WorkToDo – Um Sistema de Gerenciamento de Workflows para Ambientes de Comunicação sem Fio

Leonardo Hartleben Reinehr

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

WorkToDo – Um Sistema de Gerenciamento de Workflows para Ambientes de Comunicação sem Fio

Leonardo Hartleben Reinehr

15 de Outubro de 2002

Banca Examinadora:

- Dra. Maria Beatriz Felgar de Toledo
  IC/UNICAMP – Universidade Estadual de Campinas (Orientadora)
- Dr. Antonio Alfredo Ferreira Loureiro
  DCC/UFMG – Universidade Federal de Minas Gerais
- Dr. Edmundo Roberto Mauro Madeira
  IC/UNICAMP – Universidade Estadual de Campinas
- Dra. Cláudia Bauzer Medeiros (Suplente)
  IC/UNICAMP – Universidade Estadual de Campinas

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Reinehr, Leonardo Hartleben

R275w WorkToDo – Um sistema de gerenciamento de workflows para ambientes de comunicação sem fio / Leonardo Hartleben Reinehr -- Campinas, [S.P. :s.n.], 2002.

Orientador : Maria Beatriz Felgar de Toledo

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

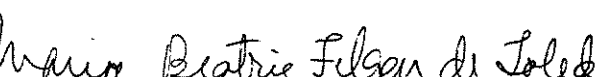
1. Computação móvel. 2. CORBA (Arquitetura de computador). 3. Fluxo de trabalho. I. Toledo, Maria Beatriz Felgar de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 29 de julho de 2002, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Antonio Alfredo Ferreira Loureiro
DCC - UFMG

Prof. Dr. Edmundo Roberto Mauro Madeira
IC - UNICAMP



Profª. Drª. Maria Beatriz Felgar de Toledo - Orientadora
IC - UNICAMP

WorkToDo – Um Sistema de Gerenciamento de Workflows para Ambientes de Comunicação sem Fio

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Leonardo Hartleben Reinehr e aprovada pela Banca Examinadora.

Campinas, 15 de Outubro de 2002.

Dra. Maria Beatriz Felgar de Toledo
IC/UNICAMP – Universidade Estadual de Campinas (Orientadora)

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

Resumo

Os Sistemas de Gerenciamento de *Workflows* (SGWFs) tradicionais somente podem ser utilizados a partir de conexões permanentes, de alta velocidade e confiáveis. Essa é uma restrição que torna-se ainda mais severa ao considerarmos a crescente utilização da computação móvel e das redes sem fio, onde as conexões são instáveis e não permanentes. Nesta dissertação propomos o sistema WorkToDo, um SGWF para ambientes de comunicação sem fio. O WorkToDo permite que um usuário execute tarefas independentemente de sua localização e tipo de conexão, preservando a autonomia do computador móvel. Além disso, os usuários podem executar tarefas mesmo sem estarem conectados ao SGWF. Para oferecer essa flexibilidade de operação o sistema se baseia em mecanismos como trancamento de tarefas, atribuição de tarefas no momento da conexão dos usuários, armazenamento local de dados de tarefas e transferência antecipada de dados e aplicações para o computador móvel.

Abstract

Traditional Workflow Management Systems (WFMS) may only be used from permanent, high-speed, trustworthy connections. This is a restriction that becomes even more severe with the increasing use of mobile computing and wireless networks, where connections are unstable and non-permanent. In this dissertation we propose the WorkToDo system, a WFMS for wireless communication environments. The WorkToDo allows a user to execute tasks independently from location and type of connection, preserving the autonomy of the mobile computer. Moreover, users can execute tasks even whithout being connected to the WFMS. In order to allow this operation flexibility, the system is based on mechanisms such as task locking, task assignment at user connection time, local storage of task data and anticipated transfer of data and applications to the mobile computer.

Agradecimentos

Aos meus pais, Paulo e Ivone, e meu irmão Leandro, que mesmo estando longe me ajudaram a chegar até aqui. Obrigado por serem a melhor família que eu poderia ter.

A Dani, minha querida namorada, por tornar os últimos sete meses do mestrado muito mais felizes e por sempre me dar apoio e incentivo. Te amo!

Ao Luciano e a Lo, que apesar de agora não estarem mais por perto (infelizmente), nos últimos dois anos estiveram sempre presentes e se tornaram meus melhores amigos. Valeu por todos os finais de semana!

Ao pessoal de casa (o CAOS!), pelas festas, churrascos, jogos de futebol e tantas outras coisas sem as quais viver em Campinas não teria muita graça. Valeu Carlos "Doidão", Guilherme "Gordo", Léo "Shark", Nadim "Projetos" e Rodrigo "Virjão", por serem grandes amigos e por me agüentarem. Agora acabou!

Aos amigos que fiz dentro da Unicamp, principalmente Fábio, Hudo, Vanessa e Adriane, por ajudarem durante as disciplinas e por gostarem de fazer outras coisas além de estudar. Valeu Fábio por tirar muitas dúvidas e Hudo por trocar idéias sobre o projeto.

À minha orientadora Beatriz, pelo auxílio durante o desenvolvimento do projeto e a escrita da dissertação e dos dois artigos, pelas idéias e críticas sempre construtivas, e por saber – ao contrário de muitos orientadores – ser compreensiva e humana.

E principalmente a Deus, por me ajudar em todos os momentos do mestrado (e da vida) e me permitir alcançar mais essa conquista. Sem Ele ninguém é nada.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Sistemas de Gerenciamento de *Workflows* (SGWFs) têm provocado grande interesse devido à sua capacidade de aumentar a eficiência de uma organização através da automatização de seus processos [4, 13]. Um SGWF coordena e gerencia a execução de processos empresariais, assumindo as funções mecânicas e burocráticas e permitindo que os usuários se concentrem em aspectos mais importantes, como gerenciamento de informações e análise de resultados.

Diversos SGWFs foram implementados comercialmente e passaram a ser utilizados com sucesso nas empresas, alcançando significativas melhoras de eficiência. Entretanto, os SGWFs existentes ainda apresentam limitações como, por exemplo, baixa interoperabilidade entre diferentes SGWFs, suporte insuficiente para recuperação de falhas e pouca flexibilidade de uso [5, 11, 12, 14, 21, 22, 34].

Normalmente os SGWFs exigem que os usuários possuam uma conexão permanente por meio de uma rede local (Ethernet, por exemplo). Esta é uma restrição severa, pois obriga os usuários a utilizarem o sistema somente em um determinado local. Considerando a grande disseminação das redes sem fio e da computação móvel, é interessante permitir que os usuários utilizem o SGWF sem restrições de localização e sem exigências de conexões permanentes, confiáveis e de alta velocidade. Isso expande sensivelmente a flexibilidade de uso de um SGWF, pois os usuários podem executar as tarefas que lhes cabem a partir de seus *laptops*, PDAs e demais dispositivos portáteis, independentemente de sua localização e do tipo de sua conexão ao SGWF.

Consideremos o seguinte exemplo. Uma empresa que presta serviços de manutenção para diversos clientes espalhados em uma região possui um processo gerenciado por um SGWF, composto por diversas tarefas, entre elas `VisitarCliente`. Essa tarefa é executada por um técnico de manutenção, que a partir de uma ordem de serviço se desloca até

o cliente e executa o serviço de manutenção, informando depois o SGWF a respeito dos serviços executados e dos materiais utilizados.

Em um SGWF convencional, no início do dia o técnico reúne as ordens de serviço atribuídas a ele e visita cada cliente. Ao visitar um cliente, após descobrir qual é o problema (por exemplo, uma peça p de um equipamento está quebrada), o técnico liga para o setor de estoque da empresa para verificar a disponibilidade de p. O estoque é então consultado: se a peça não está disponível, o técnico requisita sua compra. Então, quando a peça torna-se disponível, o técnico retorna ao cliente e completa o serviço. Além de necessitar de um atendente para efetuar as consultas ao estoque para os técnicos, com esse tipo de SGWF é necessário que o técnico periodicamente verifique junto ao setor de estoque se a peça já está disponível.

Por outro lado, se o SGWF oferece suporte à mobilidade, quando o técnico detecta que a peça p está quebrada, ao invés de ligar para a empresa ele se conecta ao SGWF (usando um *laptop* e um telefone celular, por exemplo) e consulta o sistema sobre a disponibilidade de p. Se a peça não é encontrada em estoque, o técnico inicia um outro processo para a compra de p. A partir desse ponto o técnico pode, a qualquer hora e em qualquer lugar (até mesmo enquanto estiver se deslocando de um cliente para outro), verificar o andamento desse processo (se a ordem de compra já foi autorizada, se a peça está a caminho, e assim por diante), sendo notificado automaticamente no momento em que o processo for completado e a peça estiver disponível. Quando isso ocorre, o técnico retorna ao cliente com a nova peça e completa o serviço.

Esse exemplo mostra como a operação através de redes sem fio expande a flexibilidade de uso de um SGWF, além de aumentar sua eficiência e agilidade. Entretanto, com esse tipo de operação o gerenciamento de processos torna-se mais complexo. Quando o usuário está sempre conectado ao SGWF, pode ser constantemente monitorado e pode receber notificações de novas tarefas a qualquer momento. Já em um ambiente de redes sem fio isso não é possível, pois o usuário pode permanecer longos períodos desconectado do sistema. Além disso, falhas de comunicação entre usuários e sistema ocorrem com uma freqüência muito maior.

Por esses motivos, novos mecanismos de controle se fazem necessários, a fim de que o SGWF respeite a autonomia dos usuários mas ao mesmo tempo garanta que os processos sejam executados de acordo com sua especificação. A flexibilidade de uso deve ser aumentada tanto quanto possível, mas sem comprometer a execução correta dos processos.

1.2 Objetivos

Nesta dissertação apresentamos o WorkToDo, um SGWF para ambientes de comunicação sem fio. O WorkToDo permite que os usuários executem tarefas através de conexões

instáveis e de baixa largura de banda, de forma semelhante como quando conectados a uma rede local. Mais do que isso, os usuários podem, tomando certas medidas, executar tarefas mesmo sem estarem conectados ao SGWF.

O principal objetivo é aumentar a flexibilidade de uso do SGWF, permitindo que usuários tenham acesso ao sistema independentemente de localização ou tipo de conexão. Para isso, são utilizadas técnicas como: trancamento de tarefas[1], *prefetching* de dados, notificação de tarefas para os usuários no momento de sua conexão, execução de tarefas sobre dados locais, armazenamento local de resultados e propagação de resultados após a reconexão.

É também apresentado um protótipo, desenvolvido utilizando Java e CORBA, para validar e avaliar o modelo e a arquitetura do WorkToDo.

1.3 Estrutura da Dissertação

Este documento está organizado da seguinte forma:

- O Capítulo 2 apresenta os conceitos necessários para o entendimento da dissertação, relacionados a Sistemas de *Workflow* e a ambientes de comunicação sem fio;
- O Capítulo 3 descreve o modelo e a arquitetura do WorkToDo, descrevendo suas características e seu funcionamento;
- No Capítulo 4 são descritos aspectos relativos ao protótipo implementado, além de um exemplo de utilização do sistema;
- O Capítulo 5 mostra alguns trabalhos relacionados, destacando suas características mais relevantes e comparando-os ao WorkToDo;
- Finalmente, o Capítulo 6 apresenta as conclusões do trabalho, juntamente com alguns trabalhos futuros.

[1] Do termo em inglês *lock*

Capítulo 2

Conceitos Básicos

Este Capítulo introduz os conceitos básicos relacionados a Sistemas de *Workflow*, além de um breve histórico sobre o assunto. Apresenta também um resumo do padrão CORBA, juntamente com as características principais do OrbixWeb, a implementação de CORBA utilizada no desenvolvimento do protótipo. O Capítulo se encerra com uma visão geral a respeito de computação sem fio.

2.1 Sistemas de Workflow

Um Sistema de Gerenciamento de *Workflows* (SGWF) é um sistema que define, gerencia e executa *workflows* de forma automatizada [39]. As entidades responsáveis por cada tarefa cooperam para a conclusão do *workflow* através da execução de suas tarefas individuais, sendo coordenadas pelo SGWF, que atribui tarefas às entidades, coleta os resultados, determina as próximas tarefas a serem executadas, controla a operação de cada entidade e detecta quando o *workflow* foi concluído com sucesso.

SGWFs são inerentemente distribuídos, já que as entidades não estão reunidas em um único local, podendo estar espalhadas dentro de um prédio, uma cidade, um país, ou até mesmo ao redor do mundo.

Os sistemas de *workflow* são relativamente jovens, tendo surgido no início da década de 90. Entretanto, diversas idéias e conceitos relativos a *workflow* são bem anteriores, tendo começado a surgir a partir da década de 70, nas áreas de automação de escritórios, trabalho cooperativo e gerenciamento de documentos. Posteriormente foram acrescentados também conceitos das áreas de gerenciamento de bancos de dados e processamento de transações.

O próprio termo *workflow* surgiu dentro das empresas, referenciando um processo de negócios que continha um conjunto de atividades. O gerenciamento da execução desses *workflows*, entretanto, era efetuado por uma pessoa e não por um sistema automatizado.

O advento de sistemas de *workflow* foi impulsionado devido, principalmente, a dois fatores [19]:

Disponibilidade de tecnologias. Muitas das tecnologias necessárias a um SGWF só se tornaram disponíveis e viáveis há pouco tempo. É o caso das redes de computadores, que permitem a interconectividade entre computadores. Além de um aumento de velocidade e confiabilidade das redes locais, houve também um aumento de alcance através da Internet e das redes sem fio. Atualmente é possível trocar informações entre computadores de qualquer localidade. Isso provocou um desenvolvimento de aplicações e sistemas distribuídos, os quais alcançaram maturidade e tornaram-se largamente utilizados.

Mudança na visão das empresas. Nos últimos anos houve uma mudança no objetivo das empresas em relação a sistemas de software. Enquanto antes o objetivo era automatizar determinadas tarefas isoladas, agora é automatizar e gerenciar processos de forma completa. Isso gerou como conseqüência uma mudança no desenvolvimento de software, de sistemas que automatizam atividades isoladas para sistemas que apresentam soluções integradas e abrangentes.

Quando ocorreram estas mudanças e avanços os sistemas de *workflow* passaram a ser explorados, implementando na prática os conceitos já existentes. Atualmente existem inúmeros SGWFs no mercado, como por exemplo: MQSeries (IBM), Enterprise Workflow (eiStream), ActiveWorkflow (Singularity), BizFlow (HandySoft), WorkMovr (A-Frame), Process Suite (Staffware) e muitos outros [40].

Os SGWFs são utilizados em diversos domínios e cenários, entre eles: controle de pagamentos de seguros, pedidos de financiamento, acompanhamento de pacientes em hospitais, automação de processos logísticos, planejamento de viagens, transações bancárias, gerenciamento de telecomunicações e planejamento de produção.

A seguir serão introduzidos os conceitos relacionados a sistemas de *workflow*. A Seção prossegue mostrando duas classificações para SGWFs e se encerra apresentando o Modelo de Referência para *Workflow*.

2.1.1 Processo de Negócios

Um processo de negócios é definido como um conjunto de tarefas organizadas de tal forma que executam uma determinada atividade [5]. Um processo de negócios pode ser qualquer atividade dentro de uma empresa, como por exemplo:

1. A compra de produtos para atualização do estoque de uma loja;

2. A reserva de passagens em empresas aéreas e estadias em hotéis de acordo com um roteiro de viagem;
3. A abertura de uma nova conta em um banco;
4. A instalação de uma linha telefônica por uma companhia telefônica.

Todas as atividades acima podem ser divididas em tarefas menores, de forma que, ao final da execução das tarefas, a atividade está concluída. Por exemplo, o processo 4 pode ser dividido nas seguintes tarefas: preencher um formulário com os dados do cliente, cadastrar o cliente no sistema de controle, verificar disponibilidade de linhas na região do cliente, enviar equipe para instalação da linha e efetuar teste da linha. Após o teste da linha, o usuário já pode utilizá-la, portanto o processo está concluído.

Os processos de negócios geralmente são de longa duração e envolvem muitas entidades presentes em diversas localidades diferentes.

2.1.2 Tarefa

As tarefas são cada um dos passos que devem ser executados para que um processo de negócios seja concluído. Um processo de negócios normalmente contém diversas tarefas, dos mais variados tipos e com diferentes tempos de execução. As tarefas podem ser executadas por humanos, por sistemas de software ou por ambos.

Exemplos de tarefas incluem:

1. Atualizar um registro em uma base de dados;
2. Gerar um recibo;
3. Instalar um dispositivo;
4. Preencher um formulário;
5. Efetuar backup de um conjunto de arquivos;
6. Imprimir um documento;
7. Realizar uma ligação telefônica.

2.1.3 Workflow

Um *workflow* (ou processo de *workflow* ou simplesmente processo) é a representação computacional de um determinado processo de negócios [39]. Um *workflow* define toda a estrutura de um processo de negócios:

Tarefas. O conjunto de ações que devem ser desempenhadas para que o processo seja completado.

Responsáveis. As entidades responsáveis pela execução de cada tarefa. As entidades podem ser um aplicativo ou um humano. Pode existir um mesmo responsável para diversas tarefas, bem como diversos responsáveis para uma determinada tarefa.

Seqüência de execução. A seqüência em que as tarefas devem ser executadas. No exemplo de processo mostrado anteriormente, a tarefa "efetuar teste da linha" não pode ser executada antes da tarefa "enviar equipe para instalação da linha". Então, algum tipo de seqüenciamento se faz necessário.

Fluxo de dados. As informações geradas por uma tarefa podem ser necessárias para tarefas posteriores. Assim sendo, é necessário especificar como essas informações são passadas de tarefa para tarefa.

2.1.4 Instância de Workflow

Uma instância de *workflow* (ou instância de processo) é uma execução particular de um determinado *workflow*, com parâmetros e dados próprios.

Podem existir diversas instâncias de um mesmo *workflow* em execução simultaneamente. Por exemplo, para o *workflow* abertura de uma conta em um banco, pode existir uma instância para o cliente X e outra para o cliente Y.

2.1.5 Classificação

Existem diversas classificações para sistemas de *workflow*, entretanto a mais utilizada agrupa os SGWFs de acordo com a estrutura e complexidade dos *workflows* por ele gerenciados [6]:

Administrativos. Referem-se a processos nos quais as tarefas a serem executadas são bem determinadas e seguem um conjunto de regras conhecido por todos os envolvidos em sua execução. A estrutura desse tipo de *workflow* tende a não variar muito ao longo do tempo. Exemplos de processos administrativos incluem: registro de um veículo, matrícula de disciplinas em uma universidade e demais processos nos quais existe um conjunto de formulários a serem preenchidos durante a execução de cada tarefa.

Ad Hoc. São similares aos *workflows* administrativos, com a diferença de que são capazes de lidar com exceções e casos especiais, onde determinada instância pode exigir um tratamento único. Um exemplo é um processo de submissão de artigos para revistas, onde são executados diferentes protocolos dependendo de cada revista.

Colaborativos. Sua característica principal é a presença de um grande número de participantes que colaboram para a execução das tarefas. Ao contrário dos outros tipos de *workflow*, onde há sempre uma execução seqüencial das tarefas, nos *workflows* colaborativos podem existir diversas iterações sobre uma mesma tarefa até que um consenso seja atingido. Além disso, podem ocorrer retornos a tarefas anteriores, já executadas. Um exemplo é a escrita de um artigo por diversos autores de forma cooperativa, ou a revisão de um texto ou livro por vários editores. *Workflows* colaborativos tendem a ser muito dinâmicos, pois são redefinidos constantemente à medida que são executados. Isso exige que a especificação do processo seja extremamente flexível, ou em alguns casos até mesmo inexistente, deixando a coordenação da execução do processo nas mãos dos próprios participantes. Nesse caso o sistema toma poucas (ou nenhuma) decisões, servindo apenas como uma interface pela qual os participantes podem registrar os resultados.

De Produção. São caracterizados como a implementação de processos de negócios críticos, ou seja, aqueles processos que estão diretamente relacionados à função da empresa. São *workflows* administrativos que consideram aspectos como execução em larga escala, alta complexidade dos processos e heterogeneidade de ambiente, além de variedade de participantes envolvidos e tarefas executadas. Por isso, tendem a ser executados em grandes empresas, onde esses fatores estão presentes com maior freqüência. Exemplos são controle de pagamentos de seguros e pedidos de empréstimo.

Uma outra classificação também bastante considerada, apresentada por Jablonski e Büssler [19], agrupa os SGWFs de acordo com a tecnologia utilizada para fornecer suporte à execução dos *workflows*:

Centrados em E-mail. São baseados em correio eletrônico, podendo ser associados a SGWFs que gerenciam *workflows* ad hoc e colaborativos. Dadas as características do correio eletrônico, tais sistemas não são adequados para gerenciamento de *workflows* de produção ou para ambientes com um grande número de processos concorrentes.

Centrados em Documentos. Baseiam-se apenas na idéia de troca de documentos entre as tarefas, possuindo possibilidades limitadas de interação com aplicações externas. São adequados para a execução de *workflows* administrativos, que se baseiam em formulários.

Centrados em Processos. Correspondem aos SGWFs que gerenciam *workflows* de produção. Geralmente implementam mecanismos de comunicação próprios, utilizam bancos de dados e fornecem meios de interação com aplicações externas.

2.1.6 Modelo de Referência para Workflow

Com a maior utilização de Sistemas de Gerenciamento de *Workflows*, muitos produtos surgiram no mercado, cada um com suas próprias características e funcionalidades. Como não existia nenhum tipo de padronização definida, esses diferentes sistemas se tornaram incapazes de se comunicar uns com os outros e de operar em conjunto para executar um determinado processo.

O ***Modelo de Referência para Workflow*** [38], definido pela *Workflow Management Coalition* (WfMC[1]) [39], consiste de uma série de especificações e padrões para a construção de SGWFs, com o objetivo de que estes possam interoperar.

O modelo possui três áreas funcionais básicas, conforme a Figura 2.1:

Funções de Definição. Dizem respeito à definição e modelagem de um processo e das atividades que o compõem. Estas funções são necessárias para converter um processo do mundo real para um formato computacional, denominado ***definição de processo***.

Funções de Controle de Execução. Preocupam-se com a interpretação e execução de processos, bem como com a coordenação e escalonamento da execução das várias atividades de cada processo. Estas funções são providas pela chamada ***Máquina de Workflow***.

Interações de Execução. Referem-se às interações do SGWF com usuários e também com outras aplicações, interações estas necessárias para o processamento das atividades.

O modelo define uma arquitetura básica, identificando seus componentes principais de acordo com as áreas descritas acima, juntamente com as funcionalidades de cada componente. Além disso, o modelo define um conjunto de interfaces entre os componentes, através das quais sistemas de *workflow* que implementam de forma distinta as funções dos componentes podem se comunicar.

Os componentes do modelo (Figura 2.2) são os seguintes:

Serviço de Execução. Provê o ambiente de execução para os processos, através de uma ou mais máquinas de *workflow*. É responsável pela criação, gerenciamento e execução dos processos, seqüenciando a execução das tarefas, invocando as aplicações e controlando a interação com os usuários.

[1] A WfMC é uma organização criada em agosto de 1993, composta por cerca de 285 empresas fabricantes e usuárias de sistemas de *workflow*, entre elas IBM, Lucent, Microsoft, Sun e Toshiba.

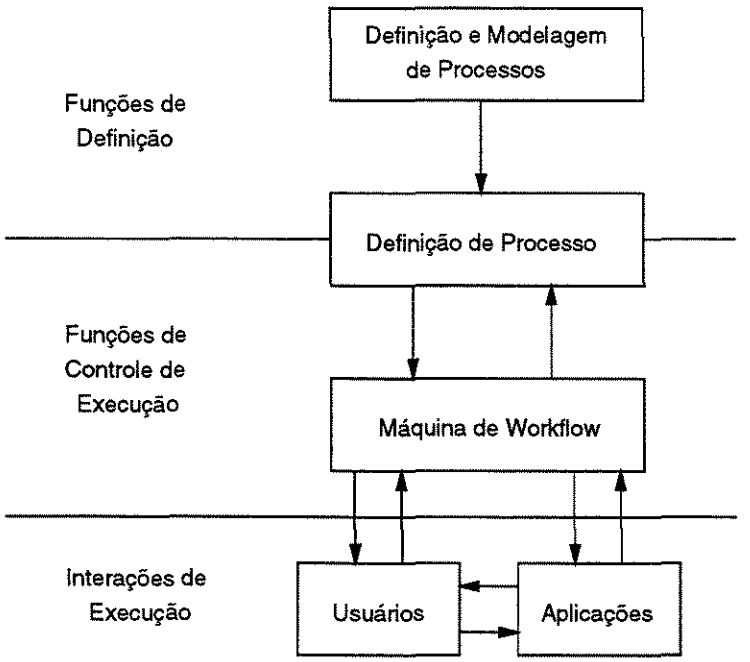


Figura 2.1: Áreas funcionais básicas do Modelo de Referência para *Workflow*.

Ferramentas de Definição de Processo. São ferramentas para gerar a definição do processo a ser executado. Esta definição contém informações a respeito das atividades que compõem o processo e das regras para a execução das atividades, entre outras.

Aplicações Cliente de Workflow. Gerenciam a interação entre os usuários participantes da execução de um processo e o Serviço de Execução. Interagem com os usuários naquelas atividades que necessitam de ação humana, através da chamada ***lista de trabalho*** – uma lista de tarefas atribuídas pela máquina de *workflow* a um determinado usuário.

Aplicações Invocadas. A execução de uma ou mais aplicações pode ser necessária para completar a execução de determinadas tarefas.

Outros Serviços de Execução. Diferentes Serviços de Execução podem ter de operar em conjunto para a execução de um mesmo processo.

Ferramentas de Administração e Monitoramento. Provê serviços para alterar regras de alocação de trabalho, identificação dos participantes que podem desempenhar cada papel dentro de um processo, visualização do histórico de execução de um

processo ou atividade, entre outros. Estes serviços são disponíveis a determinados usuários, os chamados supervisores ou administradores.

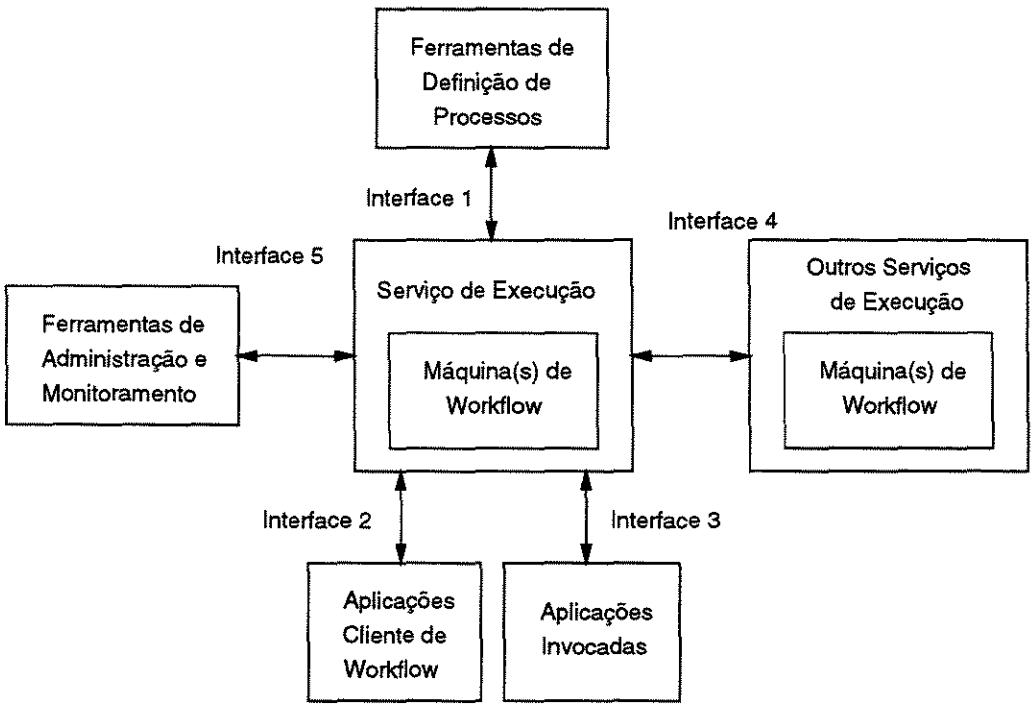


Figura 2.2: Componentes e Interfaces do Modelo de Referência para *Workflow*.

