. Barthelmess e J. Wainer. Workflow systems: a few definitions and a few suggestions. Em N. Comstock e C.A. Ellis, editores, *Proc. of the Conference on Organizational Computing Systems - COOCS'95*, páginas 138–147, Milpitas, California, September 1995. ACM Press.

[8] Grandy Booch, James Rumbauch e Ivar Jacobson. *UML Guia do Usuário.* Campus, Rio de Janeiro, 2000. Tradução de: The Unified Modeling Language User Guide.

[9] Ivan Bratko. *Prolog Programing for Artificial Inteligence.* International Computer Science Series. Addison-Wesley, 1990. 2nd Edition.

[10] Amedeo Cesta e Angelo Oddi. Gaining efficiency and flexibility in the simple temporal problem. Em *Workshop on Temporal Representation and Reasoning*, Florida, May 1996.

[11] Paul Dourish, Jim Holmes, Allan MacLean, Pernille Marqvardsen e Alex Zbyslaw. Freeflow: mediating between representation and action in workflow systems. Em *Conference on Computer Supported Cooperative Work CSCW96*. ACM, November 1996.

[12] Clarence Ellis e Karim Keddara. Ml-dews: Modeling language to support dynamic evolution within workflow systems. *Computer Supported Cooperative Work*, 9(3-4):293–333, 2000.

[13] Clarence Ellis, Karim Keddara e Grzegorz Rozenberg. Dynamic change within workflow systems. Em *Proceedings of conference on Organizational computing systems*, páginas 10–21. ACM Press, 1995.

[14] George T. Heineman, Gail E. Kaiser, Naser S. Barghouti e Israel Ben-Shaul. Rule chaining in MARVEL: Dynamic binding of parameters. Em *KBSE*, páginas 215–222, 1991.

[15] Petra Heinl, Stefan Horn, Stefan Jablonski, Jens Neeb, Katrin Stein e Michael Teschke. A comprehensive approach to flexibility in workflow management systems. Em *Proceedings of the international joint conference on Work activities coordination and collaboration*, páginas 79–88. ACM Press, 1999.

[16] David Hollingsworth. Workflow management coalition the workflow reference model, Janeiro 1995. Document Number TC00-1003.

[17] P. J. Mangan e S. Sadiq. A constraint specification approach to building flexible workflows. *Journal of Research and Practice in Information Technology*, 2002.

[18] Peter Mangan e Shazia Sadiq. On building workflow models for flexible processes. Em *The Thirteenth Australasian Database Conference ADC2002*, 2002.

[19] M. Nyanchama e S. Osborn. *The role graph model and conflict of interest. ACM Transaction on Information and System Security*, 1999.

[20] Sylvia Osborn. Mandatory access control and role-based access control revisited. páginas 31–40.

[21] C. Ramaswamy e R. Sandhu. Role-based access control features in commercial database management systems. Em *Proc. 21st NIST-NCSC National Information Systems Security Conference*, páginas 503–511, 1998.

[22] Rodrigo Quites Reis, Carla Alessandra Lima Reis, Heribert Schlebbe e Daltro José Nunes. Automatic verification of static policies on software process models. *Annals of Software Engineering*, (14):197–234, 2002.

[23] Shazia Sadiq, Wasim Sadiq e Maria Orlowska. Pockets of flexibility in workflow specification. Em *International Conference on Conceptual Modeling*, Yokohama, Japan, 2001.

[24] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein e Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[25] Marc Thomas Schmidt. The evolution of workflow standards. *IEEE Concurrency*, páginas 44–52, July-September 1999.

[26] R. Simon e M. E. Zurko. Separation of duty in role-based environments. Em *IEEE Computer Security Foundations Workshop*, páginas 183–194, 1997.

[27] M. Voorhoeve e W. van der Aalst. Ad-hoc workflow: Problems and solutions. Em *International Workshop on Database and Expert Systems*, Tolouse, France, September 1997.

[28] Jacques Wainer. Logic representation of processes in work activity coordination. Em *Proceedings of the ACM Symposium on Applied Computing, Coordination Track*, volume 1, páginas 203–209. ACM, ACM Press, March 2000.

[29] Jacques Wainer, Paulo Barthelmess e Akhil Kumar. *WRBAC* - A workflow security model incorporating controlled overriding of constraints. Technical report, Universidade Estadual de Campinas, Dezembro 2001.

[30] Jacques Wainer, Paulo Barthelmess e Akhil Kumar. W-RBAC: a workflow secutiry model incorporating controlled overriding of constraints. submitted, 2003.

[31] Jacques Wainer e Fábio Bezerra. Constraint-based flexible workflows. Em Dominique Decouchant e Jesus Favela, editores, *Proc. of the 9th International Workshop on Groupware – CRIWG2003*, Lecture Notes in Computer Science, LNCS. Springer-Verlag, Setembro 2003. Aceito para publicação.

[32] Document Number Wfmc-Tc-1011. Workflow management coalition terminology glossary, 1999.