O componente Serviço de Execução interage com todos os demais. Assim, o modelo de referência define cinco interfaces, entre o Serviço de Execução e cada um dos demais componentes. Nestas interfaces são definidas as formas de comunicação entre cada componente e também como ocorre a transferência de dados e informações entre eles. Isso garante que dois SGWF construídos respeitando as definições das interfaces serão capazes de interoperar, independentemente de suas implementações.

2.2 CORBA

O *Object Management Group* (OMG) [27] é um consórcio composto atualmente por mais de 800 empresas, criado em 1989 com o objetivo de definir padrões que permitissem a interoperabilidade de componentes de software em um ambiente distribuído, independentemente de sua localização, plataforma, sistema operacional e linguagem de programação [26, 31, 35].

A pesquisa do OMG originou, em 1991, o padrão CORBA (*Common Object Request Broker Architecture*), tendo seguimento com o CORBA 2 [26], lançado em 1996. A versão mais atual do padrão é o CORBA 2.6, lançado em Dezembro de 2001, embora o CORBA 3 esteja para ser lançado.

A especificação CORBA [10] define a arquitetura de um *Object Request Broker* (ORB), uma camada de software que implementa a interoperabilidade proposta pelo OMG. O ORB fornece mecanismos pelos quais objetos clientes podem requisitar serviços de objetos servidores de forma transparente [26]. Para tanto, o conjunto de serviços oferecidos pelos servidores devem estar definidos através de uma linguagem chamada IDL (*Interface Definition Language*). Através dessa interface o cliente pode então requisitar as operações junto ao servidor.

Os servidores devem fornecer a implementação de cada operação descrita na IDL, em qualquer linguagem de programação para a qual a IDL possa ser mapeada (atualmente C, C++, Java, Smalltalk, ADA, COBOL, LISP e Python). A especificação em IDL é convertida para a linguagem escolhida através de um compilador, que gera os *stubs* e *skeletons*.

Os clientes utilizam os *stubs* para efetuar as invocações remotas para o servidor. O *stub* invoca então as operações do *skeleton*, que encaminha as invocações ao servidor. Os *stubs* e *skeletons* tratam, portanto, os aspectos relativos à distribuição e localização dos servidores, tornando a comunicação entre clientes e servidores completamente transparente. A comunicação entre cliente e servidor se dá através de um protocolo padronizado pelo OMG, chamado GIOP (*General Inter-ORB Protocol*), o qual define um conjunto de primitivas de comunicação. O protocolo IIOP (*Internet Inter-ORB Protocol*) é a implementação do GIOP para redes baseadas nos protocolos TCP/IP. A Figura 2.3 mostra como ocorre a requisição de um serviço junto a um servidor.

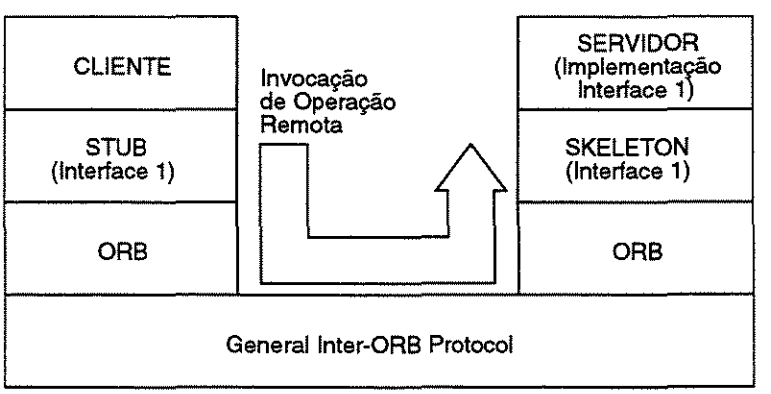


Figura 2.3: Requisição de um serviço através de um ORB.

O OMG desenvolveu ainda a arquitetura OMA (*Object Management Architecture*),

que utiliza o ORB e além disso especifica outros quatro componentes com o objetivo de facilitar o desenvolvimento de aplicações distribuídas (Figura 2.4):

CORBA *Services*. Definem serviços que estendem as funcionalidades básicas do ORB, como por exemplo Serviço de Nomes, de Persistência, de Concorrência e de Transações.

CORBA *Facilities*. Definem um conjunto de serviços que são freqüentemente utilizados pelas aplicações, mas não são considerados fundamentais. Tais serviços podem ser aplicados a qualquer domínio, como por exemplo Internacionalização e Agentes Móveis.

Interfaces de Domínios. Definem interfaces IDL para áreas específicas de aplicação, como Saúde, Finanças e Telecomunicações.

Objetos de Aplicação. São objetos criados por usuários da arquitetura OMA, não sendo portanto padronizados. Podem ser escritos em qualquer linguagem e plataforma e podem utilizar os serviços oferecidos pelos demais componentes.

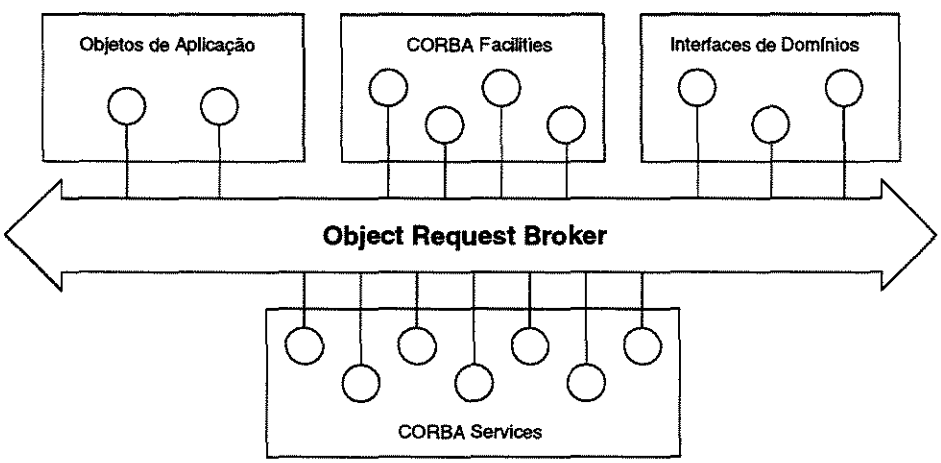


Figura 2.4: Componentes da Arquitetura OMA.

Toda a arquitetura OMA é definida em termos de interfaces e funcionalidades, sem contudo fornecer nenhuma implementação. Assim, são definidos apenas padrões que podem ser implementados de muitas formas diferentes. Uma implementação da arquitetura OMA deve fornecer os serviços definidos em CORBA, sendo os demais componentes opcionais.

2.2.1 OrbixWeb

A implementação do protótipo do WorkToDo utilizou o OrbixWeb 3.1c [28, 29, 30], a implementação de CORBA 2.0 da IONA [16]. O OrbixWeb 3.1c possui um compilador IDL com mapeamento para Java e a comunicação remota é efetuada através do padrão IIOP.

O OrbixWeb possui um componente chamado ***orbixd*** (*OrbixWeb daemon*), um processo que deve ser executado em cada *host* que abriga um servidor. Esse *daemon* é quem implementa vários dos componentes do ORB, além de controlar a ativação e desativação dos servidores. Quando ocorre a primeira requisição a um servidor, o *orbixd* é quem recebe a requisição, verificando se o servidor correspondente está ativo. Se estiver, repassa a requisição ao servidor; caso contrário, ativa o servidor e depois repassa a requisição. A partir daí o servidor recebe as próximas requisições diretamente. Assim sendo, não é necessário criar um servidor antes de requisitar suas operações; a criação é efetuada automaticamente pelo *orbixd* sempre que necessário.

O *orbixd* é capaz de substituir o serviço de nomes padrão de CORBA, pois através dele é possível localizar servidores e recuperar suas referências. Entretanto, o *orbixd* possui uma diferença: ele deve ser executado em cada *host* que abriga um servidor, pois o *orbixd* localiza e ativa servidores somente no *host* onde está sendo executado. Isso é feito por meio da operação `bind()` definida pelo Orbix, que recebe como parâmetro o nome do servidor a ser localizado e o *host* no qual se deseja localizá-lo.

O OrbixWeb oferece também a possibilidade de utilizar o serviço de nomes de CORBA, à escolha do programador. Assim, no OrbixWeb é possível localizar servidores através de requisições ao serviço de nomes CORBA ou através do *orbixd*.

O OrbixWeb estende as funcionalidades básicas do ORB, oferecendo os seguintes serviços adicionais:

Persistência dos servidores. É um serviço oferecido pelo *orbixd* que permite que o estado de um objeto servidor seja salvo e recuperado automaticamente quando este for desativado e ativado, respectivamente. Para tanto, basta definir um ***loader*** (criando uma classe que estende a classe `Loader` definida no OrbixWeb) e associá-lo ao servidor que se deseja tornar persistente. Assim, cada vez que o *orbixd* ativa o servidor, seu respectivo *loader* é acionado e carrega o estado previamente salvo. Da mesma forma, quando o servidor é desativado seu *loader* é acionado, salvando seu estado.

Threads* múltiplas nos servidores.** É outro serviço disponibilizado pelo *orbixd*, que permite que um servidor receba diversas requisições de serviços simultaneamente. Para isso, é necessário definir um ***filtro (criando uma classe que estende a classe `Filter` definida no OrbixWeb) e associá-lo ao servidor. A partir daí, cada vez que o

servidor receber uma requisição, seu respectivo filtro criará uma *thread* para tratar a requisição, deixando o servidor livre para receber outras requisições.

Filtros para operações. Como visto no item anterior, é possível criar um filtro para implementar servidores com múltiplas *threads*. Mas podem também ser criados outros tipos de filtros, que são executados no momento da invocação de operações remotas. Este recurso possibilita a execução de operações avançadas, como controle de acesso, coleta de informações a respeito de invocações remotas e depuração.

Localização de objetos por nome. O *orbixd* implementa ainda um serviço que permite localizar objetos específicos por um nome (*marker*). O *orbixd* mantém uma lista com os nomes dos objetos servidores localizados em seu *host*, juntamente com suas respectivas referências. Assim, quando o *orbixd* recebe uma requisição para um determinado objeto, ele procura o objeto na lista e retorna a sua referência. Isto funciona, portanto, como uma espécie de serviço de nomes local. O *marker* do objeto é passado como parâmetro para o `bind()`. Por exemplo, com a chamada `bind("Objeto_1:Servidor_1")`, o *orbixd* retornará a referência ao objeto do tipo `Servidor_1` cujo *marker* é `Objeto_1`. Podem existir diversos objetos do tipo `Servidor_1` ativos nesse *host*, mas o *orbixd* irá localizar especificamente aquele objeto. Caso não seja especificado um *marker* no `bind()`, o *orbixd* retornará uma referência a qualquer desses objetos.

Tempo de ativação dos servidores. É possível estabelecer, no momento da criação de um servidor, o tempo que esse servidor permanecerá ativo se não receber requisições. Se esse tempo é atingido (e o servidor não recebeu nenhuma requisição), o *orbixd* automaticamente desativa o servidor. É permitido estabelecer um tempo de ativação igual a infinito, o que significa que o servidor permanecerá ativo até que seu processo Java correspondente seja explicitamente finalizado (com um comando `kill` do Linux, por exemplo), mesmo que jamais receba uma requisição.

Modos de ativação dos servidores. É possível especificar modos de ativação diferentes para cada servidor. Existem dois modos: *shared* (compartilhado), no qual todas as requisições para determinado servidor em um *host* são tratadas pelo mesmo objeto; e *unshared* (não-compartilhado), no qual é criado um novo objeto, com sua própria máquina virtual Java, para cada requisição. Um servidor em modo *unshared* funciona como um com múltiplas *threads*, com a desvantagem de que a cada requisição é criado um novo processo ao invés de apenas uma *thread*.

Smart Proxies. Este recurso possibilita redefinir manualmente os *stubs*, adicionando e personalizando determinadas funcionalidades. Isso permite otimizar as operações do lado cliente e implementar políticas de balanceamento de carga entre servidores.

2.3 Computação sem Fio

Sistemas de computação sem fio[2] são sistemas que permitem a computação, comunicação e colaboração a qualquer hora e em qualquer lugar [23]. A computação sem fio se refere a sistemas onde computadores estão conectados ao seu ambiente de trabalho por meio de redes sem fio. É o caso, por exemplo, de um *notebook* conectado a um telefone celular.

Com a computação sem fio é possível também executar operações que utilizam dados de um servidor mesmo não estando conectado a esse servidor, através de técnicas como cache local de dados. Esses dados podem então ser modificados localmente, estando-se desconectado do servidor. Quando a conexão é reestabelecida, os dados na cache são sincronizados com as cópias no servidor [15, 23].

As principais características de um ambiente de computação sem fio são: desconexões freqüentes, grandes períodos sem existência de conexão, baixa largura de banda da rede e variações na velocidade de transmissão de dados. Assim, questões como minimização de transferência de dados e adaptação às variações do ambiente tornam-se muito importantes, exigindo mecanismos que proporcionem uma comunicação eficiente.

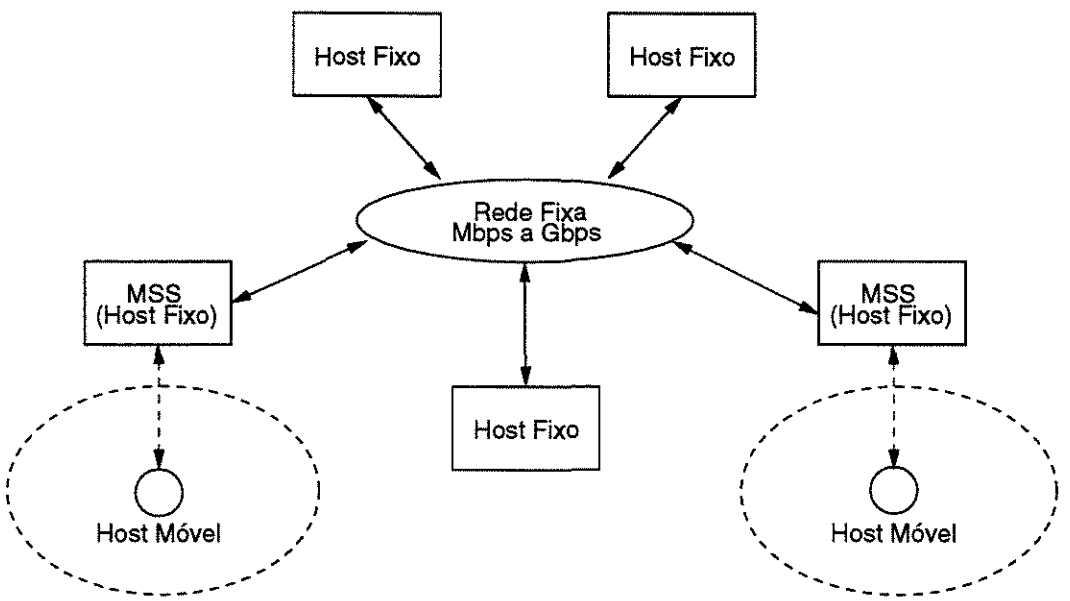


Figura 2.5: Arquitetura de um sistema de computação sem fio.

A arquitetura de um sistema de computação sem fio é mostrada na Figura 2.5 [15]. O sistema é composto por *hosts* fixos, conectados por uma rede fixa de alta velocidade (Ethernet, por exemplo), e por *hosts* móveis, conectados através de redes sem fio. Alguns

[2]Do inglês *wireless computing systems.*

dos *hosts* fixos são *Mobile Support Stations* (MSS), os quais fornecem uma interface para a comunicação dos *hosts* móveis dentro de uma área geográfica chamada célula. Os *hosts* móveis podem se conectar e desconectar do sistema a qualquer momento, e eventualmente até mesmo se conectarem à rede fixa.

Um *host* móvel pode operar em dois modos: conectado (também chamado semi-conectado), quando ligado à rede através de uma rede sem fio; e desconectado, quando desligado da rede, operando portanto de forma isolada. Mesmo no modo desconectado, o *host* móvel pode utilizar dados remotos, desde que esses dados tenham sido previamente copiados (cache local).

A mobilidade de usuários está intimamente ligada com a computação sem fio, pois um *host* móvel pode se conectar ao sistema a partir de diferentes MSS, caracterizando assim um ambiente móvel.

2.4 Requisitos de SGWFs em Ambientes de Comunicação sem Fio

Inicialmente, é importante notar que sistemas de *workflow* e ambientes de comunicação sem fio possuem objetivos um tanto contraditórios [6]. Um SGWF é uma ferramenta para cooperação e trabalho colaborativo, exigindo constante monitoramento de seus diversos componentes, enquanto que a operação em ambientes sem fio visa permitir que os usuários trabalhem de forma autônoma e isolada.

Utilizar um sistema de *workflow* em um ambiente móvel significa permitir que uma entidade (no caso, um usuário humano) execute as tarefas que lhe cabem a partir de algum dispositivo móvel, conectado ao SGWF por meio de uma rede sem fio. Mais do que isso, a entidade que utiliza o dispositivo móvel deve ser capaz de executar tarefas mesmo sem estar conectada ao SGWF, pois uma conexão nem sempre estará disponível.

Normalmente os SGWFs são utilizados em uma rede local, cujas características incluem alta velocidade, confiabilidade e estabilidade. Isso implica o fato de que qualquer entidade é capaz de executar todas as tarefas para as quais foi designada a qualquer momento, visto que os recursos necessários estão sempre disponíveis.

Ao considerarmos uma rede sem fio, entretanto, tais afirmações não são mais verdadeiras. As conexões são instáveis e possuem baixa largura de banda, os dispositivos móveis nem sempre pode ser contactados e as desconexões são freqüentes. Por isso, ao permitir a utilização de um sistema de *workflow* em ambientes móveis, diversas requisitos extras devem ser considerados:

Notificações de execução de tarefas. Podem existir diversas entidades responsáveis pela execução de uma tarefa, portanto quando uma entidade decide executar a tare-

fa, as demais entidades devem ser notificadas para que não a executem também. Em um ambiente móvel, porém, é possível que uma entidade móvel X esteja desconectada e decida executar uma tarefa. As demais entidades não receberiam nenhuma notificação da execução por parte de X e poderiam executar a mesma tarefa. Por isso, é necessário algum mecanismo que obrigue uma entidade móvel a notificar previamente o SGWF (enquanto conectada) a respeito de sua intenção em executar uma tarefa desconectada do sistema.

Aplicações. Para a execução de algumas tarefas podem ser necessárias uma ou mais aplicações. Assim sendo, para o usuário móvel executar uma tarefa é necessário que as aplicações necessárias estejam instaladas no dispositivo móvel. O SGWF deve então se preocupar em verificar se a aplicação está instalada no dispositivo móvel e copiá-la se necessário, antes da execução da tarefa.

Dados. A execução de uma tarefa utiliza dados, os quais também devem estar disponíveis no dispositivo móvel. O SGWF deve copiar esses dados para o dispositivo móvel antes da execução da tarefa. Da mesma forma, após o usuário ter concluído a execução da tarefa, os dados gerados devem ser copiados para o SGWF.

Gerenciamento de prazos de tarefas. As tarefas podem apresentar prazos máximos para sua execução, depois dos quais algum usuário responsável decide o que deve ser feito. Entretanto, como o usuário móvel se conecta apenas esporadicamente ao SGWF, é necessário que exista um controle local (no próprio dispositivo móvel) sobre os prazos das tarefas.

Um SGWF que se destine a ambientes sem fio, portanto, deve tratar corretamente desses requisitos para oferecer uma operação confiável.

Capítulo 3

Sistema WorkToDo

O ***WorkToDo*** [33, 32] é um Sistema de Gerenciamento de Workflows que oferece suporte para ambientes de comunicação sem fio, permitindo que um usuário execute tarefas sem estar conectado ao SGWF ou conectado através de uma conexão instável e de baixa largura de banda.

O modelo de processo é descrito na Seção 3.1, a Seção 3.2 mostra a linguagem utilizada para definir processos, a Seção 3.3 apresenta a arquitetura do WorkToDo e a Seção 3.4 mostra aspectos relativos ao funcionamento do sistema.

3.1 Modelo de Processo

Um ***modelo de processo*** visa modelar um processo do mundo real em termos computacionais, extraindo os aspectos relevantes do processo para proporcionar uma representação adequada.

A modelagem de um processo consiste em definir os seguintes aspectos: o que deve ser executado, em qual momento, por quem e com quais dados. Esses aspectos são chamados, respectivamente, de funcional, comportamental, organizacional e informacional [17]:

Aspecto funcional. Especifica quais as ações que devem ser executadas pelo processo. Para tanto, o processo é decomposto em diversas atividades, as quais por sua vez podem também ser decompostas em outras atividades, até que existam apenas ações elementares (tarefas).

Aspecto comportamental. Define quando cada atividade é executada, através de uma lista de interdependências entre atividades.

Aspecto organizacional. Define quem é o responsável pela execução de cada atividade. Este responsável, denominado entidade processadora, pode ser uma aplicação ou um

humano.

Aspecto informacional. Especifica os dados utilizados pelas atividades, bem como o fluxo desses dados entre as atividades. Uma atividade pode utilizar dados de entrada e gerar dados de saída, que por sua vez podem ser utilizados como entrada por outras atividades, e assim sucessivamente.

A Figura 3.1 mostra a representação do modelo de processo utilizado pelo WorkToDo na notação UML (*Unified Modeling Language*). Um processo é formado por um conjunto de atividades e dependências entre elas, e cada atividade possui um conjunto de dados de entrada e saída e uma entidade processadora responsável por sua execução.

O WorkToDo utiliza o conceito de ***Tipos de Processo***. Isso significa que quando um processo é modelado, o que se define é na verdade um tipo de processo, que pode então ser instanciado várias vezes, criando execuções particulares desse tipo.

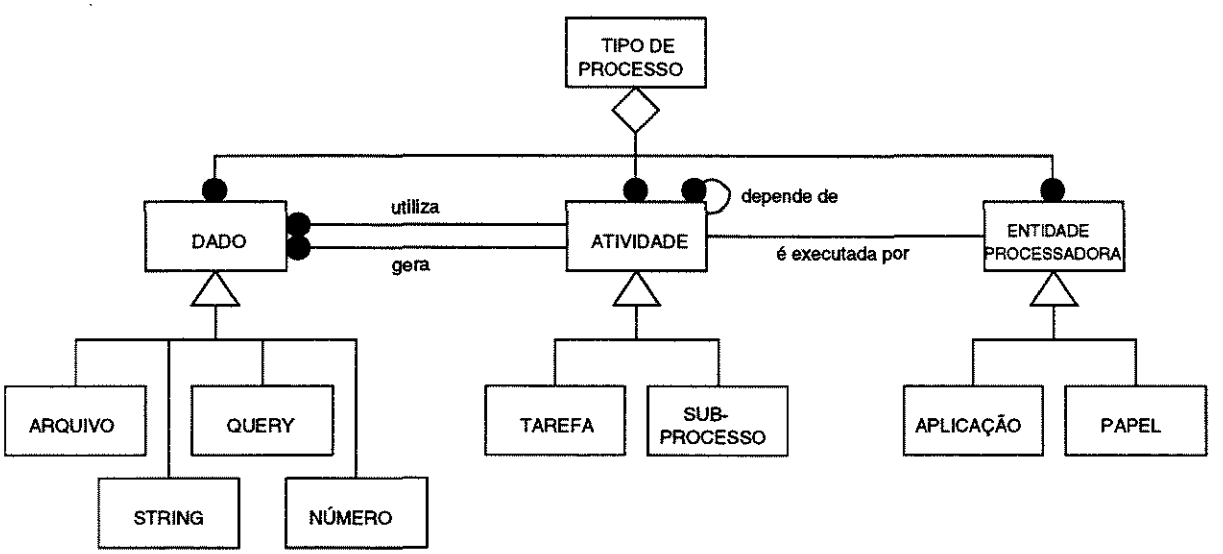


Figura 3.1: Modelo de processo do WorkToDo.

A Tabela 3.1 mostra a relação entre a nomenclatura adotada neste trabalho e aquela definida pela *Workflow Management Coalition* no Modelo de Referência para *Workflow* [39].

3.1.1 Atividades

Um processo é formado por um conjunto de atividades, de tal forma que quando tais atividades são desempenhadas a execução do processo está concluída, definindo assim o aspecto funcional de um processo.

WorkToDo	WfMC
Tipo de Processo	Definição de Processo
Instância de Processo	Instância de Processo
Atividade	Atividade
Tarefa	Atividade Atômica
Entidade Processadora	Participante de Processo

Tabela 3.1: Nomenclatura adotada no WorkToDo.

No momento da modelagem o processo é decomposto em diversas atividades, que podem ser tarefas ou sub-processos. As tarefas são as ações elementares de um processo, enquanto os sub-processos são outros processos. A possibilidade de uma atividade ser um outro processo permite um maior grau de reusabilidade, já que um processo pode ser usado como parte de outro sem ser necessária nenhuma redefinição.

Estados das Atividades

Para que o SGWF possa gerenciar corretamente a execução das atividades, é necessário que seja possível determinar qual sua situação, ou seja, se elas foram ou não executadas e qual seu resultado. Por isso, são definidos cinco estados possíveis para uma atividade:

`NOT_READY`. A atividade ainda não pode ser executada, porque as condições necessárias para sua execução não foram satisfeitas.

`READY`. As condições necessárias para a execução da atividade foram satisfeitas e portanto ela está pronta para ser executada.

`RUNNING`. A atividade está em execução.

`SUCCEEDED`. A atividade foi executada com sucesso.

`FAILED`. Ocorreu alguma falha durante a execução da atividade.

A transição entre esses estados é mostrada na Figura 3.2. `NOT_READY` é o estado inicial e `SUCCEEDED` e `FAILED` são os estados finais. Em um dado momento, uma atividade pode se encontrar em um e somente um estado.

3.1.2 Tarefas

As tarefas são aquelas atividades que não podem mais ser decompostas, representando ações elementares como, por exemplo, digitar um texto ou enviar uma carta. Uma tarefa é definida através de um ***Modelo de Tarefa***, que especifica as características gerais da

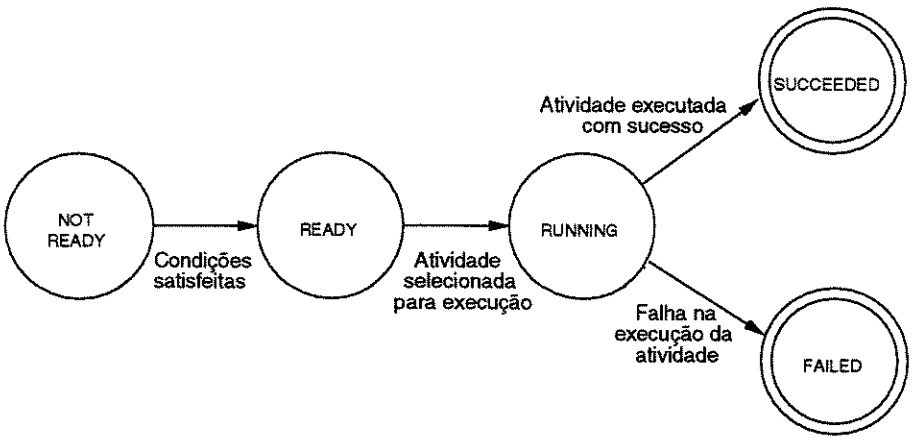


Figura 3.2: Diagrama de transição dos estados das atividades.

tarefa, como tipo e papel responsável por sua execução. Esse modelo é então instanciado pelos processos, criando uma tarefa que é utilizada como se tivesse sido definida pelo próprio processo.

Um mesmo modelo de tarefa pode ser utilizado em vários processos diferentes, bem como várias vezes em um mesmo processo, possibilitando sua reutilização. Os dados utilizados e gerados pelas tarefas, por sua vez, são especificados no tipo de processo, de tal forma que cada instanciação de um modelo de tarefa pode utilizar dados diferentes.

Tipos de Tarefas

As tarefas podem diferir completamente umas das outras, exigindo tratamentos diferenciados pelo sistema. Por isso elas são classificadas em três tipos:

Automáticas. Tarefas que são executadas por algum software invocado automaticamente pelo SGWF. Exemplos: acesso a um banco de dados, cópia de um arquivo.

Semi-automáticas. São executadas por um humano auxiliado por algum sistema de software. As ações são ditadas pelo usuário, mas ele utiliza um software como ferramenta. Exemplos: redigir um documento em um editor de textos, escanear uma foto, enviar um e-mail.

Manuais. São executadas exclusivamente por um humano, sem a participação de nenhum software. Exemplos: instalar um dispositivo, preencher um formulário em papel e enviar uma carta pelo correio.

As tarefas automáticas são executadas diretamente pelo SGWF, sem a necessidade de nenhuma intervenção por parte dos usuários. Já as semi-automáticas e manuais requerem sempre a participação de um usuário, que define as ações a serem tomadas.

3.1.3 Regras de Dependência

Como descrito anteriormente, um dos aspectos que devem ser modelados em um processo é o comportamental. No WorkToDo, isto é feito através de regras de dependência.

Uma ***Regra de Dependência*** indica um conjunto de restrições para a execução de uma atividade. Em outras palavras, contém as condições que devem ser satisfeitas para que uma determinada atividade seja executada. Uma regra de dependência é composta por um ou mais termos, onde cada termo é formado por um nome de atividade A e um estado E, representando a transição de A para E. Os termos podem se relacionar através dos operadores booleanos `and` e `or`. Um exemplo de regra de dependência é mostrado abaixo:

$$\texttt{and } (A_1 \rightarrow \texttt{SUCCEEDED}, A_2 \rightarrow \texttt{SUCCEEDED})$$

Nesse exemplo, a atividade associada à regra somente será executada se as atividades A_1 e A_2 alcançarem o estado `SUCCEEDED`. A regra de dependência possui dois termos: $A_1 \rightarrow$ `SUCCEEDED` e $A_2 \rightarrow$ `SUCCEEDED`, relacionados através da operação `and`.

É possível construir regras de dependência mais complexas, como a mostrada abaixo, onde a atividade associada à regra será executada se A_1 e A_2 alcançarem o estado `SUCCEEDED`, ou se A_3 alcançar o estado `FAILED`:

$$\texttt{or (and } (A_1 \rightarrow \texttt{SUCCEEDED}, A_2 \rightarrow \texttt{SUCCEEDED}), A_3 \rightarrow \texttt{FAILED})$$

A cada atividade é associada uma regra de dependência e a atividade somente é executada quando sua regra é avaliada para verdadeiro (`true`). A regra de dependência pode ser vazia, significando que a atividade pode ser executada tão logo a instância do processo ao qual a atividade pertence seja criada. Os estados válidos para os termos de uma regra são aqueles possíveis para as atividades, conforme mostrado na Seção 3.1.1.

Com a regra de dependência, é possível construir uma ***Árvore de Dependência***. Nessa árvore, as folhas são nomes de atividades e os nós intermediários são operadores `and` e `or`. Além do nome da atividade ou da operação, os nós possuem um campo `valor`, contendo um booleano que inicialmente é igual a `false`. As folhas possuem ainda um campo adicional `nome-de-estado`, contendo um nome de estado E.

Quando ocorre a transição de uma atividade A para E, o campo `valor` da folha correspondente à atividade é atualizado para `true`, indicando que a dependência daquele nó (e portanto da atividade A) foi satisfeita. O pai da folha é então notificado, avaliando então sua expressão booleana (o pai é sempre um operador `and` ou `or`) considerando todos os seus filhos. Se sua expressão avaliar para `true`, altera seu campo `valor` e notifica seu nó pai, que repete o processo. Quando a raiz é notificada e altera seu campo `valor` para

`true`, a atividade está pronta para executar. É importante observar que inicialmente o campo `valor` de todos os nós contém o valor `false`.

A Figura 3.3 mostra a árvore de dependência correspondente à primeira regra mostrada anteriormente. Quando a atividade A_1 atinge o estado `SUCCEEDED`, sua folha correspondente passa a ter o valor `true`, notificando então seu pai (o nó `and`). Esse nó verifica sua expressão (um `and` entre os valores dos dois filhos), mas esta ainda não é verdadeira pois o segundo filho continua com o valor `false`.

No caso da regra de dependência ser vazia, a árvore de dependência é sempre avaliada para `true`.

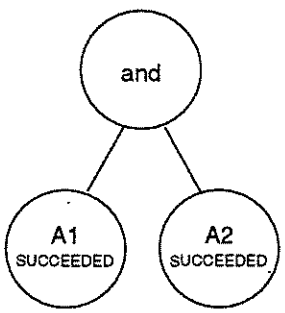


Figura 3.3: Exemplo de Árvore de Dependência.

3.1.4 Entidades Processadoras

Para toda tarefa existe uma ***Entidade Processadora*** responsável por sua execução. Quando dizemos responsável, significa que em algum momento (não necessariamente de imediato) a entidade irá executar a tarefa.

As entidades processadoras modelam o aspecto organizacional de um processo e são definidas nos modelos de tarefas. As entidades processadoras podem estar em diversos locais diferentes e podem existir zero ou mais tarefas atribuídas a uma entidade processadora em um dado momento.

Existem dois tipos de entidades processadoras: ***Aplicações*** e ***Usuários***.

Aplicações

As aplicações são as entidades processadoras das tarefas automáticas. Isso significa que para executar a tarefa, basta que a aplicação correspondente seja executada (possivelmente com parâmetros e/ou dados de entrada/saída). A cada tarefa automática é atribuída uma única aplicação.

Assim como as tarefas, as aplicações são definidas independentemente. Dessa forma um modelo de tarefa que deseja utilizar determinada aplicação apenas a referencia, sem ser necessário redefini-la. A definição de uma aplicação contém o nome de seu arquivo executável correspondente e seu tamanho total, entre outras características.

Usuários

Os usuários são as entidades processadoras de tarefas semi-automáticas ou manuais. A atribuição de usuários à execução de tarefas se dá através de um sistema de papéis. Um ***Papel*** designa um grupo de usuários que possuem determinadas características, requisitos e/ou habilidades. Um papel pode conter um ou mais usuários e um usuário pode pertencer a um ou mais papéis. Na especificação de uma tarefa é indicado um papel R responsável pela execução da mesma. Assim, qualquer usuário pertencente a R pode executar a tarefa, já que todos esses usuários possuem as características necessárias para tanto.

Isso aumenta a eficiência do sistema, pois como diversos usuários podem executar uma tarefa, a probabilidade de ela ser executada em breve é maior do que se um usuário fosse pré-escolhido.

No caso das tarefas semi-automáticas, os usuários utilizam uma aplicação para executar a tarefa. Essa aplicação é invocada durante a execução da tarefa, entretanto não é considerada uma entidade processadora.

3.1.5 Dados

As tarefas podem utilizar dados de entrada e gerar dados de saída, dados estes que correspondem ao aspecto informacional dos processos. No WorkToDo, os dados podem ser de quatro tipos:

Arquivos. Arquivos armazenados em disco (exceto no caso de dispositivos que não possuem disco, como PDAs; nesse caso o arquivo é armazenado em memória). Um arquivo possui associado a ele um nome e um tamanho.

Queries. Expressões em SQL (*Simple Query Language*) que definem consultas a bancos de dados e que possivelmente retornam um conjunto de dados (registros). A uma *query* é associado o banco de dados ao qual ela se refere, juntamente com uma expressão em SQL.

Strings. Contém valores alfanuméricos.

Números. Podem conter valores inteiros ou reais.

Cada processo em execução utiliza uma área exclusiva, denominada ***Contexto do Processo***, na qual armazena seus dados. Assim, quando uma tarefa necessita recuperar ou armazenar dados, deve fazê-lo nessa área. O contexto de uma instância de processo P somente é acessível às tarefas pertencentes a P, isolando assim os dados de cada instância de processo e evitando que a execução de um processo interfira em outro.

3.2 Linguagem de Definição de Processo

Para a especificação de um processo, o WorkToDo possui uma ***Linguagem de Definição de Processo*** (*Process Definition Language* — PDL), na qual é possível especificar tipos de processos, modelos de tarefas e aplicações. A PDL é descrita em detalhes nas seções seguintes e a gramática da linguagem está no Apêndice A.

3.2.1 Tipos de Processo

A especificação de um tipo de processo é composta basicamente por dois blocos: um contendo a declaração dos dados a serem utilizados pelas atividades e outro contendo a declaração das atividades a serem desempenhadas, descrevendo quais as condições necessárias para sua execução, as entidades que devem executá-las e quais os dados por elas utilizados.

```
WORKFLOW <workflow-id> {
    FILE <arquivo-id> {
        NAME <nome-do-arquivo>;
    }
    ⋮
    QUERY <query-id> {
        DATABASE <nome-do-BD>;
        EXPRESSION <expressão-SQL>;
    }
    ⋮
    STRING <string-id> {
        VALUE <valor>;
    }
    ⋮
    NUMBER <número-id> {
        VALUE <valor>;
    }
```

```
    ⋮

    TASK <tarefa-id>:<modelo-de-tarefa> {
        DEPENDS <regra-de-dependência>;
        IN_CONTEXT <lista-de-dados>;
        OUT_CONTEXT <lista-de-dados>;
        ROLE <nome-do-papel>;
        DESCRIPTION <descrição-textual>;
    }
    ⋮
    SUB-WORKFLOW <sub-workflow-id>:<tipo-de-workflow> {
        DEPENDS <regra-de-dependência>;
    }
    ⋮
}
```

O `workflow-id` identifica o tipo de processo. Cada instância do processo conterá esse identificador como parte de seu nome. Em seguida são especificadas as cláusulas `FILE`, `QUERY`, `STRING` e `NUMBER`, para definição dos dados de entrada e saída das tarefas. Um tipo de processo pode conter zero ou mais dessas cláusulas.

A cláusula `FILE` define um arquivo, identificado pelo `arquivo-id`. Futuras referências ao arquivo, quando este for utilizado por alguma tarefa, deverão ser feitas através desse identificador. O arquivo físico é indicado pelo `nome-do-arquivo` (que deve conter o caminho completo).

A cláusula `QUERY` define uma operação em um banco de dados, identificada por `query-id`. Em `DATABASE` é especificado o nome do banco de dados; a expressão SQL correspondente à consulta está na cláusula `EXPRESSION`.

As cláusulas `STRING` e `NUMBER` definem variáveis que armazenam, respectivamente, valores alfanuméricos e numéricos. Os valores de um `NUMBER` podem ser inteiros ou reais.

A cláusula `TASK` inclui uma tarefa ao tipo de processo, identificada pelo `tarefa-id`. As referências à tarefa nas regras de dependência utilizam este identificador. O `modelo-de-tarefa` referencia um modelo de tarefa existente, no qual estão detalhadas as características daquele modelo. O operador : indica, portanto, a instanciação de um modelo de tarefa para utilização dentro do processo.

As dependências entre tarefas são definidas na cláusula `DEPENDS`. A `regra-de-dependência` é especificada conforme descrito na Seção 3.1.3.

A cláusula `IN_CONTEXT` contém os dados utilizados como entrada pela tarefa. Da mesma forma, `OUT_CONTEXT` contém os dados que a tarefa gera como saída. Em ambos

os casos, `lista-de-dados` é uma lista de `arquivo-ids`, `query-ids`, `string-ids` e/ou `number-ids`, separados por vírgulas. As cláusulas `IN_CONTEXT` e `OUT_CONTEXT` podem não existir, significando que a tarefa não tem dados de entrada ou não gera dados de saída. Ao identificar os dados de entrada e saída de cada tarefa, essas cláusulas também especificam o fluxo de dados entre as tarefas.

`ROLE` indica qual o papel responsável pela execução da tarefa, identificado por `nome-do-papel`. Somente usuários pertencentes a esse papel podem executar a tarefa. Se a tarefa for automática, o papel não é considerado.

Na cláusula `DESCRIPTION` pode ser inserida uma descrição textual da tarefa, informando sua finalidade, forma de execução e quaisquer outras observações consideradas importantes. A `descrição-textual` pode ser vazia.

Por fim, a cláusula `SUB-WORKFLOW` permite que um *workflow* inteiro seja adicionado como uma atividade do *workflow* que está sendo definido, sendo executado em forma de subprocesso. O *workflow*, identificado por `sub-workflow-id`, é do tipo `tipo-de-processo`, e as condições para sua execução estão especificadas na regra de dependência da cláusula `DEPENDS`.

3.2.2 Modelos de Tarefa

Um modelo de tarefa especifica em detalhes as características de determinado tipo de tarefa. Dessa forma, quando um *workflow* instancia esse modelo, diversas características gerais da tarefa já estão definidas, evitando repetições.

```
TASK <modelo-de-tarefa-id> {
     TYPE <nome-do-tipo>;
     APPLICATION <aplicação-id>;
     PRIORITY <número-de-prioridade>;
     DEADLINE <prazo> <tipo-de-prazo>;
     DISCONNECTED_OPERATION <permissão>;
     RETRIES <número-de-tentativas>;
}
```

O `modelo-de-tarefa-id` é o nome do modelo de tarefa. As definições de processo que instanciam esse modelo utilizam esse nome para referenciá-lo.

A cláusula `TYPE` define o tipo da tarefa, conforme a Seção 3.1.2. O tipo da tarefa influi na forma como a execução da tarefa será gerenciada.

A cláusula `APPLICATION` define qual a aplicação – identificada por `aplicação-id` – que deve ser invocada durante a execução da tarefa. A cada tarefa é associada uma e somente uma aplicação. Se a tarefa for manual, não deve ser especificada nenhuma aplicação.

O número-de-prioridade da cláusula PRIORITY indica qual a prioridade da tarefa. O SGWF executa tarefas automáticas com maior prioridade antes de tarefas automáticas com menor prioridade.

Contido na cláusula DEADLINE está o prazo máximo para que a tarefa seja executada. tipo-de-prazo pode ser dias ou horas.

A cláusula DISCONNECTED_OPERATION indica se a tarefa pode ou não ser executada em modo desconectado. Se permissão for true a operação desconectada é permitida, se for false não. Esta cláusula é válida somente para tarefas manuais ou semi-automáticas.

Na cláusula RETRIES é indicado o número de vezes que uma tarefa deve ser re-executada em caso de falha. Se o número-de-tentativas for ultrapassado, a execução da tarefa é dada como falha. Essa cláusula somente é válida para tarefas automáticas.

3.2.3 Aplicações

```
APPLICATION <aplicação-id> {
    FILENAME <nome-do-arquivo>;
    SIZE <tamanho-da-aplicação>;
    OS <sistema-operacional>;
    CPU <processador>;
    INSTALLER <nome-do-instalador>;
    HOSTS <lista-de-hosts>;
}
```

O aplicação-id é o nome da aplicação. Os modelos de tarefas que fazem uso da aplicação a referenciam através desse identificador.

A cláusula FILENAME contém o nome do arquivo executável da aplicação, com o seu caminho completo, indicado por nome-do-arquivo.

Na cláusula SIZE é indicado o tamanho da aplicação, definida como a soma do tamanho de todos os arquivos por ela utilizados (executável e bibliotecas de ligação dinâmica, por exemplo).

A cláusula OS indica o sistema operacional no qual a aplicação executa. CPU indica o tipo de processador onde a aplicação pode executar.

A cláusula INSTALLER contém o arquivo de instalação da aplicação (se ele existir), com seu caminho completo, indicado por nome-do-instalador. Esse instalador provê todos os arquivos necessários para a execução da aplicação. Se o instalador for omitido, assume-se que o único arquivo necessário para a aplicação é seu executável.

Na cláusula HOSTS é indicada uma lista de *hosts* nos quais a aplicação está disponível para execução. É necessário informar pelo menos um *host*. A especificação de diversos *hosts* é uma medida preventiva contra possíveis falhas de um *host*.

3.3 Arquitetura do Sistema

O WorkToDo possui uma arquitetura distribuída, cujos componentes são mostrados na Figura 3.4 e descritos nas próximas Seções.

Cada componente tem uma responsabilidade básica: o Gerenciador de Usuários e Papéis (*User and Role Manager* — URM) gerencia os usuários e papéis existentes, bem como a lista de usuários pertencentes a cada papel; o Gerenciador de Definições (*Definition Manager* — DM) controla o acesso às definições de processos, tarefas e aplicações; o Gerenciador de Instâncias (*Instance Manager* — IM) registra informações a respeito das instâncias de processos em execução e também das que já terminaram; o Gerenciador de Processo (*Process Manager* — PM) coordena a execução de uma determinada instância de processo; o Gerenciador de Tarefa (*Task Manager* — TM) controla a execução de uma tarefa automática; e o Gerenciador de Lista de Trabalho (*Worklist Manager* — WM) controla uma lista com as tarefas que podem ser executadas por um determinado usuário.

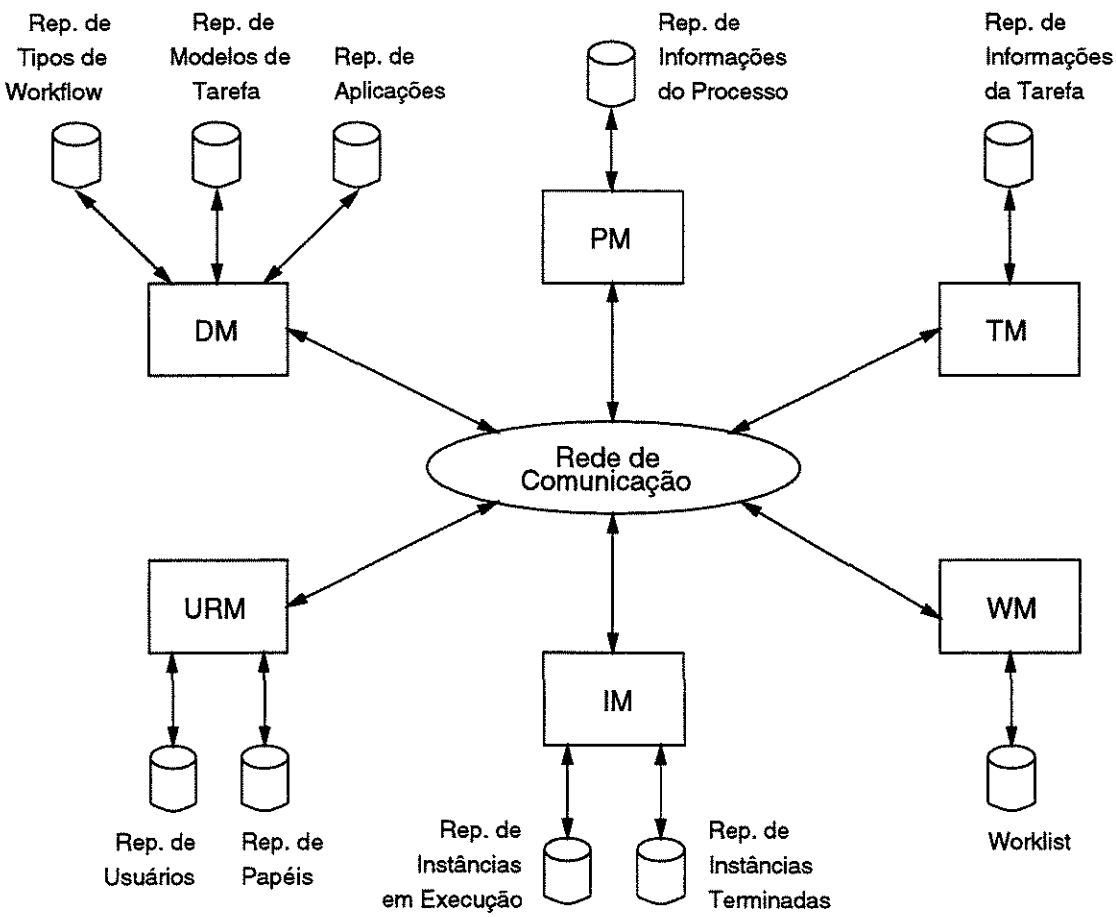


Figura 3.4: Arquitetura WorkToDo.

O relacionamento entre os componentes da arquitetura é mostrado na Figura 3.5. Como cada um desses componentes pode ser executado em um *host* diferente, a comunicação entre eles é efetuada utilizando alguma plataforma que forneça um método de comunicação remota, como RPC (*Remote Procedure Call*), Sockets, CORBA (*Common Object Request Broker Architecture*) ou RMI (*Remote Method Invocation*).

A escolha de uma arquitetura distribuída se deve ao fato de que dessa forma não há uma carga excessiva sobre um determinado *host*, evitando sobrecarga de processamento. Além disso o sistema se torna mais tolerante a falhas, já que se um *host* falhar apenas os componentes que estavam sendo executados nesse *host* se tornam inacessíveis, enquanto o restante do sistema continua funcionando normalmente.

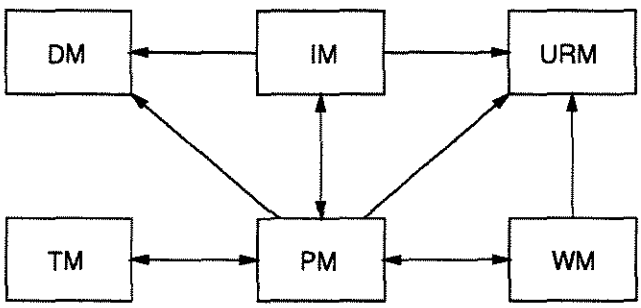


Figura 3.5: Relacionamento entre os componentes da arquitetura WorkToDo.

3.3.1 Gerenciador de Usuários e Papéis

O Gerenciador de Usuários e Papéis (*User and Role Manager — URM*) é o componente responsável por gerenciar usuários e papéis dentro do sistema. Verificações como os papéis aos quais um determinado usuário pertence e a lista de usuários de um determinado papel são todas efetuadas pelo URM, além da adição e remoção de usuários e papéis. Para tanto, este componente controla dois repositórios:

Usuários. Contém informações a respeito dos usuários do sistema: nome, senha, situação (conectado ou não ao SGWF), endereço IP de seu *host*, há quanto tempo está conectado e quando foi sua última conexão. Também guarda uma lista de mensagens para cada usuário, conforme será visto na Seção 3.4.4.

Papéis. Armazena os papéis existentes, juntamente com uma lista dos usuários pertencentes a cada papel.

Desta forma, sempre que um componente do sistema necessita acessar ou modificar as informações a respeito de um usuário ou papel, uma requisição é feita ao URM, que

então consulta o repositório correspondente. O URM fornece operações do tipo `add()`, `remove()` e `list()` para cada repositório. A listagem completa de operações do URM está no Apêndice B.

3.3.2 Gerenciador de Definições

A função do Gerenciador de Definições (*Definition Manager — DM*) é gerenciar as definições de processos, tarefas e aplicações. O DM permite recuperar as definições existentes, além de adicionar e remover definições. Para tanto, mantém três repositórios que armazenam os tipos de processos, os modelos de tarefas e as aplicações.

Desta forma, sempre que se necessita de informações a respeito de definições de processos, tarefas ou aplicações, é feita uma requisição ao DM, que então consulta o repositório correspondente. O DM fornece operações do tipo `add()`, `remove()` e `list()` para cada repositório. A listagem completa das operações do DM está no Apêndice B.

Quando um tipo de processo é adicionado ao repositório de tipos, o DM requisita o nome de um papel. Esse papel é chamado ***Papel Criador*** do tipo de processo. Instâncias desse tipo somente poderão ser criadas por usuários que pertencem ao papel criador. Cada tipo de processo possui associado a ele um papel criador.

3.3.3 Gerenciador de Instâncias

O Gerenciador de Instâncias (*Instance Manager — IM*) tem a função de manter um registro das instâncias de processo em execução, bem como daquelas cuja execução já foi concluída. O IM também é responsável por criar um Gerenciador de Processo para cada instância de processo criada.

O IM não controla a execução de instâncias; tal tarefa é responsabilidade do Gerenciador de Processo, descrito na Seção 3.3.4. O IM apenas mantém um registro das instâncias em execução e já executadas, através de dois repositórios:

Processos. Armazena informações a respeito das instâncias em execução, como nome, tipo e *host* a partir do qual a instância está sendo gerenciada.

Processos completados. Armazena informações a respeito das instâncias cujas execuções já foram concluídas. Além do nome da instância, guarda seu estado final, contendo as datas e horas de início e de término e o estado final de cada uma de suas tarefas.

Assim, sempre que um componente do sistema necessita recuperar ou atualizar dados de uma determinada instância de processo, ele consulta o IM para obter o *host* a partir

do qual a instância está sendo gerenciada. Da mesma forma, se algum componente deseja verificar quais são as instâncias de processo em execução, ou consultar o estado das instâncias completadas, ele faz uma requisição ao IM, que então consulta o repositório correspondente. O IM fornece operações do tipo `add()`, `remove()` e `list()` para cada repositório. A listagem completa das operações do IM está no Apêndice B.

O IM também é responsável por tratar eventuais falhas dos Gerenciadores de Processo (PMs). Em tais casos, o IM pode criar um novo PM em outro *host*, inicializando-o com o último estado consistente do PM que falhou. Por isso, o IM armazena no repositório de Processos o estado atual de cada instância em execução, estado esse atualizado periodicamente.

3.3.4 Gerenciador de Processo

O Gerenciador de Processo (*Process Manager — PM*) é o componente responsável por coordenar a execução de um processo. O PM verifica quais tarefas estão prontas para executar, inicia sua execução e coleta seus resultados, verificando se tais resultados permitem a execução de novas tarefas. Existe um PM para cada instância de processo em execução.

O PM mantém uma lista com as tarefas que compõem o processo, denominada ***Tasklist***. A *tasklist* armazena as informações necessárias para a execução das tarefas, como nome, tipo, entidade processadora, estado atual e dados utilizados como entrada e saída. Esse conjunto de informações recebe o nome de ***Definição de Tarefa*** (*Task Definition*). A *tasklist* constitui-se, portanto, de uma lista de *task definitions*. As estruturas da *tasklist* e da *task definition* são mostradas na Figura 3.6.

Além dos itens contidos na definição do tipo de tarefa (tipo, prioridade, prazo), a *task definition* de uma tarefa T possui também os seguintes campos:

Contexto de entrada. Uma lista com os dados que são utilizados por T. Corresponde aos dados declarados na cláusula `IN_CONTEXT` da definição do tipo de processo.

Contexto de saída. Uma lista com os dados que são gerados por T como resultado. Corresponde aos dados declarados na cláusula `OUT_CONTEXT` da definição do tipo de processo.

Árvore de Dependência. Contém a árvore de dependência de T, conforme descrito na Seção 3.1.3.

Dependentes. É uma lista com todas as tarefas que dependem de T. Esta lista é organizada da seguinte forma: para cada estado possível de T é mantida uma lista

contendo as tarefas que dependem da transição de T para aquele estado. Por exemplo, seu estado SUCCEEDED contém todas as tarefas que possuem em sua árvore de dependência a transição da tarefa T para o estado SUCCEEDED. A lista de dependentes é utilizada para notificar as tarefas interessadas nas mudanças de estado de T.

Estado. O estado atual da tarefa, conforme a Seção 3.1.2.

Usuário. O nome do usuário que selecionou a tarefa para execução (no caso de tarefas semi-automáticas ou manuais).

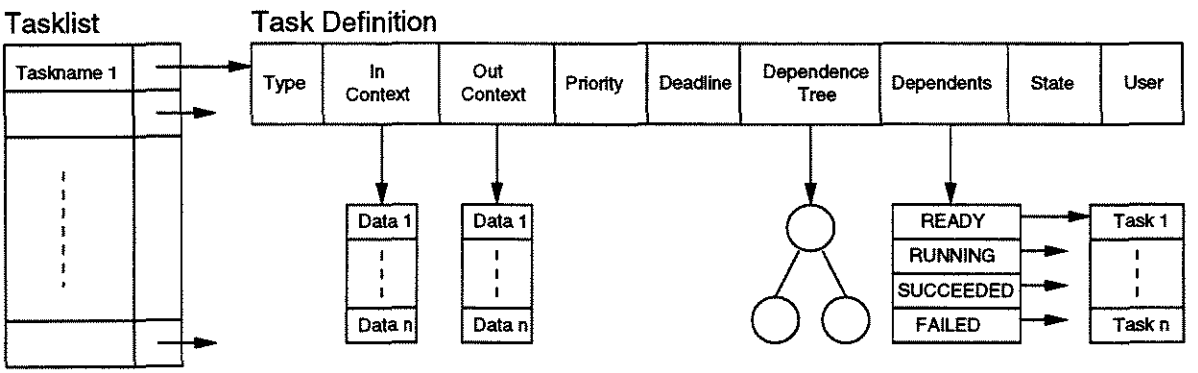


Figura 3.6: Representação da *Tasklist* e da *Task Definition*.

Se em algum momento não há nenhuma tarefa nos estados READY ou RUNNING, significa que o processo está concluído. É importante salientar que o processo pode terminar mesmo que algumas de suas tarefas não tenham sido executadas. Isso ocorre porque podem existir tarefas que dependem da transição de outras para um determinado estado que nunca é alcançado. Quando um processo é concluído, seu estado final é armazenado no repositório de Processos Completados existente no Gerenciador de Instâncias.

A *tasklist* é construída no momento da criação do PM, quando a definição do tipo de processo é interpretada. Para tanto o PM utiliza um interpretador da PDL, denominado ***Interpretador da Linguagem de Definição de Processo*** (*Process Definition Language Interpreter* — PDLI, o qual, a partir do tipo de processo e das tarefas e aplicações que o compõem, constrói a *tasklist* da instância. O PDLI contrói uma *task definition* para cada tarefa instanciada no processo e ao final reúne todas as *task definitions*, formando a *tasklist*.

O PM mantém ainda um repositório que armazena o estado da instância do processo, contendo informações como o nome do processo, seu horário de início e de término, além da *tasklist*. Dessa forma é possível consultar o estado do processo (tarefas completadas e tarefas em execução, por exemplo), assim como de cada tarefa individualmente.

É também responsabilidade do PM tratar eventuais falhas nos Gerenciadores de Tarefa. Nestes casos, o PM pode criar um novo TM, no mesmo ou em outro *host*, inicializando-o com o último estado conhecido do TM que falhou.

A listagem completa das operações do Gerenciador de Processos está no Apêndice B.

O PM utiliza dois sub-componentes locais, que auxiliam no gerenciamento do processo: o ***Escalonador*** e o ***Despachante***, descritos a seguir.

Escalonador

O Escalonador (*Scheduler*) tem por função avaliar as condições para execução das tarefas, de acordo com suas respectivas árvores de dependências. É o Escalonador, portanto, quem decide quais tarefas podem ser executadas em um determinado momento.

Sempre que uma tarefa T sofre uma transição para um estado E, o Escalonador reavalia a árvore de dependência de todas as tarefas que dependiam da transição de T para E (ou seja, que pertenciam à lista de Dependentes de T). Quando a árvore de dependência de uma tarefa é avaliada para `true`, o estado da tarefa é alterado para `READY`, indicando que ela está pronta para executar.

Sempre que o Escalonador marca uma tarefa como `READY`, os dados relativos à tarefa (se existirem) são copiados para a área do processo, para serem acessados durante a execução da tarefa.

No momento em que uma instância de processo é criada, o Escalonador marca todas as tarefas cujas árvores de dependência são vazias como prontas para executar.

Despachante

A função do Despachante (*Dispatcher*) é preparar as tarefas para execução. Ele não executa as tarefas; apenas fornece as condições necessárias para que elas possam ser executadas.

O Despachante verifica periodicamente a *tasklist* à procura de tarefas que estejam prontas para executar. Quando encontra uma tarefa pronta, executa uma das seguintes ações, conforme o tipo da tarefa:

Tarefa automática. Cria um Gerenciador de Tarefa para controlar a execução da tarefa, em um dos *hosts* indicados na definição da aplicação relacionada à tarefa.

Tarefa semi-automática ou manual. Notifica os usuários pertencentes ao papel da tarefa a respeito de sua disponibilidade. A escolha dos usuários a serem notificados se dá de acordo com uma ***Política de Notificação***:

1. Todos os usuários do papel;

2. Os n usuários do papel com menos tarefas atribuídas;
3. Os n usuários do papel com menos tarefas selecionadas;
4. Os n usuários do papel que se conectaram mais recentemente ao SGWF.

A política de notificação é escolhida no momento da criação da instância, podendo ser modificada durante sua execução. Através destas políticas é possível distribuir as tarefas de forma que usuários envolvidos com mais tarefas recebam menos notificações, balanceando a carga (políticas 2 e 3). Além disso, a política 4 permite também que sejam evitados usuários que permanecem muito tempo sem se conectar ao sistema.

3.3.5 Gerenciador de Tarefa

A função do Gerenciador de Tarefa (*Task Manager — TM*) é controlar a execução de uma tarefa automática.

O TM é criado por um PM e invoca a aplicação responsável pela execução da tarefa. Para que o TM saiba qual aplicação deve ser invocada, no momento da criação do TM o PM repassa a *task definition* correspondente à tarefa. Após a aplicação iniciar sua execução o TM a monitora e, quando ela é finalizada, notifica o PM que o criou, informando o tipo de término (com sucesso ou com falha).

Se a execução de uma tarefa falha, o TM verifica se o número de *retries* da tarefa foi alcançado. Se não foi, a aplicação é novamente invocada. Somente quando o número de *retries* é ultrapassado o TM notifica o PM correspondente a respeito da falha da tarefa.

A listagem completa das operações do Gerenciador de Tarefa está no Apêndice B.

3.3.6 Gerenciador de Lista de Trabalho

O Gerenciador de Lista de Trabalho (*Worklist Manager — WM*) tem a função de controlar e manter uma lista com as tarefas que podem ser executadas por um determinado usuário. Essa lista recebe o nome de ***Lista de Trabalho*** (*Worklist*) e as tarefas são denominadas ***Itens de Trabalho*** (*Workitems*).

Cada *workitem* corresponde a uma tarefa de um processo. Dessa forma, sempre que o estado de um *workitem* sofre alteração, o estado da tarefa correspondente também é atualizado. Essa relação garante que as alterações efetuadas pelos usuários sobre os *workitems* se refletem nos processos. Os *workitems* correspondem sempre a tarefas semi-automáticas ou manuais, já que as automáticas são tratadas de forma distinta (através de TMs) e não são adicionadas a *worklist*.

A Figura 3.7 mostra a estrutura de uma *worklist* e de um *workitem*. Eles se assemelham a *tasklist* e a *task definition*, respectivamente, com a diferença de que os primeiros se

referem às tarefas relativas a um usuário e os últimos às tarefas relativas a uma instância de processo. O campo `Task Name` indica o nome da tarefa correspondente ao *workitem* e `Process Name` o nome da instância de processo ao qual a tarefa pertence. Os demais campos são os mesmos existentes na *task definition.*

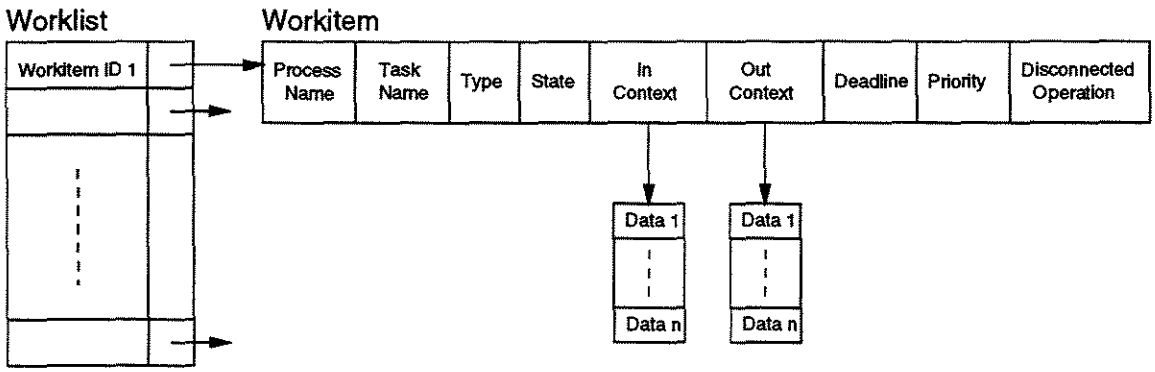


Figura 3.7: Representação da *Worklist* e do *Workitem.*

A *worklist* é armazenada no URM, no repositório de Usuários. Quando um WM é ativado, recupera junto ao URM uma cópia da *worklist* de seu respectivo usuário. Quando o usuário se desconecta do sistema a *worklist* é atualizada junto ao URM, pois pode ter sido modificada. Dessa forma, mesmo que um usuário se conecte ao sistema de diversas máquinas diferentes, sua *worklist* estará sempre disponível e atualizada. A *worklist* é também armazenada na máquina do usuário, devido à operação desconectada (Seção 3.4.6).

Existe um e somente um WM para cada usuário, executado sempre no *host* onde está seu usuário correspondente.

O Gerenciador de Lista de Trabalho possui um sub-componente, o ***Prefetcher***, que efetua o *prefetching* de *workitems*, como será visto na Seção 3.4.5.

Além disso, interage ainda com outro componente localizado na máquina do usuário, o ***Gerenciador de Recursos*** (*Resource Manager*), cuja função é controlar os recursos disponíveis em uma determinada máquina. Tais recursos incluem: espaço livre em disco, aplicações atualmente instaladas, tipo de hardware, sistema operacional, energia disponível (importante para dispositivos portáteis), largura de banda da atual conexão e velocidade média de transferência de dados remotos. A lista das operações do Gerenciador de Lista de Trabalho está no Apêndice B e as do Gerenciador de Recursos no Apêndice C.

As implementações das operações do Gerenciador de Recursos são dependentes de sistema operacional, já que elas necessitam de chamadas a funções básicas do SO. Por isso deve existir uma versão do Gerenciador de Recursos para cada SO (Windows, Linux, Solaris, MacOS e outros).

3.3.7 Modelo de Referência e Classificação

A arquitetura do WorkToDo se enquadra no Modelo de Referência para *Workflow* descrito na Seção 2.1.6:

- O PM, IM, DM e URM executam, em conjunto e de forma distribuída, as funcionalidades do Serviço de Execução, controlando e gerenciando a execução das instâncias de processos e suas respectivas tarefas;
- Os WMs correspondem ao componente Aplicações Cliente de *Workflow*, interagindo com seus respectivos usuários por meio das listas de trabalho;
- As chamadas às Aplicações Invocadas são efetuadas pelo TM ou pelo WM, dependendo do tipo de tarefa que utiliza a aplicação (automática ou não);
- As Ferramentas de Definição de Processos são implementadas pela PDL, que permite especificar processos de forma completa;
- As Ferramentas de Administração e Monitoramento estão disponíveis, ainda que de forma simplificada, através da interface específica ao Administrador, que permite a alteração das definições de papéis, a consultas aos históricos de instâncias já executadas e assim por diante;
- Somente a característica de interoperação com diferentes SGWFs não foi considerada, pelo fato de não ser esse o foco do trabalho.

Segundo as classificações apresentadas na Seção 2.1.5, o WorkToDo pode ser considerado um SGWF:

- Administrativo e/ou de Produção, pois é capaz de gerenciar ambos os tipos de processos. Processos Ad Hoc e Colaborativos fogem do escopo do sistema, pois necessitam de tratamentos específicos para algumas instâncias de um mesmo processo (Ad Hoc) e flexibilidade para alterar dinamicamente as regras de execução dos processos (Colaborativos), ambas características inexistentes no WorkToDo;
- Centrado em Processos, pois implementa um mecanismo de comunicação próprio, fornece interação com aplicações externas e utiliza repositórios de dados. Também poderia ser considerado em parte Centrado em Documentos, pois a troca de documentos entre tarefas é constante.

3.4 Funcionamento

Esta seção descreve o funcionamento básico do sistema WorkToDo, ressaltando os aspectos relativos à operação no ambiente de comunicação sem fio. Ao final são mostrados diagramas de seqüência relativos a algumas operações básicas do sistema.

3.4.1 Interação com os Usuários

Um usuário interage com o SGWF por meio de uma interface, que fornece mecanismos de comunicação com os diversos componentes do sistema. A interação ocorre principalmente através da *worklist*, pela qual o usuário pode efetuar as seguintes ações:

Visualizar *workitems*. A *worklist* mostra todos os *workitems* do usuário, listados de acordo com um de quatro tipos de ordenamento: por ordem de chegada, por prioridade, por prazo e por tamanho dos dados relacionados ao *workitem*. O usuário também pode mover *workitems* para a posição que desejar.

Selecionar *workitems*. Quando o usuário deseja executar um *workitem*, ele deve selecioná-lo para execução. Nesse momento, o PM correspondente é notificado e remove o *workitem* da *worklist* dos demais usuários daquele papel, avisando seus respectivos WMs. O WM do usuário recebe então uma confirmação e a partir daí o usuário pode executar o *workitem*. Esse processo de seleção evita que dois usuários executem um mesmo *workitem*.

Trancar *workitems*. Quando o usuário deseja executar um *workitem* em modo desconectado, ele deve trancá-lo[1] (como será visto na Seção 3.4.6). Dessa forma, o *workitem* será removido das *worklists* dos demais usuários daquele papel, exatamente como se o usuário o tivesse selecionado. Um *workitem* somente pode ser trancado se sua tarefa correspondente foi definida como sendo passível de operação desconectada – isto é, se o atributo `DISCONNECTED_OPERATION` da tarefa foi definido como `true`.

Executar *workitems*. Após ter selecionado ou trancado um *workitem*, o usuário executa esse *workitem*. Essa ação faz com que a aplicação referente ao *workitem* seja invocada (se houver uma aplicação relacionada ao *workitem*).

Habilitar o *prefetching*. O usuário pode escolher, quando conectado ao SGWF, se deseja que o *prefetching* dos *workitems* (Seção 3.4.5) seja ou não efetuado pelo Prefetcher. Os *workitems* são copiados de acordo com a ordem que o usuário estabeleceu para os *workitems*.

Além disso, a interface permite também que um usuário verifique quais são os processos em execução e consultar o estado dos processos em que ele participa. Ainda é possível criar instâncias de um determinado tipo de processo, desde que o usuário pertença ao papel criador daquele tipo.

[1] Do termo em inglês *lock*.

Para o SGWF, os usuários podem estar ***ativos*** ou ***inativos***. Um usuário ativo está conectado ao SGWF, podendo receber notificações de tarefas prontas para executar, selecionar tarefas para execução, criar processos, e consultar estados de processos, entre outras operações. Um usuário inativo não está conectado ao SGWF, portanto não pode interagir com componentes do sistema que exijam chamadas remotas. Entretanto, são capazes de interagir normalmente com seus respectivos Gerenciadores de Lista de Trabalho, conforme será visto na Seção 3.4.6.

Administrador

Existe um tipo especial de usuário, denominado administrador, que possui alguns privilégios em relação aos demais usuários. Esses privilégios são tanto de ordem administrativa quanto gerencial. É capaz de:

- Incluir, remover e consultar usuários e papéis;
- Incluir/Remover usuários em/de papéis;
- Verificar quais são os usuários pertencentes a cada papel;
- Verificar quais são os usuários ativos em dado momento;
- Verificar as tarefas executadas por cada usuário;
- Verificar a quantidade de itens atribuídos e selecionados por cada usuário;
- Incluir, remover, consultar e modificar tipos de processo, modelos de tarefas e aplicações;
- Consultar o estado de todos os processos, tanto os que estão em execução quanto os que já terminaram.

O administrador é na verdade um papel, que pode ser assumido por diversos usuários.

Responsável pelo Processo

Sempre que uma instância de processo é criada, é atribuído a ela um usuário responsável. Esse usuário deve acompanhar o andamento do processo, tomando certas decisões quando necessário. O sistema notifica o usuário a respeito de prazos de tarefas que serão ou estão sendo executadas, de tarefas que falharam em sua execução, bem como do início e término da execução do processo. O usuário se responsabiliza por efetuar as medidas que considerar adequadas.

Também cabe ao usuário responsável a decisão do que fazer quando a execução de uma tarefa falha e a especificação do processo não define uma ação a ser tomada.

3.4.2 Notificação de Tarefas

Como mencionado anteriormente, quando o Despachante detecta que uma tarefa semi-automática ou manual está pronta para executar, ele notifica um grupo de usuários, de acordo com a política de notificação estabelecida. Essa notificação é realizada através dos Gerenciadores de Lista de Trabalho, que recebem a notificação do Despachante e adicionam um *workitem* na *worklist* de seu usuário.

Os usuários notificados nesse momento, entretanto, são apenas aqueles que estão ativos, visto que o Despachante não é capaz de se comunicar com os WMs de usuários inativos. Como em um ambiente de comunicação sem fio as conexões e desconexões de usuários são muito freqüentes, é possível que não existam muitos usuários ativos entre aqueles que devem ser notificados. Isso faz com que a tarefa seja adicionada na *worklist* de um número reduzido de usuários, pois os usuários que se conectam após a notificação não recebem o *workitem*. A conseqüência disso é que o tempo para a execução da tarefa pode ser maior, pois a probabilidade de que a tarefa seja selecionada por um usuário é menor.

Por esse motivo, a notificação de tarefas prontas para executar não é efetuada somente no momento em que o Despachante detecta que a tarefa está pronta. Periodicamente o Despachante verifica se existe algum novo usuário conectado ao sistema que deveria ser notificado a respeito da disponibilidade da tarefa. Se existe, o Despachante notifica o WM correspondente. Dessa forma, todos os usuários que deveriam ser notificados o são, assim que se conectam ao SGWF. A partir do momento em que a tarefa é selecionada ou trancada por algum usuário essa notificação não é mais efetuada, pois nesse caso não é mais útil.

3.4.3 Prazos para Tarefas

Todas as tarefas de um processo possuem um prazo de execução, especificado na definição do modelo de tarefa (ver Seção 3.2.2), que representa o tempo máximo para que uma tarefa seja executada. À medida que o tempo passa, o valor desse prazo é decrementado, e quando ele chega a zero significa que o prazo máximo foi alcançado. Caso o prazo de uma tarefa seja ultrapassado, o responsável pelo processo decide se a execução do processo deve continuar ou ser interrompida.

É possível definir um conjunto de valores V, de tal forma que quando o valor do prazo é igual a um dos valores de V uma mensagem de prazo é gerada, conforme será visto na próxima Seção. Além disso, é definido um valor específico entre os valores de V, denominado ***Valor de Transferência***. Quando o valor do prazo é igual ao valor de transferência, a execução da tarefa é atribuída automaticamente ao responsável pelo processo, mesmo que ela tenha sido selecionada por algum usuário. Isso evita que uma tarefa

não seja executada dentro de seu prazo estipulado porque nenhum usuário a selecionou ou porque o usuário que a selecionou não a executa. Tanto os valores de V quanto o valor de transferência são descritos em forma de horas e definidos no momento da criação de uma instância de processo. Um exemplo de um conjunto V é $\{6, 12, 24\}$ (quando faltarem 24, 12 e 6 horas para o prazo da tarefa ser atingido, uma mensagem de prazo é gerada).

É importante notar que, se o *workitem* correspondente à tarefa está selecionado por algum usuário no momento em que o valor de transferência é atingido, o *workitem* é removido da *worklist* desse usuário. O usuário não tem mais controle sobre a execução da tarefa, pois agora ela é responsabilidade do responsável pelo processo.

Existe a possibilidade de o responsável estar desconectado no momento em que o valor de transferência de um *workitem* é atingido. Nesse caso, o responsável será notificado a respeito do *workitem* assim que se reconectar.

O controle dos prazos das tarefas é efetuado pelos PMs aos quais elas pertencem. Entretanto, como visto anteriormente, um usuário trabalhando em modo desconectado não é capaz de interagir com componentes remotos, não recebendo portanto as mensagens de prazo da tarefa enviadas pelo PM. Isso torna necessária a existência de um controle local sobre os prazos. Assim, quando um usuário está em modo desconectado, o WM é quem exerce controle sobre os prazos das tarefas presentes na *worklist* do usuário, mostrando diretamente para o usuário as mensagens que seriam enviadas pelo PM. Isso garante que o usuário é sempre notificado a respeito do prazo de tarefas, independentemente de estar conectado ou desconectado. A transferência de tarefas para o responsável, porém, continua sendo efetuada normalmente pelo PM.

3.4.4 Mensagens

Como descrito na Seção 3.3.1, o Gerenciador de Usuários e Papéis mantém uma lista de mensagens para cada usuário, contendo notificações importantes do SGWF sobre a ocorrência de eventos. Existem quatro tipos de mensagens, uma para cada tipo de evento:

Início de processo. Notifica o responsável pelo processo de que este foi iniciado, ou seja, suas tarefas começaram a ser executadas.

Término de processo. Notifica o responsável pelo processo de que este teve sua execução completada.

Falha de tarefa. Notifica o responsável pelo processo de que ocorreu alguma falha durante a execução da tarefa, impedindo que ela fosse executada com sucesso. O responsável pode então verificar qual foi o problema e tomar as medidas que considerar adequadas (cancelar o processo ou executar a tarefa novamente, por exemplo).

Prazo de tarefa. Notifica o responsável pelo processo e o usuário que atualmente está com a tarefa selecionada ou trancada (se houver algum) de que o prazo para execução da tarefa está se aproximando.

Como pode ser notado, os três primeiros eventos geram mensagens somente para o usuário responsável pelo processo, enquanto o último gera uma mensagem também para o usuário que tem a tarefa selecionada ou trancada.

As mensagens para cada usuário são armazenadas mesmo quando o usuário está inativo. Nesse caso, quando o usuário se reconecta recebe todas as mensagens armazenadas. Um usuário ativo recebe as mensagens no momento em que elas são enviadas.

A estrutura de uma mensagem é mostrada na Figura 3.8. O campo `type` identifica o tipo da mensagem, conforme descrito acima. O campo `sender` indica o remetente da mensagem (o nome de uma instância em execução). O campo `time` contém a data e hora de envio da mensagem. E finalmente, no campo `text` está o texto da mensagem.

Figura 3.8: Estrutura das mensagens enviadas aos usuários.

3.4.5 Prefetching

A técnica de *prefetching* consiste em copiar para a máquina do usuário um *workitem* que mais tarde pode vir a ser executado de forma desconectada (Seção 3.4.6). Assim, quando esse *workitem* for trancado, as informações necessárias para sua execução já estarão disponíveis antes mesmo de sua seleção, minimizando o tempo de espera. O *prefetching* é sempre efetuado em *background*, de forma transparente para o usuário, aproveitando a largura de banda disponível.

O *prefetching* ocorre da seguinte forma: enquanto o usuário está conectado ao SGWF, o WM, através do Prefetcher, verifica periodicamente se a *worklist* contém algum *workitem* que pode ser executado de forma desconectada. Em caso afirmativo, o WM copia o *workitem* para a máquina do usuário. Copiar um *workitem* significa copiar tanto os dados quanto a aplicação relacionados a ele (se existirem). Quando a cópia é completada, o *workitem* é dito ***transferido***. A aplicação somente é copiada se ela não está instalada no computador do usuário.

Todos os *workitems* que podem ser executados de forma desconectada são incluídos no *prefetching* e a ordem com que são copiados é dada de acordo com seu ordenamento na *worklist*. Entretanto, um *workitem* trancado ganha prioridade sobre os demais e seu *prefetching* é efetuado primeiro. Tal medida evita que um *workitem* trancado – e que

portanto tem grandes chances de ser incluído em uma execução desconectada – tenha seu *prefetching* atrasado por um outro *workitem* que não foi trancado e assim não deve participar de uma execução desconectada.

Podem ocorrer casos em que os dados copiados jamais são utilizados, pois o *workitem* correspondente não é trancado pelo usuário. Nesses casos os dados ficariam ocupando espaço na máquina do usuário sem motivo. Por isso, quando um *workitem* transferido é removido da *worklist* do usuário os dados relativos ao *workitem* são também removidos.

3.4.6 Operação Desconectada

No WorkToDo, um usuário participante de um processo pode executar tarefas sem estar conectado ao SGWF. O usuário escolhe um ou mais *workitems* de sua *worklist*, desconecta-se do sistema e, após executar os *workitems* (minutos, horas ou dias mais tarde), conecta-se novamente ao sistema e os resultados de seu trabalho são repassados ao sistema.

O objetivo do WorkToDo é permitir a operação desconectada e torná-la o mais transparente possível para os usuários. Isso significa que as diferenças entre a operação normal (conectada) e a desconectada devem ser reduzidas ao máximo. Em modo desconectado, as únicas operações válidas serão aquelas invocadas junto ao Gerenciador de Lista de Trabalho, que é local ao usuário. Nesse caso, o usuário pode acessar seus *workitems* da mesma forma como quando está conectado ao SGWF. Podemos dizer então que um usuário está operando em modo desconectado quando ele está inativo e acessa sua *worklist*.

O WorkToDo oferece suporte à operação desconectada planejada. Isto significa que quando o usuário pretende trabalhar de forma desconectada ele notifica o sistema, que toma as medidas necessárias. A estratégia adotada para efetuar essa notificação foi a de ***trancamento*** de tarefas, na qual o usuário reserva *workitems* para si. Durante a operação desconectada o usuário pode visualizar e modificar sua *worklist* normalmente, além de selecionar e executar *workitems* da mesma forma como quando está conectado. A única diferença é que somente aparecem na *worklist* aqueles *workitems* que foram trancados antes da desconexão; os demais ficam desabilitados.

Para poder executar um *workitem* em modo desconectado é necessário que os dados relativos ao *workitem* estejam disponíveis localmente. Por isso, antes da desconexão esses dados devem ser copiados para a máquina do usuário. Essa cópia é efetuada no momento da desconexão do usuário (ou antecipadamente, se considerarmos o *prefetching*). Além disso, quando o usuário se reconecta, os *workitems* executados devem ser repassados ao SGWF para que seus resultados sejam armazenados. O WorkToDo define então dois protocolos, um para a desconexão e outro para a reconexão dos usuários.

Protocolo de Desconexão

1. O usuário notifica seu WM de que deseja se desconectar do SGWF, estabelecendo o tempo máximo que aceita esperar até a desconexão;
2. O WM verifica se o usuário possui algum *workitem* selecionado para execução. Se possui, pergunta ao usuário se ele realmente deseja desconectar;
3. O WM notifica o URM de que o usuário está se desconectando;
4. O URM marca o usuário como inativo;
5. O WM calcula o tamanho de cada *workitem* trancado e não transferido, definido como a soma do tamanho da aplicação e dos dados relacionados ao *workitem* (se existirem). A aplicação somente é considerada caso ela não esteja instalada na máquina do usuário;
6. O WM calcula o volume total de dados a transferir, definido como a soma dos tamanhos dos *workitems* trancados e não transferidos;
7. O WM verifica se a largura de banda da conexão permite que o volume total de dados seja transferido dentro do tempo estabelecido pelo usuário. Em caso negativo, o sistema indica ao usuário o tempo necessário e permite que o usuário aceite esse tempo ou destranque algum *workitem*, retornando a seguir ao item 5;
8. O WM verifica se há espaço na máquina do usuário para abrigar o volume total de dados. Em caso negativo, o sistema notifica o usuário para que este solucione o problema (liberando espaço em disco, por exemplo) ou destranque algum *workitem*, retornando a seguir ao item 5;
9. O WM copia cada *workitem* trancado e não transferido;
10. O WM desabilita todos os *workitems* que não estão trancados;
11. O WM notifica o usuário, através da interface, de que ele está desconectado do sistema.

Como se trata de um ambiente de comunicação sem fio, a qualquer momento durante a execução desse protocolo podem ocorrer quedas de conexão, tanto por falhas como por decisão do usuário. Nesse caso, quando o usuário reconecta a transferência dos dados é retomada a partir do ponto onde foi interrompida.

Além disso, um *workitem* transferido pode ser executado em modo desconectado mesmo se a execução do protocolo de desconexão for interrompida (ou mesmo se não for executada), pois ele já foi trancado (evitando que outros usuários o executem) e todos os dados que ele necessita estão na máquina do usuário. Dessa forma, é possível que um usuário trabalhe de forma desconectada mesmo na ocorrência de desconexões não planejadas.

Quando ocorre uma desconexão não planejada, o WM detecta a desconexão e desabilita todos os *workitems* que não estão trancados, bem como aqueles que estão trancados mas não estão transferidos. Isso garante que somente permanecerão habilitados os *workitems* que estão aptos a serem executados pelo usuário.

Protocolo de Reconexão

1. O usuário notifica seu respectivo WM de que deseja se conectar ao SGWF;
2. O WM notifica o URM de que o usuário está se conectando;
3. O URM marca o usuário como ativo;
4. O SGWF notifica o usuário, através da interface, de que ele está conectado e de que os resultados da execução dos *workitems* serão transferidos para o SGWF;
5. O WM habilita todos os *workitems*;
6. Para cada *workitem* executado em modo desconectado, o WM:
   (a) Transfere em *background* os dados resultantes da execução ao PM correspondente;
   (b) Remove o *workitem* da *worklist*.

No momento da desconexão, os *workitems* que não estavam trancados são desabilitados; por isso o item 5 os habilita novamente. Podem existir *workitems* desatualizados, ou seja, que já foram selecionados, trancados ou mesmo executados por outros usuários. Por isso, na próxima interação entre WM e PM a *worklist* é atualizada, momento no qual alguns *workitems* podem ser removidos. Se antes dessa atualização o usuário tentar selecionar um *workitem* inválido o PM o notificará de que o *workitem* não está mais disponível.

Como no protocolo anterior, podem ocorrer desconexões durante a execução do protocolo e, da mesma forma, quando o usuário se reconecta a transferência dos dados é retomada do ponto onde foi interrompida. Isto diminui o tempo de retardo mediante a presença de falhas.

Considerações

É importante ressaltar que, mesmo em um SGWF que suporta operação desconectada, nem todas as tarefas podem ser executadas em modo desconectado [9].

Tarefas automáticas obviamente não devem ser consideradas, exatamente por não necessitarem da participação de usuários. Mas mesmo as tarefas semi-automáticas e manuais devem ser analisadas com cuidado. O tamanho excessivo de algumas aplicações e dados, aliado à baixa largura de banda da conexão do usuário, podem tornar sua transferência extremamente demorada.

Além disso, podem existir dados que são utilizados por um grande número de tarefas. Se tais dados participarem de uma operação desconectada na qual são alterados, eles não devem ser utilizados por outras tarefas enquanto a operação desconectada não for concluída. Como a operação desconectada normalmente é de longa duração, isso provoca um retardo excessivo na execução de muitas tarefas, prejudicando o desempenho do SGWF como um todo.

Outro aspecto relevante é que podem existir tarefas que possuem muitas tarefas dependentes; se uma tarefa desse tipo for executada em modo desconectado pode demorar muito para ser concluída, atrasando em demasia a execução do processo.

Todas essas particularidades devem ser levadas em consideração na modelagem dos processos. Faz-se necessária uma análise da relação custo-benefício de tornar as tarefas passíveis de execução desconectada, sob pena de degradar o desempenho do sistema e não obter os resultados esperados. A função do modelador, portanto, é de extrema importância nesse aspecto.

3.4.7 Operação Semi-Conectada

A operação semi-conectada ocorre quando um usuário acessa o sistema através de uma rede sem fio (por exemplo, um telefone celular). O usuário executa as tarefas da mesma forma como se estivesse em uma conexão convencional, com a diferença de que a conexão é instável e possui baixa largura de banda, tornando falhas e desconexões muito mais freqüentes e dificultando a transferência de dados.

Durante a operação semi-conectada podem existir períodos nos quais há pouca ou nenhuma comunicação entre o usuário e o SGWF. Esses períodos são utilizados para *prefetching*, como explicado anteriormente, aproveitando o máximo possível da largura de banda da conexão.

Assim, enquanto o usuário está conectado, os *workitems* que podem ser executados de forma desconectada vão sendo transferidos. É possível, portanto, que quando o usuário selecionar um *workitem* para execução semi-conectada todos os seus dados já estejam disponíveis localmente. Por isso, quando o usuário seleciona um *workitem* em modo semi-conectado, o WM verifica se esse *workitem* já foi transferido. Em caso afirmativo, a execução se dá sobre os dados locais e após a reconexão os dados gerados são transferidos para o SGWF. Caso contrário a execução utiliza os dados remotos.

A execução sobre os dados locais representa uma economia em termos de comunicação entre a máquina do usuário e o SGWF, economia esta que pode ser ser bastante grande, além de mascarar e diminuir prováveis falhas na comunicação devido à qualidade da conexão. Por outro lado, se a execução do *workitem* gera muitos dados de saída, o custo para transferir os dados após a execução será muito alto; nesse caso, a execução sobre

dados remotos seria mais eficiente. Este *tradeoff* deve ser levado em consideração para maximizar a eficiência. Por isso, com base na largura de banda da conexão e no volume de dados gerados, o sistema recomenda o usuário sobre qual execução é mais adequada.

No caso de queda na conexão, o WM desabilita tanto os *workitems* que não estão trancados quanto os que estão trancados mas não foram transferidos, permanecendo habilitados somente aqueles *workitems* trancados e transferidos. Dessa forma o usuário pode continuar trabalhando normalmente nesses *workitems*, da mesma forma como na desconexão planejada. Se o usuário tinha algum *workitem* sendo executado no momento da queda de conexão, existem três casos a serem considerados:

1. Se o *workitem* estava sendo executado sobre os dados locais, sua execução pode prosseguir normalmente. A execução do *workitem* ocorre como se ele estivesse sendo executado em modo desconectado, sendo que seus resultados serão informados ao SGWF quando o usuário se reconectar;
2. Se o *workitem* utiliza dados remotos, o usuário não poderá acessá-los enquanto não se reconectar, logo a execução da tarefa é temporariamente paralisada. Quando o usuário se reconecta, os dados se tornam novamente acessíveis e a execução pode ser retomada;
3. Se o *workitem* utiliza uma aplicação remota, a execução da aplicação será terminada. Cabe ao usuário, após se reconectar, reiniciar a aplicação e retomar a execução da tarefa.

É importante notar que a operação conectada, ou seja, através de uma conexão confiável e de grande largura de banda (como por exemplo uma rede local), se dá de forma quase idêntica à operação semi-conectada. A única diferença é que o *prefetching* não é realizado, portanto *workitems* selecionados nunca são executados sobre dados locais.

3.4.8 Diagramas de Seqüência

Esta seção apresenta os diagramas de seqüência das principais operações do WorkToDo, com o objetivo de tornar mais claro o funcionamento do sistema e discutir alguns detalhes importantes.

As operações sempre se iniciam por meio de uma requisição do usuário à interface do sistema, que então encaminha a requisição ao componente adequado. Ao final, a interface retorna uma notificação ao usuário, indicando o sucesso ou falha da operação.

Criação de uma Instância de Processo

O procedimento para criação de uma instância, mostrado na Figura 3.9, é detalhado a seguir:

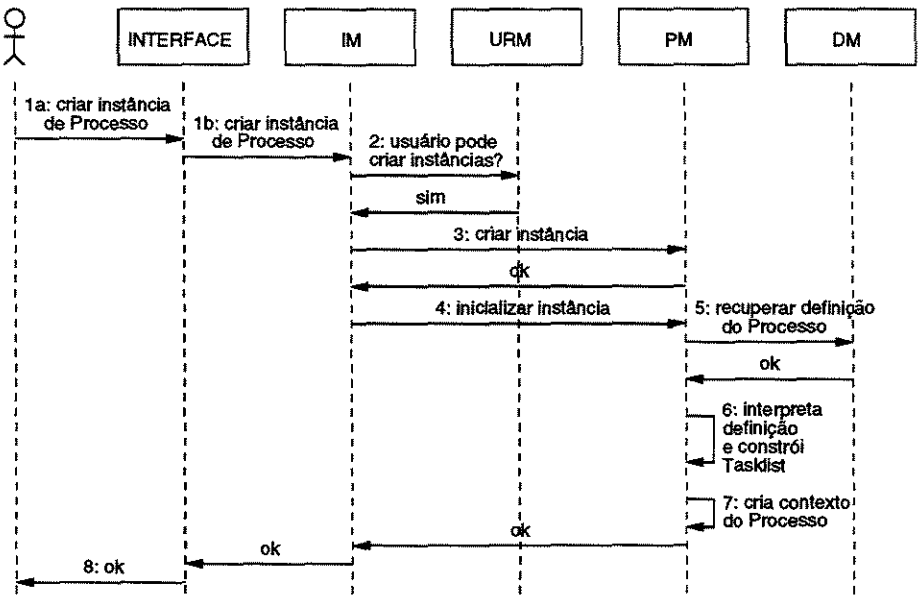


Figura 3.9: Diagrama de seqüência para a criação de uma instância de processo.

1. Através da interface do sistema, o usuário requisita ao IM que seja criada uma instância de processo, informando o tipo do processo e o *host* a partir do qual a instância será gerenciada;
2. O IM verifica, junto ao URM, se o usuário pertence ao papel criador daquele tipo de processo;
3. O IM cria um novo PM no *host* especificado pelo usuário;
4. O IM inicializa a instância, ou seja, informa ao PM recém-criado qual o tipo de processo que este irá gerenciar;
5. O PM recupera junto ao DM a definição do tipo de processo;
6. O PM ativa o PDLI, que interpreta a definição do tipo de processo da seguinte forma. Para cada tarefa instanciada na definição, o PDLI interpreta a definição do modelo de tarefa correspondente e constrói uma *task definition*. Ao final, todas as *task definitions* são inseridas em uma lista, formando a *tasklist* do processo;
7. O PM cria o contexto do processo no *host* onde está sendo executado. O PM cria um diretório com o nome da instância, onde são armazenadas as informações relativas à instância (como por exemplo seu estado atual). Os dados utilizados pelas tarefas também são copiados para esse diretório à medida que são necessários;
8. A interface notifica o usuário de que a instância foi criada com sucesso e que sua execução já foi iniciada.

Se qualquer um dos itens entre 2 e 7 falha, a criação da instância é automaticamente cancelada e a interface informa o usuário a respeito da falha na operação. As ações que porventura tenham sido efetuadas (construção da *Tasklist*, criação do contexto) são desfeitas.

Seleção de um Workitem

O procedimento para seleção de um *workitem*, mostrado na Figura 3.10, é detalhado a seguir:

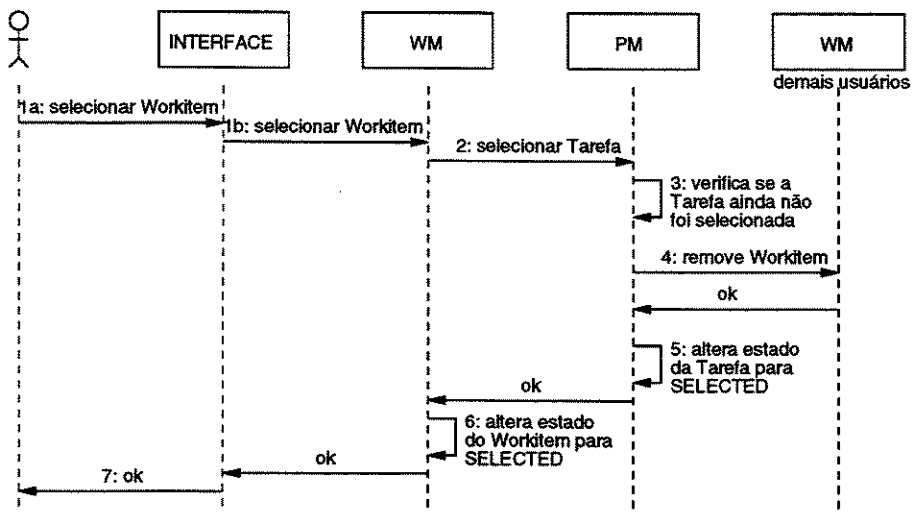


Figura 3.10: Diagrama de seqüência para a seleção de um *workitem*.

1. Através da interface do sistema, o usuário requisita ao WM a seleção de um determinado *workitem*;
2. O WM verifica qual a tarefa correspondente ao *workitem* e envia uma requisição ao PM ao qual a tarefa pertence solicitando a seleção da tarefa para execução;
3. O PM verifica se a tarefa ainda está disponível (algum outro usuário pode tê-la selecionado);
4. O PM remove os *workitems* correspondentes à tarefa das *worklists* dos demais usuários que haviam sido notificados da disponibilidade da tarefa;
5. O PM altera o estado da tarefa para SELECTED, indicando que ela foi selecionada por um usuário. Além disso, armazena na *task definition* o nome do usuário que efetuou a seleção;
6. O WM altera o estado do *workitem* para SELECTED;

7. A interface notifica o usuário de que o *workitem* foi selecionado com sucesso e está pronto para ser executado.

Se um dos itens entre 2 e 3 falha, a seleção do *workitem* é automaticamente cancelada e a interface informa o usuário a respeito da falha na operação. No item 4, se algum usuário não puder ser contactado (se estiver desconectado, por exemplo), a seleção do *workitem* é efetuada normalmente; o *workitem* será removido da *worklist* desse usuário na próxima interação entre seu WM e o PM.

É interessante observar que o procedimento para trancamento de um *workitem* é análogo ao mostrado acima. A única diferença é que os estados do *workitem* e da tarefa correspondente são alterados para `LOCKED` ao invés de `SELECTED`.

Outro ponto importante é que um usuário somente é capaz de selecionar e trancar *workitems* quando está ativo, caso contrário o PM correspondente não poderá ser notificado da seleção/trancamento.

Reconexão de um Usuário

O procedimento para reconexão de um usuário, mostrado na Figura 3.11, é detalhado a seguir:

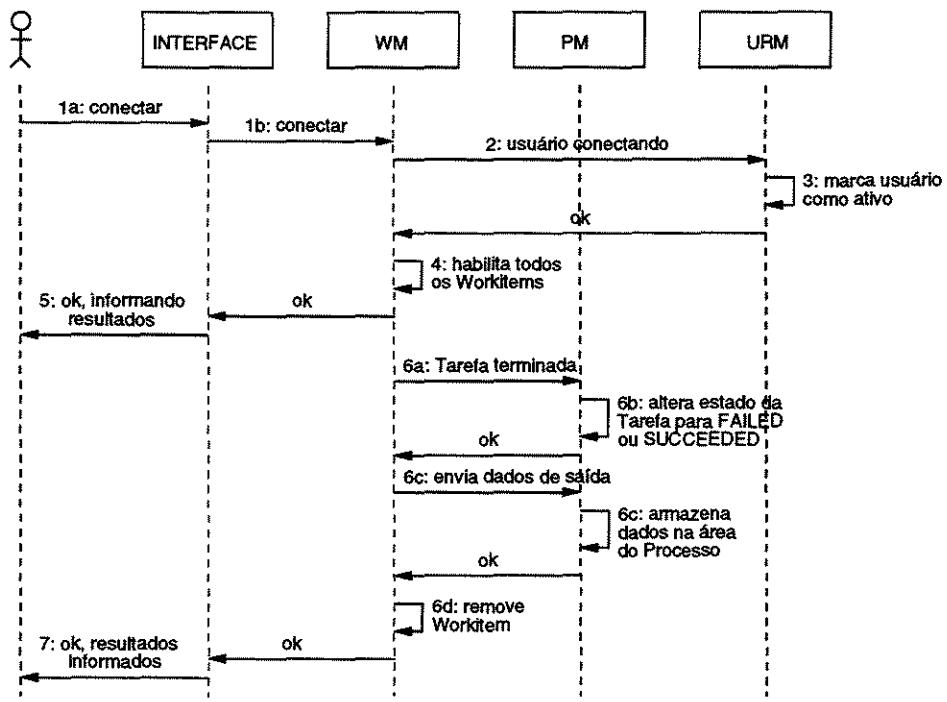


Figura 3.11: Diagrama de seqüência para a reconexão de um usuário.

1. Através da interface do sistema, o usuário informa seu WM de que deseja se conectar ao SGWF;
2. O WM informa o URM de que o usuário está se conectando ao SGWF;
3. O URM altera a situação do usuário para ativo;
4. O WM habilita todos os *workitems* do usuário, inclusive aqueles que haviam sido desabilitados na desconexão;
5. A interface notifica o usuário de que a conexão foi efetuada com sucesso e de que os resultados dos *workitems* executados em modo desconectado (se existirem) serão repassados ao SGWF;
6. Para cada *workitem* executado:
   (a) O WM notifica o PM ao qual pertence a tarefa correspondente ao *workitem* de que a tarefa foi completada, informando o resultado (`FAILED` ou `SUCCEEDED`) de acordo com o que foi informado pelo usuário;
   (b) O PM verifica se a tarefa havia realmente sido trancada ou selecionada pelo usuário que está agora informando o resultado. Em caso afirmativo, altera o estado da tarefa de acordo com o informado pelo WM;
   (c) O WM envia os dados gerados pela execução do *workitem* para o PM, que os armazena na área do processo;
   (d) O WM remove o *workitem* da *worklist*;
7. A interface informa o usuário de que os resultados dos *workitems* já foram repassados.

Como pode ser visto, o diagrama de seqüência contém, entre outras ações, o protocolo de reconexão descrito anteriormente.

Enquanto os itens 6 e 7 são executados, o usuário pode interagir com seu WM e com o SGWF normalmente. Portanto, esses itens são executados em *background*, de forma transparente para o usuário.

O WM mantém um registro com os dados que já foram enviados ao PM. Assim, se ocorrem falhas durante o repasse dos resultados, o procedimento pode ser retomado do ponto onde foi interrompido. Quando um *workitem* é removido da *worklist*, significa que todas as informações e dados necessários já foram repassados ao respectivo PM.

Desconexão de um Usuário

O procedimento para desconexão de um usuário, mostrado na Figura 3.12, é detalhado a seguir:

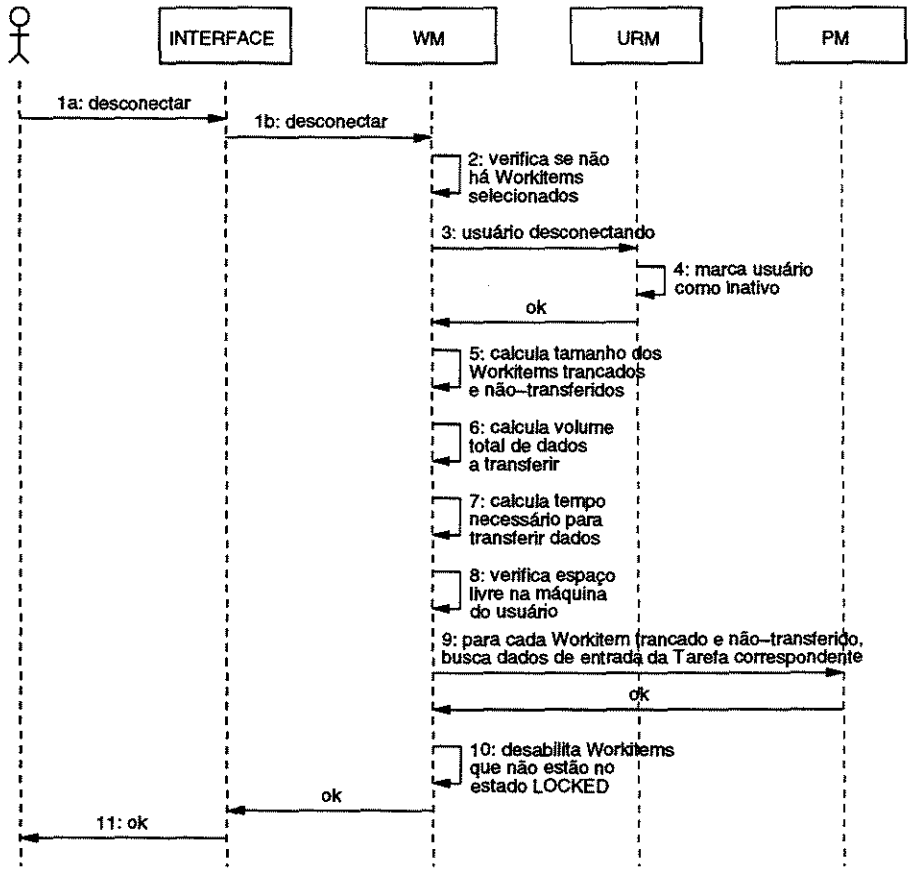


Figura 3.12: Diagrama de seqüência para a desconexão de um usuário.

1. Através da interface do sistema, o usuário informa seu WM de que deseja se desconectar do SGWF, estabelecendo também o tempo máximo que aceita até a desconexão;
2. O WM verifica se o usuário possui algum *workitem* selecionado, caso em que aguarda uma confirmação do usuário para desconectar;
3. O WM informa o URM de que o usuário está se desconectando do SGWF;
4. O URM altera a situação do usuário para inativo;
5. O WM calcula o tamanho de cada *workitem* trancado e não transferido, de acordo com os dados e a aplicação utilizados pelo *workitem*;
6. O WM calcula o volume total de dados a serem transferidos do SGWF para a máquina do usuário;
7. O WM consulta o Gerenciador de Recursos para que este verifique a velocidade da conexão do usuário e calcule o tempo necessário para transferir esse volume de dados. O WM verifica se esse tempo é maior que o tempo estabelecido pelo usuário

– se for, a interface notifica o usuário para que este aceite o tempo necessário ou destranque *workitems* e retorna ao item 5;

8. O WM consulta o Gerenciador de Recursos para que este verifique se há espaço livre em disco na máquina do usuário para copiar todo o volume de dados. Se não houver, a interface notifica o usuário para que este libere espaço em disco ou detranque *workitems* e retorna ao item 5;
9. Para cada *workitem* trancado e não transferido, o WM busca no respectivo PM os dados e a aplicação referentes à tarefa que corresponde ao *workitem* e armazena esses dados na máquina do usuário;
10. O WM desabilita todos os *workitems* que não estão trancados (no estado `LOCKED`), já que estes não poderão fazer parte da operação desconectada;
11. A interface informa o usuário de que a desconexão foi efetuada com sucesso.

Como pode ser visto, o diagrama de seqüência contém o protocolo de desconexão descrito anteriormente, apenas com alguns detalhamentos.

O tempo de execução desse procedimento depende principalmente dos itens entre 5 a 9. Isso porque, se não existem *workitems* trancados, ou se todos os *workitems* trancados já foram transferidos, esses itens serão executados instantaneamente. Surge assim a importância do *prefetching*: quanto maior o número de *workitems* que foram copiados com antecedência, menor o tempo de execução do procedimento acima. A variação pode ser bastante grande, visto que podem existir *workitems* de tamanho considerável, ou então um grande número de *workitems* a serem transferidos.

Novamente, o WM mantém um registro com os dados que já foram recuperados junto ao PM, para que na ocorrência de falhas o procedimento possa ser retomado de onde foi interrompido. Quando um *workitem* é transferido para a máquina do usuário, ele pode ser executado em modo desconectado mesmo que o procedimento não tenha sido executado por completo, já que todos os dados necessários já estarão disponíveis localmente.

Execução em Modo Conectado

O procedimento para execução de um *workitem* em modo conectado (ou semi-conectado), mostrado na Figura 3.14, é detalhado a seguir:

1. Através da interface do sistema, o usuário indica ao WM de que deseja executar um *workitem*;
2. O WM verifica se o *workitem* está selecionado ou trancado pelo usuário. Se não estiver, não permite a execução;
3. O WM verifica se o *workitem* já foi transferido. Se estiver, marca o *workitem* como "execução local", caso contrário marca como "execução remota";

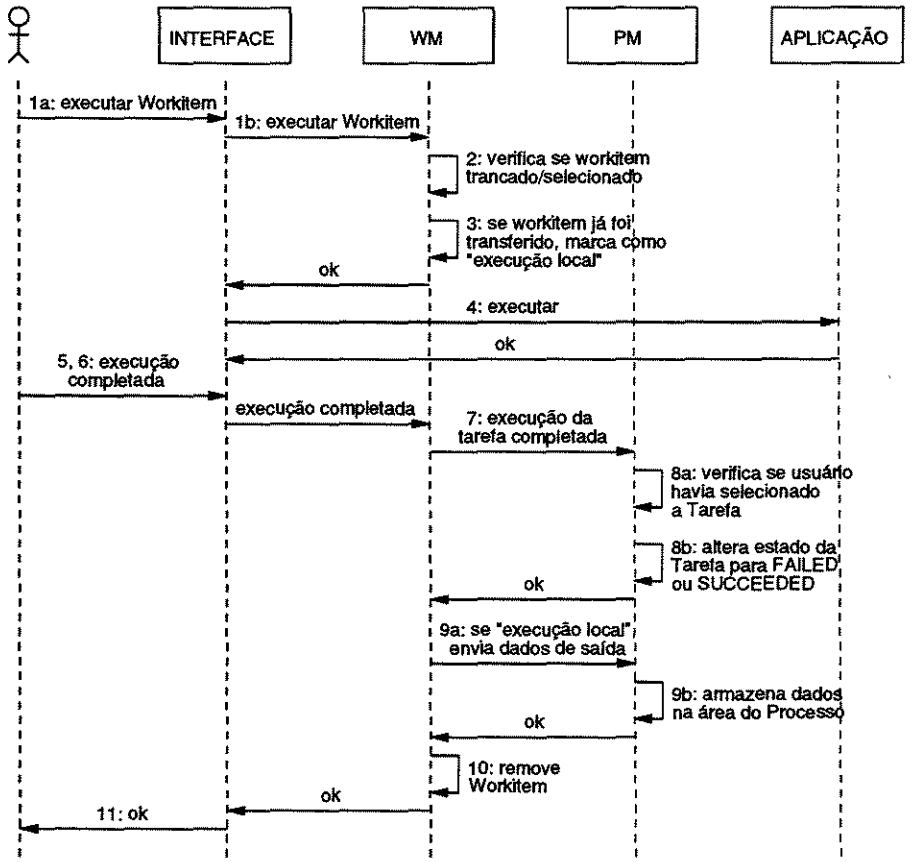


Figura 3.13: Diagrama de seqüência para a execução de um *workitem* em modo conectado/semi-conectado.

4. Se a tarefa for semi-automática, a interface automaticamente invoca a aplicação correspondente com os dados de entrada (se existirem). A aplicação é executada localmente ou remotamente, conforme a marcação do *workitem*;
5. O usuário executa a tarefa, desempenhando as ações especificadas;
6. Através da interface, o usuário notifica seu WM de que completou a execução do *workitem*, informando o resultado da execução (sucesso ou falha);
7. O WM notifica o PM ao qual pertence a tarefa correspondente ao *workitem* de que a tarefa foi completada, repassando o resultado informado pelo usuário;
8. O PM verifica se a tarefa havia realmente sido selecionada pelo usuário que está agora informando o resultado. Em caso afirmativo, altera o estado da tarefa de acordo com o informado pelo WM;
9. Se o *workitem* foi marcado como "execução local", o WM envia os dados gerados pela execução do *workitem* para o PM, que os armazena na área do processo;

10. O WM remove o *workitem* da *worklist*;
11. A interface notifica o usuário de que o resultado da execução foi informado com sucesso.

No caso do *workitem* ter sido marcado como "execução remota", a escrita dos dados é efetuada remotamente, sem utilizar cópias locais, sendo portanto desnecessário transferir os dados para o PM após a execução do *workitem*.

Execução em Modo Desconectado

O procedimento para execução de um *workitem* em modo desconectado, mostrado na Figura 3.14, é detalhado a seguir:

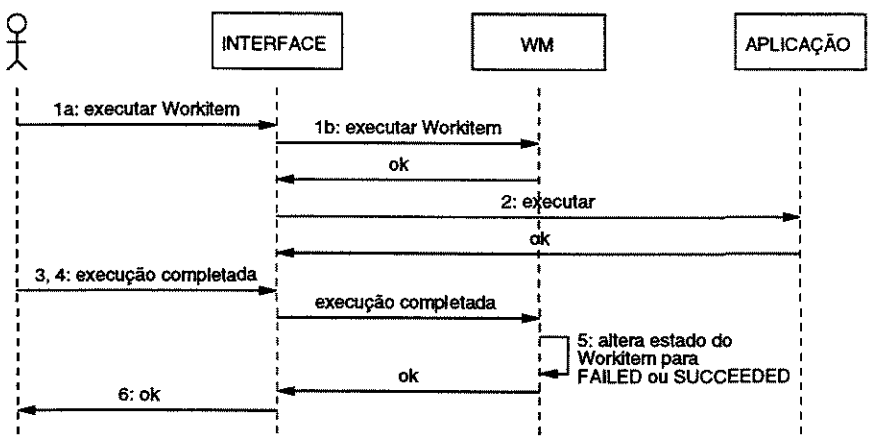


Figura 3.14: Diagrama de seqüência para a execução de um *workitem* em modo desconectado.

1. Através da interface do sistema, o usuário informa o WM de que deseja executar um *workitem*;
2. Se a tarefa for semi-automática, a interface invoca automaticamente a aplicação correspondente com os dados de entrada (se existirem);
3. O usuário executa a tarefa, desempenhando as ações especificadas;
4. Através da interface, o usuário notifica seu WM de que completou a execução do *workitem*, informando o resultado da execução (sucesso ou falha);
5. O WM altera o estado do *workitem* de acordo com o informado pelo usuário, armazenando localmente os dados gerados como resultado;
6. A interface notifica o usuário de que o resultado da execução foi informado com sucesso e será repassado ao PM correspondente assim que o usuário reconectar ao SGWF.

Como o usuário está inativo, somente aparecerão na *worklist* os *workitems* que foram previamente trancados e cujos dados estão disponíveis localmente. Por esse motivo o WM não precisa verificar se o *workitem* indicado pelo usuário está realmente trancado.

Quando o usuário se reconectar, o resultado da execução e os dados gerados serão repassados ao PM correspondente, como mostrado no diagrama de seqüência da conexão de um usuário. Enquanto a reconexão não ocorre, o *workitem* permanece na *worklist*, com seu estado final (`SUCCEEDED` ou `FAILED`) e desabilitado.

3.5 WorkToDo x Requisitos de SGWFs em Ambientes de Comunicação sem Fio

Na Seção 2.4, descrevemos os requisitos extras que um Sistema de Gerenciamento de *Workflows* deve atender para fornecer suporte à operação em ambientes sem fio. O WorkToDo atende tais requisitos da seguinte forma:

Notificações de execução de tarefas. Através do mecanismo de *lock*, todo usuário que deseja executar uma tarefa de forma desconectada deve trancar a tarefa. Nesse momento os demais usuários são notificados e a tarefa é removida de suas *worklists*.

Aplicações. No momento da desconexão (ou previamente se o *prefetching* for utilizado) o WM copia a aplicação usada pela tarefa para a máquina do usuário. Isso é garantido pelo protocolo de desconexão.

Dados. No momento da desconexão (ou previamente se o *prefetching* for utilizado) o WM copia os dados usados pela tarefa para a máquina do usuário, conforme descrito no protocolo de desconexão. Quando o usuário reconecta e a tarefa já foi executada, o WM copia os dados gerados para o SGWF, como mostra o protocolo de reconexão.

Gerenciamento de prazos de tarefas. Como descrito na Seção 3.4.3, quando em modo desconectado, o WM efetua o controle sobre os prazos das tarefas presentes na *worklist* de seu respectivo usuário.

Capítulo 4

Protótipo do Sistema

Este capítulo descreve as principais características do protótipo do WorkToDo desenvolvido. O protótipo foi implementado em Java – JDK 1.1.8 e para fornecer suporte à distribuição foi utilizado o OrbixWeb 3.1c.

O capítulo contém também um exemplo de processo e os resultados obtidos com os testes do protótipo.

4.1 Java e CORBA

O WorkToDo pode ser implementado utilizando qualquer linguagem de programação e qualquer infra-estrutura que forneça mecanismos para comunicação remota. Na implementação do protótipo, escolhemos utilizar Java e CORBA.

CORBA foi escolhido por apresentar um extenso conjunto de funcionalidades que fornecem transparência de acesso e facilitam a implementação de aplicações distribuídas, como visto na Seção 2.2. A implementação de CORBA utilizada foi o OrbixWeb 3.1c [16], um produto comercial da IONA, descrito na Seção 2.2.1. Esta escolha se deu ao fato do OrbixWeb fornecer o mapeamento de IDL para Java, além de estar disponível no Instituto de Computação da Unicamp. Quanto às implementações gratuitas, como por exemplo a da SUN que acompanha o próprio Java, na época em que o projeto foi iniciado elas se limitavam a implementar apenas as funcionalidades básicas do ORB, desconsiderando aspectos como persistência de objetos e *multithreading* de servidores. Atualmente a implementação da SUN já implementa tais aspectos, tornando-se assim bastante interessante de ser utilizada.

A escolha de Java foi motivada devido à sua característica de ser multiplataforma, facilitando consideravelmente a execução do sistema em diferentes plataformas de hardware e sistemas operacionais. Este é um aspecto muito importante ao considerarmos um sistema de *workflow*, no qual participam muitos usuários que utilizam o sistema a partir dos

mais variados computadores. A versão utilizada na implementação foi a JDK 1.1.8 [20]. Não foi possível utilizar uma versão mais recente de java, como a 1.2 ou 1.3, devido ao fato de que o mapeamento de IDL para Java oferecido pelo OrbixWeb gerava código na versão 1.1.8 e, além disso, todas as classes do Orbix também estão implementadas nesta versão de Java. Ao tentar executar o sistema compilado em uma versão posterior de Java, o Orbix gerava erros de execução.

4.2 Simplificações

Devido ao tempo limitado, não foi possível implementar todas as características e funcionalidades propostas no WorkToDo. O protótipo concentrou-se nos aspectos relativos à operação dos usuários, objetivando fornecer suporte a usuários móveis, sem contudo dar muita ênfase a outros aspectos também importantes ao considerarmos um SGWF como um todo. Entre esses aspectos podemos citar: recuperação de falhas de componentes (exceto falhas de conexão dos usuários, que foram tratadas), execução de tarefas automáticas, ferramentas para definição de processos, políticas de escalonamento de tarefas e de notificação de usuários, interoperabilidade com outros SGWFs e ferramentas para administração e supervisão. A implementação de tais aspectos reduziu-se apenas ao necessário para o funcionamento adequado do sistema e para a interação com os usuários.

Abaixo são descritas as simplificações efetuadas, juntamente com os aspectos implementados:

Gerenciador de Usuários e Papéis, de Definições e de Instâncias. Foram implementados como servidores com múltiplas *threads*, permitindo assim que diversos repositórios sejam acessados simultaneamente.

Ao considerarmos um grande número de processos em execução, esses componentes poderiam tornar-se gargalos de performance do sistema. Para evitar isso, uma alternativa de implementação é replicá-los, eliminando assim a sobrecarga a um nó da rede. Essa abordagem traz a vantagem adicional de tornar o sistema mais tolerante a falhas, pois se um componente falhar haverão outros que disponibilizarão os mesmos serviços. Contudo, no protótipo optou-se pela não replicação devido às limitações de tempo.

Gerenciador de Processo. O PM teve a maioria das funcionalidades implementadas, já que elas são necessárias à execução de instâncias e portanto ao funcionamento do sistema.

O Escalonador foi completamente implementado. Quando uma tarefa muda de estado, o Escalonador reavalia as árvores de dependência de todas as tarefas presentes

na lista de dependentes da tarefa. Se alguma dessas árvores é avaliada para `true`, o Escalonador insere a tarefa correspondente em uma fila de tarefas prontas para executar. Essa fila é periodicamente consultada pelo Despachante, que então despacha a tarefa para ser executada.

No Despachante, a única política de notificação de usuários implementada foi a de "todos os usuários do papel". Além disso, o Despachante não leva em consideração a prioridade das tarefas, despachando-as na ordem em que ficam prontas para executar.

Não foi implementada a recuperação de falhas dos Gerenciadores de Tarefa, considerando-se que estes componentes sempre informam o resultado da execução da tarefa correspondente. Tampouco foi tratada a recuperação de falhas dos PMs. Esses aspectos são considerados em outro trabalho [Ref. Hudo].

Constantemente um PM precisa se comunicar com os usuários do sistema – para enviar notificações de tarefas prontas para executar, remover tarefas que tenham sido selecionadas por outros usuários, e assim por diante. Para evitar consultar o URM sempre que precisar notificar algum usuário (é o URM quem guarda a referência ao WM do usuário), cada PM mantém uma lista com os usuários ativos de cada papel (como uma *cache*). Essa lista é inicializada no momento da criação do PM e atualizada periodicamente junto ao URM.

Gerenciador de Tarefa. Quando a execução de uma tarefa é iniciada, o TM automaticamente invoca a aplicação correspondente, aguardando até que a aplicação seja concluída para informar o resultado ao PM correspondente. O mecanismo que verifica o número de *retries* das tarefas e as executa novamente caso falhem foi implementado como um contador, de forma que enquanto esse contador não chega a zero a tarefa é executada novamente.

Se a aplicação correspondente à tarefa pode ser encontrada em mais de um *host*, a escolha dos *hosts* pelo TM se dá conforme a ordem especificada na definição da tarefa. O TM não utiliza, portanto, nenhum mecanismo de balanceamento de carga.

Gerenciador de Lista de Trabalho. O Prefetcher foi implementado como uma *thread* do Gerenciador de Lista de Trabalho, criada quando o usuário se conecta ao SGWF e finalizada quando o usuário se desconecta do sistema. Se, enquanto o Prefetcher está realizando o *prefetching* de um *workitem*, o usuário altera a ordenação dos *workitems* na *worklist*, essa alteração somente é considerada após a conclusão da transferência do dado que atualmente está sendo copiado. Isto significa que enquanto algum dado está sendo transferido nenhuma nova ordenação dos *workitems* é levada em conta.

Da mesma forma, se o *prefetching* é desabilitado durante a transferência de algum dado, ele será realmente desabilitado somente após o término dessa transferência.

O Gerenciador de Recursos foi implementado para o sistema operacional Linux, utilizando operações pré-definidas desse SO, como por exemplo `df` (para verificar o espaço livre em disco) e `ls` (para verificar a existência de um arquivo).

Interpretador da Linguagem de Definição de Processo. O PDLI parte de uma definição de processo e, para cada tarefa instanciada no processo, interpreta a definição da tarefa, bem como a definição da aplicação relacionada à tarefa (se houver alguma). Durante a interpretação de cada tarefa o PDLI constrói uma *task definition* e, ao final, todas as *task definitions* são reunidas e formam a *tasklist* da instância.

O PLDI possui duas suposições a respeito das definições de processo. Primeiro, que existe pelo menos uma tarefa que não possui dependências, caso contrário o processo não pode ser iniciado. Segundo, que não existem dependências mútuas, ou seja, uma tarefa t_1 depender de outra tarefa t_2 e ao mesmo tempo t_2 depender de t_1 (ou, de forma similar, t_1 depender de t_2, t_2 de t_3 e t_3 de t_1). No primeiro caso, o processo será considerado concluído antes que alguma tarefa seja executada; no segundo, as tarefas que compõem a dependência mútua não serão executadas e o processo será considerado concluído.

Prazo das tarefas. Apesar da PDL permitir que o prazo das tarefas seja expresso em horas ou dias, apenas a opção de horas foi implementada. Isso significa que o valor do prazo é sempre considerado em horas, mesmo que na definição ele esteja especificado em dias.

Tipos de Dados. Foi implementado apenas o tipo de dado Arquivo, os demais são considerados inválidos no momento em que o PDLI interpreta a definição do processo.

Atividades. As atividades podem ser somente tarefas, sem considerar a existência de sub-processos.

4.3 Considerações de Implementação

Os componentes da arquitetura foram implementados como servidores CORBA, cuja interface está descrita no Apêndice B. Os componentes se comunicam entre si através de uma rede local – a única exceção é o Gerenciador de Lista de Trabalho, que pode utilizar um *link* sem fio.

O protótipo possui uma interface gráfica baseada em menus, através da qual o usuário interage com seu Gerenciador de Lista de Trabalho e com o SGWF, visualizando, selecionando, trancando e executando *workitems*, consultando estados de processos em execução, criando novas instâncias e assim por diante. Há um tipo especial de interface, destinado aos usuários do papel Administrador, que permite a execução das operações de Administrador descritas na Seção 3.4.1.

	DM			IM		
	GA	Escritas	Total	GA	Escritas	Total
Com comentários	1103	495	1598	775	472	1247
Sem comentários	766	246	1012	546	251	797
Número de classes	9	6	15	9	3	12

	URM			PM		
	GA	Escritas	Total	GA	Escritas	Total
Com comentários	1251	836	2087	1121	2931	4052
Sem comentários	869	414	1283	794	1733	2527
Número de classes	9	7	16	18	16	34

	TM			WM		
	GA	Escritas	Total	GA	Escritas	Total
Com comentários	804	305	1109	1306	1747	3053
Sem comentários	579	217	796	912	991	1903
Número de classes	17	5	22	17	9	26

	INTERFACE			PDLI		
	GA	Escritas	Total	GA	Escritas	Total
Com comentários	-	1832	1832	-	1632	1632
Sem comentários	-	1358	1358	-	1252	1252
Número de classes	-	7	7	-	14	14

	UTIL			TOTAL		
	GA	Escritas	Total	GA	Escritas	Total
Com comentários	-	1684	1684	6360	11934	18294
Sem comentários	-	1091	1091	4466	7553	12019
Número de classes	-	12	12	79	79	158

Tabela 4.1: Linhas de código do protótipo.

A Tabela 4.1 mostra, para cada componente da arquitetura, a quantidade de linhas de código Java do protótipo, juntamente com o número de classes. A tabela contém também

informações a respeito do interpretador da Linguagem de Definição de Processo (PDLI), da interface gráfica do sistema e de outras classes utilitárias compartilhadas por todos os componentes (declarações de constantes, nomes de servidores e assim por diante). A coluna GA indica as linhas e classes geradas automaticamente pelo compilador IDL do OrbixWeb.

4.3.1 Nomes de Instâncias de Processos

O nome de uma instância de processo é composto por dois termos separados por um sublinhado (_): o primeiro termo é o nome do tipo do processo e o segundo é um sufixo com três dígitos. Este sufixo é o menor número possível que não é sufixo de alguma outra instância daquele tipo. Por exemplo: `Manutencao_001` e `Workflow_Compras_012`. Nesses exemplos, quando a instância `Manutencao_001` foi criada, todas as instâncias daquele tipo (se havia alguma) possuíam sufixos maiores que `001`. Já no momento da criação da instância `Workflow_Compras_012`, haviam pelo menos onze instâncias de `Workflow_Compras` em execução, com sufixos de `001` a `011` (poderiam haver outras, mas estas teriam sufixos maiores que `012`).

A atribuição de nomes desta forma garante que, em um dado momento, não existirão duas instâncias com o mesmo nome, sendo portanto suficiente para identificar unicamente as instâncias no sistema. Quando a execução de uma instância é completada, seu nome é reutilizado por novas instâncias daquele tipo.

4.3.2 Nomes dos Servidores

O protótipo utiliza o serviço oferecido pelo OrbixWeb de localização de objetos servidores por nome, através da operação `bind()`. Assim, cada servidor deve possuir um nome (*marker*) único, o qual é usado como parâmetro para o `bind()`. Os nomes utilizados foram os seguintes:

Gerenciadores de Processo. Como os nomes das instâncias são únicos, o *marker* de um PM é o nome da instância por ele gerenciada.

Gerenciadores de Tarefa. O *marker* de um TM é formado pelo nome da instância à qual a tarefa pertence e pelo nome da tarefa, separados por um sublinhado. Por exemplo, o *marker* do TM que gerencia a execução da tarefa `t2` da intância `wf1_005` é `wf1_005_t2`.

Gerenciadores de Lista de Trabalho. Cada WM é associado a um e somente um usuário. Como o nome de um usuário é único no sistema, o *marker* de um WM é o nome de seu respectivo usuário. Por exemplo, `Paulo`, `Maria` e assim por diante.

Gerenciador de Usuários e Papéis, de Definições e de Instâncias. Como são servidores únicos, utilizam *markers* fixos: `urm`, `dm` e `im`, respectivamente.

Como o *orbixd* é executado em cada *host* que abriga um servidor, é necessário também informar no `bind()` o *host* no qual se deseja invocar o servidor. Assim, a referência completa a um servidor WorkToDo tem o seguinte formato:

```
<nome-do-host>:<tipo-do-servidor>:<marker>
```

Por exemplo, `araguaia:processManager:wf1_005`.

4.3.3 Serialização de Objetos

Quando alguma operação remota de um servidor precisa retornar como resultado um objeto de um tipo que não é pré-definido de CORBA (os pré-definidos são `string`, `int` e demais tipos básicos), geralmente a solução adotada é definir esse tipo na IDL, de tal forma que os objetos desses tipos se tornam objetos CORBA e podem ser utilizados em retornos de operações invocadas remotamente. O cliente invoca a operação e recebe como retorno uma referência a esse objeto remoto, de forma que futuras invocações a métodos desse objeto serão na verdade repassadas (através do *stub*, ORB e *skeleton*) para o objeto remoto.

Entretanto, não foi esta a solução adotada no protótipo. Ao invés disso foi utilizada a serialização[1], um recurso existente em Java que permite converter um objeto qualquer em uma *string*. Mais tarde essa *string* pode ser novamente convertida para a classe original, criando um novo objeto com o mesmo estado do objeto que foi serializado. Dessa forma, quando uma operação remota precisa retornar um objeto que não é pré-definido de CORBA, esse objeto é serializado e a *string* é retornada pela operação. No cliente, a *string* é convertida e o objeto criado é utilizado normalmente.

Essa abordagem traz a vantagem de que as invocações aos métodos do objeto retornado são efetuadas localmente, sem necessidade de passar pelo ORB, tornando as invocações mais rápidas e menos sujeitas a falhas. Entretanto, ela possui duas desvantagens: as alterações efetuadas no objeto remoto após sua serialização não serão percebidas pelo cliente; e as alterações efetuadas pelo cliente não são efetivadas diretamente no objeto remoto, podendo gerar conflitos com outras escritas desse objeto. Essa abordagem é útil, portanto, somente quando o objeto remoto será utilizado para uma consulta breve, como foram os casos do protótipo.

[1] Do inglês *serialization*.

4.3.4 OrbixWeb

O protótipo utilizou alguns dos serviços oferecidos pelo OrbixWeb, conforme descrito na Seção 2.2.1:

Servidores com múltiplas *threads*. Foram usados para minimizar o tempo de resposta dos servidores, visto que um servidor com múltiplas *threads* trata uma requisição imediatamente. Esta medida diminui o tempo que um cliente precisa esperar pela resposta de um servidor.

Tempo de ativação dos servidores. Este serviço foi utilizado no protótipo para proporcionar economia de memória: se um servidor não recebe nenhuma requisição durante um certo período de tempo, ele é desativado, liberando memória. O único aspecto negativo dessa abordagem é que há um processamento extra se, ao receber uma requisição, o servidor está desativado; mas esse tempo mostrou-se pouco significativo. O tempo de ativação estabelecido para os servidores nos testes do protótipo foi de 10 minutos.

Persistência de servidores. Foram usados em conjunto com o tempo de ativação para salvar o estado dos servidores periodicamente. Quando o servidor era desativado, algumas das informações de seu estado eram salvas de forma persistente; na reativação essas informações eram automaticamente recuperadas.

Durante os testes, ocorriam erros freqüentes nas invocações das operações `bind()` usadas para localizar os servidores. O *orbixd* apresentava mensagens de erro como a seguinte:

```
org.omg.CORBA.COMM_FAILURE: iguacu.ic.unicamp.br/2025
```

Através de testes, descobriu-se que isso ocorria devido a uma sobrecarga no *host* onde se tentava efetuar o `bind()` (uso excessivo de memória da máquina, ou processador muito ocupado). Como a máquina estava temporariamente sobrecarregada, o Orbix não era capaz de instanciar um servidor (ou, na prática, criar um novo processo Java). Se o mesmo `bind()` era executado novamente, geralmente se obtinha sucesso após algumas tentativas.

Por esse motivo, todas as chamadas à operação `bind()` foram substituídas pelo seguinte trecho de código (usando, é claro, o nome do servidor desejado):

```
int retries = Servers.RETRIES;
String reason = "no reason";
while (retries > 0) {
    try {
```

```
            return DefinitionManagerHelper.bind (Servers.DEFINITION_MANAGER,
                        Servers.DEFINITION_MANAGER_HOST);
        }
        catch (org.omg.CORBA.SystemException exc) {
            reason = exc.toString ();
            retries--;
        }
    }

    System.out.println ("Can't connect to Definition Manager:  "+ reason);
    return null;
```

Assim, são realizadas até RETRIES tentativas (valor este definido na classe Servers) antes de ser gerado um erro na ativação dos servidores. O valor utilizado para RETRIES nos testes do protótipo foi de 5 tentativas, mas observou-se que normalmente a segunda ou terceira tentativa já eram bem sucedidas.

4.3.5 Repositórios

Todos os repositórios foram implementados através de uma tabela (um objeto da classe *Hashtable* de Java), que guarda um elemento em cada posição. Para tornar os repositórios persistentes, foi utilizado novamente o recurso de serialização de objetos de Java.

A tabela é serializada e depois gravada em um arquivo com o nome do repositório seguido da extensão `.rep` (por exemplo: `ProcessosCompletados.rep`). Para recuperar o repositório, a tabela é reconstruída a partir do conteúdo lido do arquivo correspondente.

4.4 Exemplo

Para demonstrar a utilização do WorkToDo, vamos considerar o exemplo descrito anteriormente, onde uma empresa que presta serviços de manutenção cria um processo para gerenciar seus serviços.

A Figura 4.1 mostra o diagrama que representa o processo. Os retângulos são as tarefas e os nomes abaixo deles são as entidades processadoras. As dependências entre tarefas são representadas por setas nomeadas SUCCEEDED ou FAILED.

O processo é especificado na PDL do WorkToDo da seguinte forma:

```
WORKFLOW Manutencao {
    FILE OrdemServico {
```

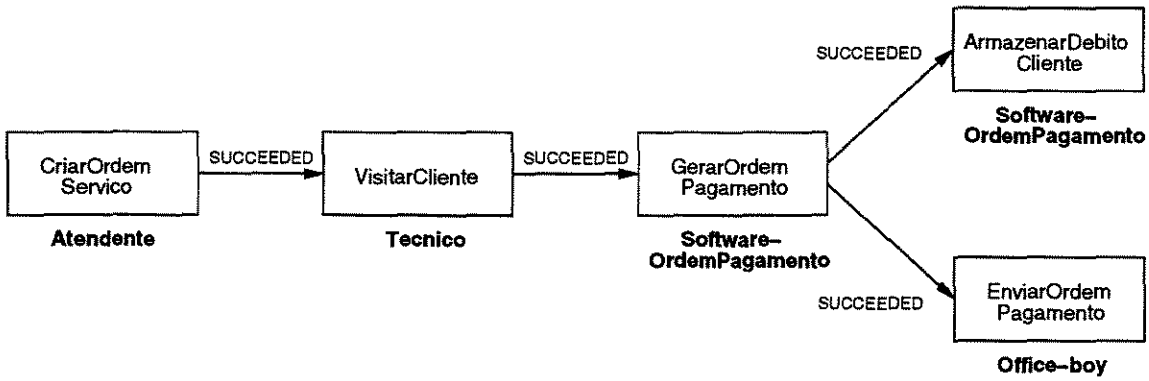


Figura 4.1: Diagrama do processo do exemplo.

```
    NAME "/processos/arquivos/OrdemServico.txt";
}

FILE OrdemPagamento {
    NAME "/processos/arquivos/OrdemPagamento.txt";
}

FILE DebitosClientes {
    NAME "/processos/arquivos/Debitos.txt";
}

TASK CriarOrdemServico: PreencherFormulario {
    ROLE Atendente;
    IN_CONTEXT OrdemServico;
    OUT_CONTEXT OrdemServico;
    DEPENDS;
    DESCRIPTION "Preencher os itens 'Nome do Cliente' e 'Endereço' da
                 Ordem de Servico";
}

TASK VisitarCliente: PreencherFormulario {
    ROLE Tecnico;
    IN_CONTEXT OrdemServico;
    OUT_CONTEXT OrdemServico;
    DEPENDS CriarOrdemServico -> SUCCEEDED;
    DESCRIPTION "Preencher os itens 'Servicos' e 'Materiais' da Ordem
                 de Servico";
}
```

```
    TASK GerarOrdemPagamento:  GerarOrdemPagamento {
        IN_CONTEXT OrdemServico;
        OUT_CONTEXT OrdemPagamento;
        DEPENDS VisitarCliente -> SUCCEEDED;
        DESCRIPTION "Gera automaticamente a Ordem de Pagamento do servico."
    }

    TASK ArmazenarDebitoCliente:  GravarArquivo {
        IN_CONTEXT OrdemPagamento;
        OUT_CONTEXT DebitosClientes;
        DEPENDS GerarOrdemPagamento -> SUCCEEDED;
        DESCRIPTION "Armazena o debito do cliente, de acordo com o servico
                     executado e os materiais utilizados."
    }

    TASK EnviarOrdemPagamento:  EnviarDocumento {
        ROLE Office_boy;
        IN_CONTEXT OrdemPagamento;
        DEPENDS GerarOrdemPagamento -> SUCCEEDED;
        DESCRIPTION "Colocar a Ordem de Pagamento num envelope, endereca-lo
                     ao cliente indicado e envia-lo";
    }
}
```

O processo `Manutencao` contém cinco tarefas, identificadas por `CriarOrdemServico`, `VisitarCliente`, `GerarOrdemPagamento`, `ArmazenarDebitoCliente` e `EnviarOrdemPagamento`. Cada uma dessas tarefas instancia um modelo de tarefa (`PreencherFormulario`, `GravarArquivo` e assim por diante).

A tarefa `VisitarCliente` é instância do modelo de tarefa `PreencherFormulario` e a responsabilidade por sua execução cabe aos usuários do papel `Tecnico`. Ela somente será iniciada se a tarefa `CriarOrdemServico` for executada com sucesso. O único dado de entrada e saída de `VisitarCliente` é o arquivo `OrdemServico.txt`, identificado por `OrdemServico` e localizado no diretório `/processos/arquivos`. Cada tarefa contém ainda uma descrição de como proceder para executá-la.

O modelo de tarefa `PreencherFormulario` é definido da seguinte forma:

```
TASK PreencherFormulario {
    TYPE Semi-automatic;
    APPLICATION EditorTexto;
```

```
    PRIORITY 10;
    DEADLINE 48 HOURS;
    DISCONNECTED_OPERATION true;
}
```

A tarefa é do tipo semi-automática e a aplicação relacionada a ela é `EditorTexto`, cujas características estão definidas separadamente. São ainda indicados a prioridade e o prazo para execução da tarefa. A cláusula `DISCONNECTED_OPERATION` indica se a tarefa pode ser executada em modo desconectado (`true`) ou não (`false`).

Já o modelo de tarefa `GerarOrdemPagamento` é definido como segue:

```
TASK GerarOrdemPagamento {
    TYPE Automatic;
    APPLICATION Software-OrdemPagamento;
    PRIORITY 10;
    DEADLINE 24 HOURS;
    DISCONNECTED_OPERATION false;
    RETRIES 5;
}
```

A tarefa é do tipo automática, sendo executada pela aplicação `Software-OrdemPagamento`. Sua prioridade é 10 e seu prazo máximo para execução é 24 horas. A tarefa não pode ser executada em modo desconectado (`DISCONNECTED_OPERATION = false`) e o TM pode re-executá-la até 5 vezes em caso de falha.

As definições das aplicações `EditorTexto` e `Software-OrdemPagamento` são as seguintes:

```
APPLICATION EditorTexto {
    NAME "/n/gnu/emacs/bin/emacs";
    SIZE 3125;
    OS "Linux";
    CPU "486";
    INSTALLER;
    HOSTS "iguacu.ic.unicamp.br", "tuiuiu.ic.unicamp.br";
}
```

O nome do arquivo executável da aplicação, localizado no diretório `/n/gnu/emacs/bin` é `emacs` e seu tamanho é `3125` KB. A aplicação executa sobre o sistema operacional `Linux`

e o processador mínimo para que possa ser executada é um 486. Não foi especificado nenhum instalador (cláusula INSTALLER), então o único arquivo necessário para executar a aplicação é seu próprio executável. A aplicação está disponível em dois *hosts*: {iguacu, tuiuiu}.ic.unicamp.br.

```
APPLICATION Software-OrdemPagamento {
    NAME "/tmp/soft/sop";
    SIZE 250;
    OS "Linux";
    CPU "486";
    INSTALLER;
    HOSTS "araguaia.ic.unicamp.br";
}
```

O nome do arquivo executável da aplicação é sop e seu tamanho é 250 Kb. Novamente, como não foi especificado nenhum instalador o único arquivo necessário para executar a aplicação é seu próprio executável. Esta aplicação está disponível somente no *host*: araguaia.ic.unicamp.br.

Abaixo segue a explicação passo a passo de uma possível execução de uma instância desse processo:

1. Um usuário do papel criador do tipo de processo Manutencao cria uma instância do processo, ocasionando a criação de um Gerenciador de Processo para gerenciar a execução da instância;
2. A tarefa CriarOrdemServiço não possui dependências, assim logo que a instância é iniciada o Escalonador a marca como pronta para executar. Como a tarefa é semi-automática (o modelo de tarefa PreencherFormulario define essa tarefa como sendo desse tipo), o Despachante notifica os Gerenciadores de Lista de Trabalho de todos os usuários do papel Atendente (ou de alguns desses usuários, conforme a política de notificação escolhida);
3. Os WMs notificados adicionam um *workitem* correspondente à tarefa nas *worklists* de seus respectivos usuários;
4. Um dos usuários do papel Atendente seleciona o *workitem* e o executa, informando em seguida o resultado da execução (SUCCEEDED);
5. Quando a tarefa CriarOrdemServico alcança o estado SUCCEEDED, o Escalonador detecta que as dependências da tarefa VisitarCliente foram satisfeitas, e portanto ela já pode ser executada. O Despachante notifica os WMs dos usuários do papel Tecnico, os quais adicionam um *workitem* para a tarefa nas *worklists* dos usuários;

6. Um dos usuários do papel `Tecnico` tranca o *workitem* e se desconecta do sistema. Ele executa o *workitem* e depois se reconecta. Nesse momento o WM repassa o resultado da execução (`SUCCEEDED`) para o PM;
7. Quando a tarefa `VisitarCliente` alcança o estado `SUCCEEDED`, o Escalonador detecta que as dependências da tarefa `GerarOrdemPagamento` foram satisfeitas, e portanto ela já pode ser executada.
8. Como `GerarOrdemPagamento` é uma tarefa automática, o Despachante cria um Gerenciador de Tarefa em um dos *hosts* indicados na definição da aplicação correspondente à tarefa (no caso, `araguaia.ic.unicamp.br`, como indicado na definição da aplicação `Software-OrdemPagamento`);
9. O TM inicia a execução da tarefa invocando a aplicação `Software-OrdemPagamento`, e aguarda que ela seja concluída. Quando isso ocorre, o TM informa o PM do resultado da execução da tarefa (`SUCCEEDED`);
10. A partir do resultado da execução de `GerarOrdemPagamento`, o sistema pode executar as tarefas `ArmazenarDebitoCliente` e `EnviarOrdemPagamento`. Após a execução dessas tarefas, o Escalonador detecta que o processo foi completado;
11. O PM notifica o IM, que armazena o estado final da instância no repositório de processos completados. Após essa operação o PM é desativado;
12. O sistema envia uma notificação para o usuário responsável pela instância, informando que o processo foi concluído.

A Figura 4.2 mostra uma possível *worklist* de um usuário do papel `Tecnico`. A *worklist* pertence ao usuário `Paulo`, que no momento está conectado ao SGWF, e contém cinco *workitems*, sendo três deles correspondentes a tarefas `VisitarCliente` de diferentes instâncias do processo `Manutencao` e os outros dois correspondentes a tarefas `task_1` pertencentes a instâncias de `wf2`. O primeiro *workitem* está pronto para ser executado (estado `READY`), o segundo e o terceiro estão selecionados (`SELECTED`) e o quarto está trancado (`LOCKED`). Tarefas de mesmo nome são diferenciadas entre si pelo nome da instância à qual pertencem.

A *worklist* contém diversas informações a respeito dos *workitems*, como o nome da instância de processo e da tarefa correspondentes, o tipo do *workitem*, seu estado atual e se pode ser executado em modo desconectado. A partir dos menus o usuário pode interagir com a *worklist*, com os *workitems* e com o SGWF como um todo.

A Figura 4.3 mostra a lista de instâncias em execução em um determinado momento, contendo informações como: o nome da instância, o tipo de processo ao qual ela pertence, além do nome e endereço IP do *host* onde está sendo executado o PM que a gerencia.

A Figura 4.4 mostra em detalhes o estado de uma instância em particular (`Manutencao_001`). A instância foi criada em `21/03/2002` às `10:57` e seu usuário responsável é

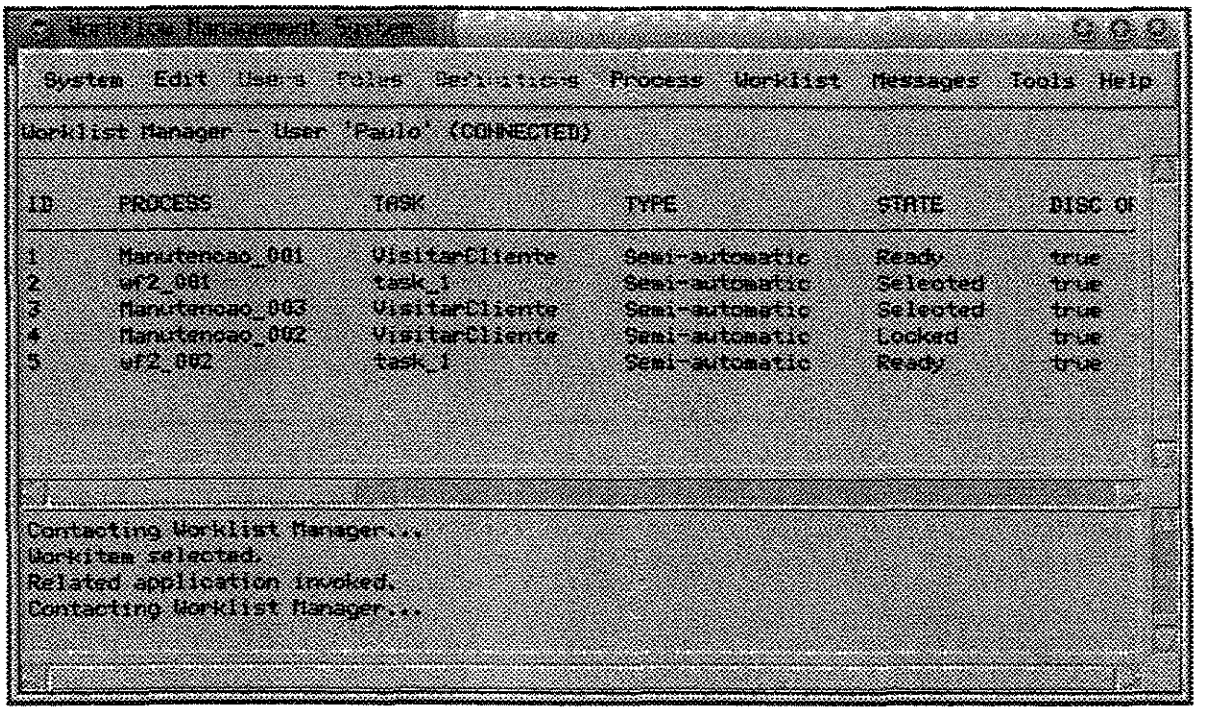


Figura 4.2: Exemplo de Worklist.

Maria. São fornecidas também informações detalhadas a respeito de cada uma das tarefas do processo, como tipo, estado atual e prazo para execução, entre outras.

A Figura 4.5 ilustra o recebimento de mensagens por um usuário. As duas primeiras mensagens se referem a prazos para execução de duas tarefas do processo `Manutencao_003`: `AtualizarDebitoCLiente` e `GerarOrdemPagamento`; a terceira alerta o usuário a respeito da falha na execução da tarefa `CriarOrdemServico` do processo `Manutencao_002`; e a última informa o usuário de que o processo `Manutencao_002` completou sua execução. Para cada mensagem é indicado também o horário de recebimento.

4.5 Resultados

As máquinas que compõem o sistema distribuído utilizado nos testes do protótipo são estações de trabalho com sistemas operacionais Linux/Windows NT e máquinas Sparc com sistema operacional Solaris, conectados através de uma rede local de 10 Mb.

Para efetuar os testes, o URM, DM e IM foram executados cada um em um *host* diferente. O *host* de cada PM era escolhido no momento da criação da instância, pelo usuário que a criava. Os *hosts* dos TMs eram escolhidos de acordo com as definições de suas respectivas aplicações, já que o TM deve ser executado no *host* onde a aplicação

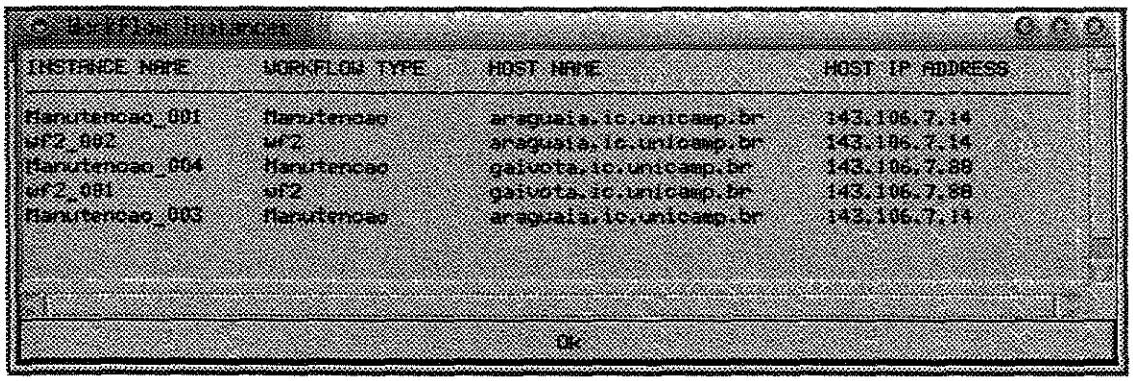


Figura 4.3: Exemplo de listagem de instâncias em execução.

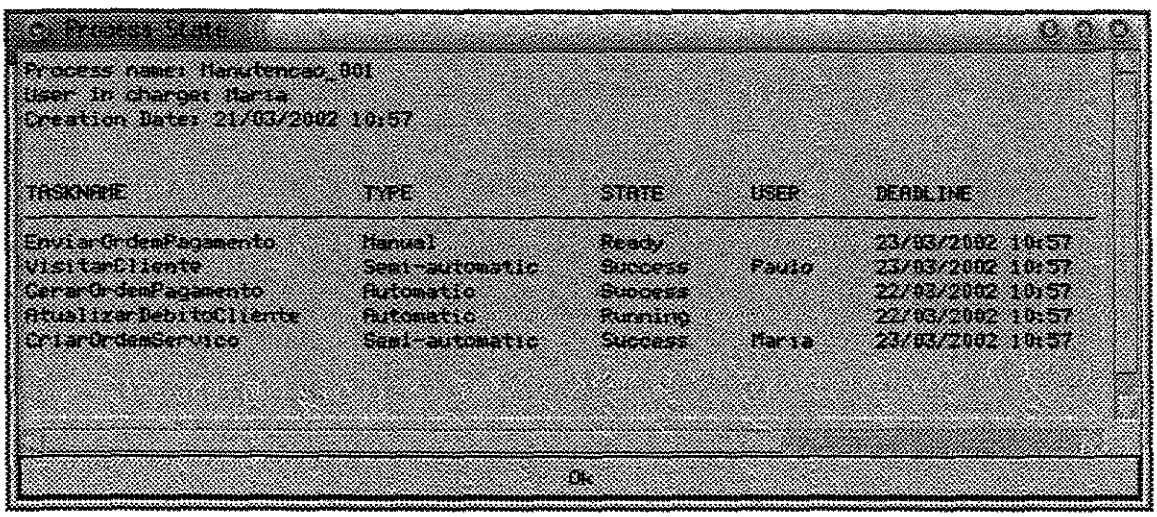


Figura 4.4: Exemplo de consulta ao estado de uma instância de Processo.

está disponível para execução. Já os *hosts* dos WMs eram os mesmos de seus respectivos usuários. A Tabela 4.2 mostra os *hosts* escolhidos.

Em termos de ambiente sem fio, nosso interesse nos testes se concentrou em dois aspectos: baixa largura de banda e desconexões freqüentes. Não havia uma rede sem fio disponível para uso no Instituto de Computação, por isso esses dois aspectos foram testados através de simulação. Para simular variações na largura de banda, o Gerenciador de Recursos estabelece uma largura de banda aleatória quando é consultado; para simular desconexões, o `orbixd` é finalizado na máquina do usuário, impedindo a comunicação entre o WM e os demais componentes. Assim foi possível avaliar o comportamento do sistema frente a variações nesses aspectos.

As operações desconectada e semi-conectada mostraram-se possíveis de serem execu-

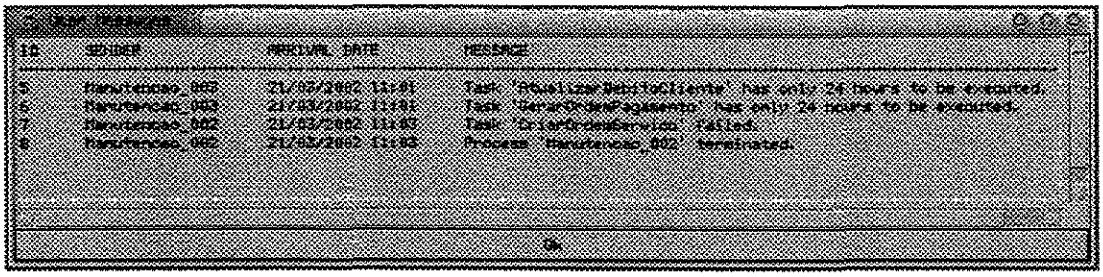


Figura 4.5: Exemplo da lista de mensagens de um usuário.

Componente	*Host*
URM	`iguacu.ic.unicamp.br`
DM	`araguaia.ic.unicamp.br`
IM	`tuiuiu.ic.unicamp.br`
PM	Escolhido pelo usuário no momento da criação da instância.
TM	Escolhido de acordo com a definição da aplicação correspondente à tarefa.
WM	Mesmo do usuário correspondente.

Tabela 4.2: *Hosts* utilizados nos testes do protótipo.

tadas e o sistema se mostrou estável frente a quedas de conexão, sem gerar nenhum tipo de inconsistência.

Na ocorrência de quedas de comunicação, o usuário não pode notificar o SGWF de que está se desconectando. Assim sendo, se um PM não consegue contactar determinado WM um certo número de vezes consecutivas (3 nos testes), o PM considera que o usuário se desconectou e notifica o URM, que marca o usuário como inativo. Quando o WM desse usuário interage com algum outro componente, o URM é notificado, marcando o usuário novamente como ativo.

Com a utilização do *prefetching* o tempo de execução do protocolo de desconexão mostrou-se bastante pequeno, reduzindo-se praticamente ao tempo da chamada remota ao URM. Isto porque a transferência dos *workitems* trancados, que é o item do protocolo que consome mais tempo, já foi executado.

4.5.1 Avaliação de CORBA

CORBA se enquadrou bem dentro da proposta de um sistema de *workflow* distribuído. A transparência fornecida é claramente um grande benefício, separando os aspectos de distribuição dos aspectos funcionais do sistema.

Entretanto, a memória necessária para executar um objeto servidor é relativamente

grande. Além disso, CORBA exige uma considerável capacidade de processamento e armazenamento. Esses aspectos são bastante limitados em dispositivos portáteis, tais como PDAs, limitando o uso de CORBA nesse tipo de dispositivo.

Esses problemas poderiam ser evitados, talvez, utilizando *minimumCORBA*[2], já que no protótipo foram utilizados apenas os serviços estáticos de CORBA. Outra possibilidade seria utilizar algum outro mecanismo de comunicação remota menos custoso computacionalmente, como por exemplo Sockets.

Para alcançar a solução mais adequada, é necessário uma avaliação dos recursos utilizados por cada uma dessas tecnologias, elaborando um estudo capaz de determinar sua aplicabilidade na prática.

4.5.2 IORs de CORBA

Existe um problema com relação às IORs[3] de CORBA para alguns objetos do WorkToDo – mais especificamente os WMs, que são executados em *hosts* móveis. Quando um *host* móvel se move para uma célula diferente, ele recebe um novo endereço IP na nova rede, invalidando todas as IORs para os objetos residentes no *host* móvel.

Uma maneira de resolver este problema é armazenar o estado do WM no URM antes de se mover de uma célula para outra, recuperando esse estado após instanciar um novo WM. Nesse momento o WM também informa ao URM seu novo endereço IP.

Para essa solução ser possível, entretanto, o WM deve ser capaz de detectar (ou então deve ser notificado) quando seu *host* está se movendo para uma nova célula.

[2] *minimumCORBA* é um subconjunto de CORBA cujo objetivo é o desenvolvimento de aplicações para dispositivos com recursos de computação limitados. *minimumCORBA* não inclui os aspectos dinâmicos de CORBA, como facilidades para criação, localização e ativação dinâmicas de objetos e operações.

[3] *Interoperable Object References*: são as referências a objetos remotos, utilizadas pelo CORBA para invocar operações de objetos na rede. A IOR contém, entre outras informações, o identificador do objeto e seu endereço IP (no caso da utilização de uma rede baseada no protocolo TCP/IP).

Capítulo 5

Trabalhos Relacionados

Existem inúmeros trabalhos de pesquisa em torno de sistemas de *workflow*, tais como METEOR [24, 25], WAMO [12, 13], MENTOR [36, 37], MOBILE [18] e muitos outros. Já os trabalhos que abordam a questão da utilização de SGWFs em ambientes de computação móvel não são tão numerosos, embora também existam.

Este Capítulo aponta os principais entre estes trabalhos, descrevendo suas características e funcionalidades e comparando-os com o WorkToDo.

5.1 Exotica

O projeto ***Exotica*** [1, 2, 3], desenvolvido pela IBM, tem por objetivo explorar a utilização de alguns conceitos presentes em modelos avançados de transação, em arquiteturas cliente/servidor e em computação móvel, dentro do contexto de gerenciamento de *workflows*.

O Exotica se baseia no FlowMark, um produto comercial da IBM. No FlowMark um *workflow* é representado como um grafo acíclico dirigido, onde os nós representam atividades e os arcos representam conectores de controle e conectores de dados. As ***atividades*** podem ser: simples, quando compostas apenas pela invocação de uma aplicação; ou processos, quando são compostas por um conjunto de outras atividades.

Cada atividade possui uma condição de início e uma condição de término, que especificam as condições para que a atividade seja iniciada e completada, respectivamente. A ordem de execução das atividades é especificada por meio de ***conectores de controle***, de tal forma que cada atividade possui como entrada e saída um ou mais destes conectores. Quando todos os conectores de entrada de uma atividade são avaliados, a condição de início da atividade é testada. Da mesma forma, quando uma atividade é completada sua condição de término é testada.

O fluxo de dados entre as atividades ocorre através dos ***conectores de dados*** e dos ***recipientes de dados de entrada e de saída***. Cada atividade possui um recipiente de

entrada e um recipiente de saída, e os conectores de dados mapeiam valores do recipiente de dados de saída de uma atividade para o recipiente de dados de entrada de outra atividade.

O componente responsável pelo gerenciamento da execução dos processos é o ***servidor FlowMark***. Este servidor originalmente acessa um único banco de dados centralizado, onde são armazenadas todas as informações a respeito dos processos. Isso limita a performance do sistema, pois pode haver um grande número de processos em execução simultaneamente, sobrecarregando o banco de dados. Além disso, se o banco de dados falha, a execução de todos os processos é interrompida. Para resolver esse problema o Exotica propôs duas soluções, nas quais o banco de dados centralizado é substituído por um conjunto de bancos de dados localizados em diferentes *hosts*:

- Na primeira, o banco de dados centralizado é substituído por um conjunto de bancos de dados [1] autônomos, capazes de se comunicar entre si informando a situação da execução de atividades e processos. Assim, se um determinado banco de dados falha, os demais processos em execução que utilizavam outros bancos de dados podem prosseguir normalmente;
- Na segunda, são formados diversos grupos de servidores FlowMark [2], de tal forma que todos os servidores de um determinado grupo estão conectados a um mesmo banco de dados. Assim, se um determinado banco de dados falha, os processos em execução nos outros grupos continuam sendo executados normalmente.

O Exotica aborda a questão da operação desconectada [4]. Para isso são necessárias algumas modificações nos componentes do modelo FlowMark, para que as informações a respeito das atividades a serem executadas de forma desconectada sejam armazenadas no computador do usuário e para que os resultados sejam repassados ao SGWF. A operação desconectada é tratada da seguinte forma:

- Antes de o usuário desconectar as atividades que serão executadas são trancadas (para que um outro usuário não as selecione para execução) e todas as informações necessárias para sua execução são transferidas para o computador do usuário;
- Durante a operação desconectada o usuário pode iniciar normalmente a execução das atividades, visto que todos os dados referentes a elas estão armazenados localmente. Os resultados da execução das atividades são armazenados também localmente;
- Quando o cliente se reconecta, o servidor FlowMark transfere ao usuário sua lista de trabalho, contendo inclusive aquelas atividades que foram trancadas. O cliente então atualiza a lista, indicando as atividades completadas durante o período de desconexão, e envia a lista de volta para o servidor, que dessa forma é informado da execução das atividades.

Como pode ser visto, o Exotica modifica o modelo e arquitetura originais do FlowMark para permitir a operação desconectada, adicionando funcionalidades a alguns componentes e alterando os procedimentos para conexão e reconexão dos usuários, além da execução de atividades.

Como o Exotica, o WorkToDo utiliza a idéia de trancar as tarefas que serão executadas em modo desconectado, para que o sistema tome as medidas necessárias para garantir a execução consistente de tarefas. Em termos arquiteturais, entretanto, os dois sistemas são diferentes, pois no WorkToDo existe um Gerenciador de Processo para cada instância em execução, enquanto que no Exotica todas as instâncias são gerenciadas pelo servidor FlowMark.

5.2 INCAS

No modelo ***INCAS*** [7, 8], as informações sobre a execução dos processos migram de uma estação de processamento para outra. Essas estações são autônomas, podendo ser computadores móveis que se conectam à rede por períodos breves.

A cada processo é associado um ***INformation CArrier*** (INCA), um objeto que contém um conjunto de informações a respeito do processo: dados que definem o contexto de execução; seu histórico de execução (usado para recuperação de falhas); um conjunto de regras que especificam o fluxo de controle e dados entre as diversas atividades; e informações sobre qual atividade deve ser executada.

A execução de um processo é efetuada através da execução do INCA associado a ele. A execução é iniciada com o envio de um INCA para uma estação de processamento, a fim de que a primeira atividade seja executada. Uma ***estação de processamento*** pode variar desde uma aplicação automatizada até um usuário executando uma operação manual. Cada uma dessas estações possui um conjunto de dados e de regras próprios, fornecendo também um conjunto de serviços (ou procedimentos) que são realizados por ela.

Os INCAs que chegam a uma estação de processamento requisitam um determinado serviço. A execução desse serviço se baseia nos dados e regras trazidos pelo INCA e também nos dados e regras da própria estação. A estação de processamento realiza então as seguintes operações:

- Registra no histórico do INCA qual foi o serviço requisitado;
- Executa o serviço, possivelmente modificando os dados e regras próprios da estação e também os dados e regras do INCA;
- Através das regras do INCA e de suas regras próprias, a estação determina o próximo destino do INCA (uma outra estação de processamento) e redireciona o INCA para

este local. O próximo destino pode ser também várias estações de processamento diferentes.

Dessa forma, o INCA migra de uma estação de processamento para outra, executando determinadas operações em cada uma delas e possivelmente modificando seus dados e regras de execução. Este ciclo se repete até que as regras do INCA especifiquem que sua execução foi completada com sucesso. Nesse momento o INCA (e conseqüentemente o processo) é finalizado.

O modelo INCAS utiliza uma arquitetura completamente distribuída para o gerenciamento dos processos, não existindo nenhum tipo de controle centralizado. O INCAS não exige que as estações de processamento estejam conectadas entre si através de uma rede de comunicação durante todo o tempo ou que as conexões sejam de alta velocidade e estáveis, permitindo portanto a execução em um ambiente de comunicação sem fio. A única exigência é que as estações sejam capazes de receber um INCA, executar as operações requisitadas por ele e redirecioná-lo para a próxima estação de processamento. Mas existem alguns problemas de implementação relacionados exatamente a essa última condição (por exemplo: se uma entidade X deve redirecionar um INCA para outra entidade Y, o que X deve fazer se Y não puder ser contactada por muito tempo?), para os quais ainda não foram apresentadas soluções.

Como no INCAS, o WorkToDo permite que um usuário execute tarefas conectado ao sistema por meio de uma rede sem fio. Há uma semelhança também em relação às instâncias de processo: enquanto o INCAS cria um INCA para cada instância, o WorkToDo cria um Gerenciador de Processo. A diferença, porém, é que o INCA migra de um *host* para outro, enquanto que o PM permanece fixo em um *host*, apenas criando TMs e adicionando *workitems* em diferentes *hosts*.

Em termos arquiteturais, os dois sistemas utilizam abordagens distintas: o INCAS possui uma arquitetura completamente distribuída, enquanto que a arquitetura do WorkToDo é semi-distribuída. Além disso, diferentemente do INCAS, apresentamos um protótipo que avalia os mecanismos e soluções propostos.

5.3 Mobile Worklists

Büssler [9] identifica uma série de requisitos necessários para que um SGWF permita a operação desconectada, tais como: cópia dos dados relacionados às tarefas para o computador móvel, gerenciamento local de prazos de tarefas, identificação dos dados e requisitos necessários às aplicações e reintegração de resultados. Esses requisitos são então tratados nas ***mobile worklists*** [9], de tal forma que um SGWF convencional que as incorpore torne-se capaz de permitir operação desconectada.

Uma *mobile worklist* é como uma *worklist* convencional, com a diferença de que ela somente contém *workitems* que podem ser executados de forma desconectada, sem necessidade de comunicação com o SGWF, ou seja, cujos dados e informações estão todos disponíveis localmente. Cada *workitem* pertencente à *mobile worklist* possui associado a ele um ***download container***, que contém toda a informação necessária à execução do *workitem* (dados e aplicações).

Quando o usuário informa ao SGWF de que deseja se desconectar, é construído um *download container* para cada *workitem* que pode ser executado em modo desconectado. De posse desses *containers* é construída então a *mobile worklist* e o usuário pode se desconectar. Na reconexão os resultados são repassados ao SGWF.

A idéia, portanto, é estender um SGWF convencional para que este implemente as *mobile worklists*, garantindo automaticamente o suporte à operação desconectada. Entretanto, não há referência a alguma implementação que comprove a viabilidade e eficiência desse método.

O WorkToDo incorpora diversos dos requisitos apontados por Büssler, entre eles: cópia local dos dados relacionados às tarefas (efetuada pelo WM no momento da desconexão, ou antes se o *prefetching* estiver habilitado), gerenciamento local de prazos de tarefas (efetuado pelo WM quando o usuário está desconectado) e reintegração de resultados (também efetuada pelo WM no momento da reconexão do usuário).

5.4 Sistemas de Workflow e Computação Móvel

Jing *et al.* [21] descreve diversos aspectos referentes à integração entre computação móvel e sistemas de *workflow*, visando maximizar as melhorias de performance e os benefícios provenientes dessa integração. Dentre esses aspectos destacamos os seguintes:

- Utilização de duas estratégias de atribuição de *workitems* a usuários: no momento em que o *workitem* torna-se pronto para executar, ou no momento em que o usuário se conecta ao sistema. Isso é útil porque em um ambiente de comunicação sem fio a conexão e desconexão de usuários são muito freqüentes, então é necessário notificar usuários a respeito de tarefas sempre que estes se conectam ao sistema;
- *Prefetching* de *workitems* em *background*, com o objetivo de minimizar a transferência de dados quando o usuário seleciona um *workitem*;
- Priorização de *workitems* de acordo com a disponibilidade de dados na máquina do usuário. Um *workitem* cuja aplicação já está instalada no computador móvel aparece na *worklist* com uma prioridade maior que os demais. Da mesma forma, quanto mais dados referentes ao *workitem* já foram transferidos, maior sua prioridade.

O WorkToDo utiliza as duas estratégias de atribuição de *workitems*, visto que quando uma tarefa fica pronta para executar um *workitem* correspondente é adicionado na *worklist* dos usuários ativos do papel, mas um usuário desse papel que se conecta mais tarde também recebe um *workitem* (desde que nenhum outro usuário tenha selecionado o *workitem* nesse período).

A técnica do *prefetching* também foi utilizada no WorkToDo, tirando proveito dos momentos em que os usuário está conectado ao sistema sem no entanto estar realizando transferência de dados.

Capítulo 6

Conclusão

Nesta dissertação apresentamos o WorkToDo, um SGWF para ambientes de comunicação sem fio. O sistema provê três modos distintos de operação: conectado, semi-conectado e desconectado. O WorkToDo proporciona, portanto, mobilidade aos usuários, por conseqüência aumentando a flexibilidade de uso.

A operação conectada é aquela existente nos SGWFs tradicionais, onde o usuário está conectado ao sistema por meio de uma rede tradicional de alta velocidade, restringindo portanto a operação do usuário a um único local (um escritório ou um prédio, por exemplo).

A operação semi-conectada permite que o usuário utilize o sistema a partir de seu *laptop*, PDA ou outro dispositivo portátil, sem restrições de localização. O usuário se conecta ao sistema de qualquer localidade a uma rede sem fio e independentemente da qualidade de sua conexão (largura de banda, erros de transmissão) é capaz de executar operações no sistema, praticamente como na operação conectada.

A operação desconectada permite que o usuário possa também executar tarefas sem estar conectado ao sistema, bastando para isso indicar, através de um mecanismo de trancamento, as tarefas que deseja executar em modo desconectado. O usuário tranca um conjunto de tarefas e, no momento da desconexão, o WorkToDo copia todos os dados referentes a esse conjunto para a máquina do usuário. A execução das tarefas se dá sobre os dados copiados e na reconexão os resultados da execução são repassados para o WorkToDo. Para permitir a operação desconectada, consideramos aspectos como o tratamento local de prazos de tarefas (efetuado pelo WM) e a cópia de dados de entreada e saída (efetuada pelo WM durante a execução dos protocolos de desconexão e reconexão).

O WorkToDo implementa a técnica de *prefetching* para otimizar a cópia de *workitems* e aproveitar a largura de banda da conexão. Enquanto o usuário está conectado ao SGWF, possivelmente executando tarefas, o WM transfere em *background* os *workitems* que podem ser executados de forma desconectada. Essa cópia é utilizada em dois casos: se

o usuário trancar os *workitems* eles não precisarão ser transferidos, economizando tempo; e se ocorrer uma desconexão não planejada, o usuário poderá continuar trabalhando nos *workitems* que estavam sendo executados sobre dados locais, bem como naqueles que tenham sido trancados e já estejam transferidos.

Considerando a execução de tarefas, no WorkToDo as operações conectada/semi-conectada e desconectada são praticamente idênticas. A única diferença é que, em modo desconectado, somente podem ser executados os *workitems* trancados pelo usuário antes da desconexão.

Nossa abordagem difere das existentes por apresentar um modelo e uma arquitetura que fornecem suporte não apenas à operação desconectada, mas também à operação semi-conectada em ambientes de comunicação sem fio. O WorkToDo não impõe nenhuma restrição sobre a localização dos usuários nem tampouco sobre a qualidade das conexões. Isso proporciona um significativo aumento da flexibilidade de uso, sem comprometer a execução correta dos processos e garantindo portanto a integridade do sistema.

Apresentamos também um protótipo capaz de validar e verificar as soluções propostas, revelando-as bastante satisfatórias. Embora a implementação do protótipo tenha utilizado Java e CORBA, o WorkToDo pode ser mapeado para outras linguagens de programação e plataformas de comunicação remota, sendo portanto um sistema bastante flexível.

6.1 Trabalhos Futuros

Podemos identificar alguns trabalhos futuros para o WorkToDo, descritos a seguir:

- Implementar os aspectos que não foram implementados no protótipo: recuperação de falhas dos Gerenciadores de Processo e dos Gerenciadores de Tarefas, demais políticas de notificação de usuários e prioridades de tarefas no Despachante, atividades que são sub-processos, prazos de tarefas expressos em dias, entre outros;
- Realizar uma análise de desempenho do protótipo, avaliando questões como: custo de processamento, total de memória alocada e quantidade de mensagens trocadas entre os componentes.
- Implementar um protótipo com um mapeamento para diferentes linguagens de programação e plataformas de comunicação remota, com o objetivo de realizar comparações de performance e eficiência;
- Armazenar os repositórios utilizando um banco de dados, aumentando assim sua confiabilidade e segurança;
- Desenvolver ferramentas gráficas para a definição de tipos de processos, facilitando sua especificação e a visualização de sua estrutura;

- Desenvolver ferramentas de administração mais sofisticadas que ofereçam maiores facilidades para o gerenciamento do estado global do sistema;
- Tratar problemas de sincronização de tarefas que acessam recursos compartilhados, tais como arquivos e bases de dados. Nesses casos, os dados gerados pelas tarefas não podem ser acessados ao mesmo tempo, pois haverão inconsistências. Faz-se necessário então algum controle de sincronização (ou seqüenciamento) entre essas tarefas;
- Verificar o comportamento do sistema com relação a dados do tipo *Query*, que não foram implementados no protótipo. As *queries* trazem algumas complicações a serem verificadas (por exemplo, a máquina do usuário deve ser capaz de acessar os dados gerados pela *query* – necessitando talvez de um Sistema Gerenciador de Banco de Dados);
- Permitir a seleção de tarefas interdependentes para aumentar a flexibilidade de uso. Se existe uma tarefa A sem dependências e outra tarefa B que depende de A, o WorkToDo permite que o usuário selecione somente a tarefa A. Para executar a tarefa B o usuário deve primeiro informar o resultado de A, para então B ficar disponível e poder ser selecionada. A seleção de tarefas interdependentes permite que o usuário selecione as tarefas A e B no mesmo momento e, ao informar o resultado de A, B tornar-se automaticamente disponível. Entretanto, isto é bastante complicado, exigindo mudanças no escalonamento das tarefas, que teria de ser realizado no WM e não apenas no PM.
- Incluir um novo tipo de entidade processadora: Agente Móvel, capaz de executar tarefas automáticas que necessitam de dados que não estão disponíveis para o SGWF. Por exemplo, determinar o *host* onde um serviço é oferecido e então executar esse serviço, ou coletar um conjunto de dados de diversos *hosts*. Assim como Aplicações e Papéis, essa nova entidade deve ser tratada de forma diferenciada pelo sistema;
- Investigar a importância de outros aspectos do ambiente de computação móvel (além das desconexões freqüentes e da baixa largura de banda, que já foram considerados), como energia e capacidade de processamento do dispositivo móvel, na execução de tarefas. Por exemplo, se o dispositivo móvel no qual um usuário deseja executar uma tarefa possui pouca energia, o sistema pode evitar que o usuário selecione a tarefa.

Referências Bibliográficas

[1] G. Alonso, D. Agrawal, A. El Abbadi, R. Günthör, M. Kamath, and C. Mohan. Exotica/FMQM: A Persistent Message-based Architecture for Distributed Workflow Management. Technical Report RJ9912, IBM Almaden Research Center, 1994.

[2] G. Alonso, D. Agrawal, A. El Abbadi, R. Günthör, M. Kamath, and C. Mohan. Failure Handling in Large Scale Workflow Management Systems. Technical Report RJ9913, IBM Almaden Research Center, 1994.

[3] G. Alonso, D. Agrawal, A. El Abbadi, R. Günthör, M. Kamath, and C. Mohan. Exotica: A Project on Advanced Transaction Management and Workflow Systems. *ACM SIGOIS Bulletin*, 16(1), 1995.

[4] G. Alonso, D. Agrawal, A. El Abbadi, R. Günthör, M. Kamath, and C. Mohan. Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System. In *Proceedings of the 3rd International Conference on Cooperative Information Systems*, pages 99–110. Vienna, Austria, 1995.

[5] G. Alonso, D. Agrawal, A. El Abbadi, R. Günthör, M. Kamath, and C. Mohan. Advanced Transaction Models in Workflow Context. In *Proceedings of the 12th International Conference on Data Engineering.* New Orleans, USA, 1996.

[6] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. Technical report, IBM Almaden Research Center, 1997.

[7] D. Barbará, S. Mehrotra, and M. Rusinkiewicz. INCAs: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments. Technical report, Department of Computer Science, University of Houston, 1994.

[8] D. Barbará, S. Mehrotra, and M. Rusinkiewicz. INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1), 1996.

[9] C. Büssler. User Mobility in Workflow Management Systems. In *Proceedings of the Telecommunication Information Networking Architecture Conference (TINA95).* Melbourne, Australia, 1995.

[10] Common Object Request Broker Architecture. http://www.corba.org, 2002.

[11] J. Eder, H. Groiss, and W. Liebhart. Workflow Management and Databases. In *Proceedings of the 2nd Forum International d'Informatique Appliquee.* Tunis, 1996.

[12] J. Eder and W. Liebhart. A Transaction-oriented Workflow Activity Model. In *Proceedings of the 9th International Symposium on Computer and Information Systems.* Antalya, Turkey, 1994.

[13] J. Eder and W. Liebhart. The Workflow Activity Model WAMO. In *Proceedings of the 3rd International Conference on Cooperative Information Systems*, pages 87–98. Vienna, Austria, 1995.

[14] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[15] T. Imielinski and B. R. Badrinath. Mobile Wireless Computing: Solutions and Challenges in Data Management. Technical report, Department of Computer Science, Rutgers University, USA, 1993.

[16] IONA Technologies. http://www.iona.com, 2002.

[17] S. Jablonski. Functional and Behavioral Aspects of Process Modelling in Workflow Systems. In A. Benczur G. Chroust, editor, *CON 94 Workflow Management: Challenges, Paradigms and Products*, pages 113–133. R. Oldenburg, 1994.

[18] S. Jablonski. MOBILE: A Modular Workflow Model and Architecture. In *Proceedings of International Working Conference on Dynamic Modelling and Information Systems.* Nordwijkerhout, Netherlands, 1994.

[19] S. Jablonski and C. Büssler. *Workflow Management: Modelling Concepts, Architecture and Implementation.* International Tompson Computer Press, 1996.

[20] Java Development Kit, version 1.2 API Specification. http://java.sun.com/products/jdk/1.2/docs/api, 2002.

[21] J. Jing, K. Huff, H. Sinha, B. Hurtwitz, and B. Robinson. Workflow and Application Adaptations in Mobile Environments. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications.* New Orleans, Louisiana, USA, 1999.

[22] M. Kamath and K. Ramamritham. Bridging the Gap Between Transaction Management and Workflow Management. In *Proceedings of NSF Workshop on Workflow and Process Automation in Information Systems.* Athens, Georgia, 1996.

[23] R. H. Katz. Adaptation and Mobility in Wireless Information Systems. Technical report, Computer Science Division, Electrical Engineering and Computer Science Department, University of California, August 1995.

[24] N. Krishnakumar and A. Sheth. Specification of Workflows with Heterogeneous Tasks in METEOR. Technical Report TM-24198, Bell Communications Research, 1994.

[25] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh. WebWork: METEOR2's Web-based Workflow Management System. *Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems*, 10(2):185–215, 1998.

[26] OMG. The Common Object Request Broker: Architecture and Specification. Technical report, Object Management Group, 1996.

[27] Object Management Group. http://www.omg.org, 2002.

[28] OrbixWeb Programmers's Guide. IONA Technologies PLC. http://www.iona.com, 2002.

[29] OrbixWeb Programmers's Reference. IONA Technologies PLC. http://www.iona.com, 2002.

[30] OrbixWeb – IONA Technologies Java ORB Implementation. http://www.iona.com/products/orbixweb/index.html, 2002.

[31] R. Orfali and D. Harkey. *Client/Server Programming with Java and CORBA.* John Wiley & Sons, Inc., Second edition, 1998.

[32] L. H. Reinehr and M. B. F. Toledo. A CORBA-based Workflow Management System for Wireless Communication Environments. In R. Meersman and Z. Tari, editors, *Proceedings of Confederated International Conferences: CoopIS, DOA and ODBASE*, pages 827–844. Irvine, California, USA, 2002.

[33] L. H. Reinehr and M. B. F. Toledo. WorkToDo: Um Sistema de Gerenciamento de Workflows para Ambientes de Comunicação sem Fio. In *Proceedings of 20th Simpósio Brasileiro de Redes de Computadores*, pages 619–634. Búzios, Rio de Janeiro, Brasil, 2002.

[34] J. Veijalainen, A. Lehtola, and O. Pihlajamaa. Research Issues in Workflow Systems. Technical report, VTT Information Technology, Finland, 1995.

[35] X. Wang. Implementation and Performance Evaluation of CORBA-based Centralized Workflow Schedulers. Technical report, University of Georgia, 1995.

[36] J. Weissenfels, D. Wodtke, G. Weikun, and A. Dittrich. The MENTOR Project: Steps Towards Enterprise-Wide Workflow Management. *IEEE International Conference on Data Engineering*, 1996.

[37] J. Weissenfels, D. Wodtke, G. Weikun, and A. Dittrich. The MENTOR Architecture for Enterprise-Wide Workflow Management. Technical report, University of Saarland, Germany, 1997.

[38] WfMC. The Workflow Reference Model. Technical Report TC00-1003, Workflow Management Coalition Group, 1995.

[39] Workflow Management Coalition. http://www.wfmc.org, 2002.

[40] http://directory.google.com/Top/Computers/Software/Workflow/Products/, 2002.

Apêndice A

Gramática da Linguagem de Definição de Processo

A.1 Notação BNF

Esta Seção apresenta a notação BNF (*Backus Normal Form*), utilizada para descrever a gramática da Linguagem de Definição de Processo.

Notação	Significado
<...>	Símbolos não-terminais. Os caracteres que não são escritos entre < e > são símbolos terminais (tokens)
... ::= ...	Regra de produção. O lado esquerdo contém um símbolo não terminal e o lado direito sua expansão, que pode conter símbolos terminais e não terminais
... | ...	Alternativa
[...]	Opcional. O(s) símbolo(s) entre colchetes podem ou não aparecer na produção
x*	Repetição. O símbolo x (terminal ou não terminal) pode ser repetido 0 ou mais vezes

Tabela A.1: Descrição da notação BNF.

A.2 Tipo de Processo

```
<WORKFLOW>     ::= WORKFLOW <IDENT> { <DEFINITION>* }
<DEFINITION>   ::= FILE <IDENT> { <FILE_DEF> } |
                   QUERY <IDENT> { <QUERY_DEF> } |
                   STRING <IDENT> { <STRING_DEF> } |
                   NUMBER <IDENT> { <NUMBER_DEF> } |
                   TASK <IDENT> :<IDENT> { <TASK_DEF> } |
                   SUB-WORKFLOW <IDENT> :<IDENT> { <SUB_DEF> }
<FILE_DEF>     ::= NAME <STRING> ;
<QUERY_DEF>    ::= DATABASE <STRING> ; EXPRESSION <STRING> ;
<STRING_DEF>   ::= VALUE <STRING>;
<NUMBER_DEF>   ::= VALUE <INTEGER>;
<TASK_DEF>     ::= [<DEPENDS> ;] [<IN_CONTEXT> ;] [<OUT_CONTEXT> ;]
                   [<ROLE> ;] [<DESCRIPTION> ;]
<SUB_DEF>      ::= [<DEPENDS> ;]
<DEPENDS>      ::= DEPENDS DEPENDENCE_RULE
<IN_CONTEXT>   ::= IN_CONTEXT <IDENT> <IDENT_LIST>*
<OUT_CONTEXT>  ::= OUT_CONTEXT <IDENT> <IDENT_LIST>*
<ROLE>         ::= ROLE <IDENT>
<DESCRIPTION>  ::= DESCRIPTION <STRING>
<IDENT_LIST>   ::= , <IDENT>
<IDENT>        ::= ID
<STRING>       ::= STR
<INTEGER>      ::= INT
```

A.3 Modelo de Tarefa

```
<TASK>            ::= TASK <IDENT> { <DEFINITION> }
<DEFINITION>      ::= <TYPE> ; [<APPLICATION> ;] <PRIORITY> ;
                      [<DEADLINE> ;] <DISCONNECTED_OP> ; [<RETRIES> ;]
<TYPE>            ::= TYPE [AUTOMATIC | SEMI_AUTOMATIC | MANUAL]
<APPLICATION>     ::= APPLICATION <IDENT>
<PRIORITY>        ::= PRIORITY <INTEGER>
<DEADLINE>        ::= DEADLINE <INTEGER> [HOURS | DAYS]
<DISCONNECTED_OP> ::= DISCONNECTED_OPERATION <BOOLEAN>
<RETRIES>         ::= RETRIES <INTEGER>
<IDENT>           ::= ID
<INTEGER>         ::= INT
<BOOLEAN>         ::= TRUE | FALSE
```

A.4 Aplicação

```
<APPLICATION> ::= APPLICATION <IDENT> { <DEFINITION> }
<DEFINITION>  ::= <FILENAME> ; <SIZE> ; <OS> ; <CPU> ; [<INSTALLER> ;]
                  <HOSTS> ;
<FILENAME>    ::= FILENAME <STRING>
<SIZE>        ::= SIZE <INTEGER>
<OS>          ::= OS <STRING>
<CPU>         ::= CPU <STRING>
<INSTALLER>   ::= INSTALLER <STRING>
<HOSTS>       ::= HOSTS <STRING> <STRING_LIST>*
<STRING_LIST> ::= , <STRING>
<IDENT>       ::= ID
<STRING>      ::= STR
<INTEGER>     ::= INT
```

A.5 Regra de Dependência

```
<EXPRESSION>   ::= <OPERATOR> ( <EXPRESSION_1> <EXPRESSION_2>* )
<EXPRESSION_1> ::= <ITEM> | <EXPRESSION>
<EXPRESSION_2> ::= , <EXPRESSION_1>
<ITEM>         ::= <IDENT> → <STATE>
<OPERATOR>     ::= AND | OR
<STATE>        ::= READY | RUNNING | FAIL | SUCCEEDED
<IDENT>        ::= ID
```

Apêndice B

Interface dos Servidores WorkToDo

Este Apêndice contém a interface, descrita em IDL (*Interface Definition Language*), dos servidores da arquitetura WorkToDo.

```
module sgwf {
   typedef sequence<string> StringList;
   typedef sequence<octet> ByteList;

   module dMgr {
      interface DefinitionManager {
         // Operações sobre Tipos de Workflow
         boolean addWorkflowType (in string typeName, in string fileName,
                                  in string roleName);
         boolean removeWorkflowType (in string typeName);
         StringList listWorkflowTypes ();
         string getWorkflowTypeInfo (in string typeName);
         string getWorkflowTypeDefinition (in string typeName);

         // Operações sobre Modelos de Tarefas
         boolean addTaskModel (in string modelName, in string fileName);
         boolean removeTaskModel (in string modelName);
         StringList listTaskModels ();
         string getTaskModelInfo (in string modelName);
         string getTaskModelDefinition (in string modelName);

         // Operações sobre Aplicações
         boolean addApplication (in string applName, in string fileName);
         boolean removeApplication (in string applName);
         StringList listApplications ();
```

```
      string getApplicationInfo (in string applName);
      string getApplicationDefinition (in string applName);

      // Operações sobre arquivos
      ByteList readFile (in string fileName);
      boolean writeFile (in string fileName, in ByteList fileContents);
      boolean copyFile (in string source, in string destination);
   };
};


module iMgr {
   interface InstanceManager {
      // Operações sobre Instâncias de Workflow
      string createProcess (in string workflowType, in string hostName,
                            in string creator, in string userInCharge);
      boolean removeProcess (in string processName);
      StringList listProcesses ();
      StringList listProcessesByType (in string workflowType);
      StringList listTerminatedProcesses ();
      StringList listTerminatedProcessesByType (in string workflowType);
      string getHostName (in string processName);
      string getHostAddress (in string processName);
      boolean processTerminated (in string processName,
                            in string processState);
   };
};


module urMgr {
   interface UserRoleManager {
      // Operações sobre Papeis
      boolean addRole (in string roleName);
      boolean removeRole (in string roleName);
      StringList listRoles ();

      // Operações sobre Usuarios
      boolean addUser (in string userName, in string roleName);
      boolean addUserToRole (in string userName, in string roleName);
      boolean removeUser (in string userName);
      boolean removeUserFromRole (in string userName,
```

```
                                    in string roleName);
        StringList listUsers ();
        StringList listUsersByRole (in string roleName);
        StringList listActiveUsers ();
        StringList listActiveUsersByRole (in string roleName);
        StringList userRoles (in string userName);
        boolean userInRole (in string userName, in string roleName);
        boolean userActive (in string userName);
        string getUserInfo (in string userName);
        boolean connectUser (in string userName, in string hostName,
                             in string hostAddress);
        boolean disconnectUser (in string userName);
        boolean addMessage (in string userName, in string sender,
                            in string message);
        boolean removeMessage (in string userName, in long messageID);
        boolean removeAllMessages (in string userName);
        StringList getMessages (in string userName);
    };
};


module pMgr {
    interface ProcessManager {
        boolean init (in string typeName);
        boolean remove ();
        string retrieveState ();
        void newTaskState (in string taskName, in long newState);
        boolean selectTask (in string userName, in string taskName);
        boolean unselectTask (in string userName, in string taskName);
    };

    interface ProcessManagerFactory {
        string createProcessManager (in string processName,
                                     in string userInCharge);
        void deleteProcessManager (in string processName);
    };
};


module wlMgr {
    interface WorklistManager {
```

```
            boolean connect ();
            boolean disconnect ();
            boolean addWorkitem (in string processName,
                                 in string serialTaskDefinition);
            boolean removeWorkitem (in long workitemID);
            StringList listWorkitems ();
            long getWorkitemID (in string processName, in string taskName);
            boolean selectWorkitem (in long workitemID);
            boolean unselectWorkitem (in long workitemID);
            boolean lockWorkitem (in long workitemID);
            boolean unlockWorkitem (in long workitemID);
            boolean executionCompleted (in long workitemID,
                                        in boolean success);
            void prefetch (in boolean enable);
            void changePolicy (in string policy);
            void exchangeWorkitems (in long workitemID1, in long workitemID2);
         };

         interface WorklistManagerFactory {
            string createWorklistManager (in string userName);
            void deleteWorklistManager (in string userName);
         };
      };


      module tMgr {
         interface TaskManager {
            void initiate ();
            void execute (in string appl, in StringList inContext,
                          in StringList outContext);
            void remove ();
         };

         interface TaskManagerFactory {
            TaskManager createTaskManager (in string taskName,
                                           in string processName);
            void deleteTaskManager (in string taskManagerName);
         };
      };
   };
```

Apêndice C

Interface do Gerenciador de Recursos

Este Apêndice apresenta a interface básica do Gerenciador de Recursos. A implementação das funções abaixo depende da plataforma (hardware e sistema operacional) sobre a qual o Gerenciador de Recursos será executado.

```
long getDiskSpace ();
long getBandwidth ();
boolean fileExists (string fileName);
boolean applicationInstalled (string applName);
```