UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM APLICATIVO PARA CONTROLE DE ORDENS DE PRODUÇÃO COM ACESSO VIA INTRANET

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

RODRIGO KRIEGER FERNANDES

BLUMENAU, JULHO/2000

2000/1-58

PROTÓTIPO DE UM APLICATIVO PARA CONTROLE DE ORDENS DE PRODUÇÃO COM ACESSO VIA INTRANET

RODRIGO KRIEGER FERNANDES

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Wilson Pedro Carli — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Wilson Pedro Carli

_____

Prof. Oscar Dalfovo

_____

Prof. Maurício Capobianco Lopes

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

O trabalho proposto abordará o estudo das tecnologias *Component Object Model* (COM) e *Active Server Pages* (ASP). Paralelamente será desenvolvido um protótipo de software aplicativo para auxiliar no controle de Ordens de Produção. O protótipo visa reduzir alguns problemas de uma empresa no setor metalúrgico para cálculo de suas horas de produção.

ABSTRACT

The proposed work will approach the study of *Component Object Model* (COM) and *Active Server Pages* (ASP) technologies. It will be developed an application software prototype to aid in the Production Orders control. The prototype tries to reduce some problems of a metallurgy company on its production times calculation.

1 INTRODUÇÃO

Nos dias de hoje, face à necessidade de informações mais confiáveis, as empresas procuram, conforme a tecnologia e recursos financeiros disponíveis, desenvolver aplicativos que assegurem um andamento das suas necessidades diárias. Sendo assim, qualquer área sensível de uma empresa que tenha processos controlados manualmente, devem passar a ter controles automatizados, de preferência integrados a outros processos da empresa. Para tanto, é necessário o uso de tecnologias na área de informática que auxiliem o desenvolvimento de aplicativos.

Algumas das recentes tecnologias disponíveis são o *Component Object Model* (COM) e *Active Server Pages* (ASP), e a metodologia de orientação a objetos. O ASP é uma tecnologia de *scripts* que rodam no servidor, que pode ser usada para criar aplicações web dinâmicas e interativas. Uma página ASP é uma página HTML que contém *scripts* que são processados pelo servidor web antes de serem enviadas ao navegador do cliente. Segundo [AND1999], é possível estender as capacidade do ASP utilizando componentes COM, provendo um método compacto, reusável e seguro de obter acesso às informações, além de outras vantagens como facilidade na manutenção e a distribuição mais simples e segura. Componentes permitem construir aplicações web mais robustas, com melhor performance e escalabilidade.

Segundo [GRI1998], o COM é uma especificação e um conjunto de serviços que permitem ao profissional de computação criar aplicações distribuídas modulares, orientadas a objetos, customizadas e atualizáveis, usando diversas linguagens. COM não é a primeira forma de se reutilizar código compilado. Um exemplo são as famosas *Dynamic Link Libraries* (DLL), que foram e ainda são utilizadas em programas para Windows. Algumas vantagens das DLL's são que elas permitem que parte da aplicação seja substituída ou atualizada sem a necessidade de uma recompilação completa, podem ser carregadas na memória somente quando são necessárias e o código pode ser compartilhado entre processos.

A metodologia de orientação a objetos é definida por [RUM1994] como "uma nova maneira de pensar os problemas utilizando modelos organizados a partir de conceitos do mundo real. O componente fundamental é o objeto que combina estrutura e comportamento em uma única unidade". A tecnologia de objetos oferece modularidade de seus elementos,

podendo-se tomar um subconjunto existente e integrá-lo de uma maneira diferente em outra parte do sistema. Apresenta também uma correspondência com o mundo real, visualizando objetos da natureza conforme são, individualizados e caracterizados com finalidade própria. Tais objetos permitem ser combinados ou utilizados separadamente, pois, em tese, apresentam baixa dependência externa e alta coesão interna de seus componentes, o que permite promover substituições sem afetar interconexões ou interferir com a operação dos demais objetos [FUR1998].

Desta forma, o presente trabalho de conclusão de curso apresentará o desenvolvimento de um protótipo de software aplicativo para o Controle de Ordens de Produção, utilizando-se de COM e ASP, que será acessado através de uma intranet, bem como fazendo uso da metodologia de orientação a objetos. Para a descrição da modelagem será utilizada a ferramenta Rational Rose 2000, a qual suporta a metodologia UML. No desenvolvimento do componente COM, utilizar-se-á a ferramenta Microsoft Visual C++.

1.1 OBJETIVOS

O objetivo principal deste trabalho é o desenvolvimento de um software aplicativo para auxiliar no Controle de Ordens de Produção e a disponibilização das informações através de uma intranet utilizando ASP.

1.2 ORGANIZAÇÃO

Este trabalho está composto da seguinte maneira:

O capítulo 1 faz uma introdução ao trabalho proposto.

O capítulo 2 traz uma introdução sobre administração e controle da produção.

O capítulo 3 fornece uma visão da metodologia de análise orientada a objetos.

O capítulo 4 descreve as tecnologias e ferramentas utilizadas, como o COM e o ASP.

No capítulo 5 são apresentadas as especificações do protótipo, englobando o seu funcionamento e aspectos de implementação.

O capítulo 6 faz uma análise conclusiva sobre o trabalho, limitações e futuras extensões que poderão ser realizadas.

No capítulo 7 consta a referência bibliográfica.

2 CONTROLE DE PRODUÇÃO

2.1 ADMINISTRAÇÃO DA PRODUÇÃO

Segundo [CHI1991], a Administração da Produção (AP) é a área da administração que cuida dos recursos físicos e materiais da empresa que realizam o processo produtivo. É através da administração da produção que uma empresa extrai as matérias-primas, transforma-as para produzir o produto acabado ou presta serviços especializados que a empresa fornece ao mercado.

Os dois grandes objetivos da administração da produção são alcançar eficiência e eficácia na administração dos recursos físicos e materiais. A eficiência significa a utilização adequada dos recursos empresariais, e está relacionada com os meios (métodos, procedimentos, normas, processos, etc). Reside basicamente em fazer as coisas corretamente, isto é, da melhor maneira possível. A eficácia significa o alcance dos objetivos da empresa. A eficácia está ligada aos fins, isto é, aos objetivos que a empresa pretende alcançar através do seu funcionamento. Reside basicamente em fazer as coisas que são importantes para os resultados.

A administração da produção apresenta geralmente uma estrutura organizacional voltada para o máximo aproveitamento dos recursos físicos e materiais da empresa. Isto significa que ela se estrutura de acordo com o sistema de produção adotado e com a tecnologia empregada. A estrutura organizacional de administração da produção é geralmente composta pelos seguintes órgãos:

a) desenvolvimento do produto: é a área da administração da produção que cuida do planejamento e desenvolvimento do produto, de suas especificações, características, etc;

b) engenharia industrial: cuida do arranjo físico e layout, processo produtivo, tempos e movimento, etc;

c) planejamento e controle da produção: trata de planejar e controlar a produção de acordo com a capacidade de produção da empresa;

d) produção: é responsável pelas operações de produção, da atividade dos operários e das máquinas e, portanto, da transformação de matérias-primas em produtos acabados;

e) administração de materiais: cuida da busca, aprovisionamento e abastecimento dos materiais necessários à produção dos bens e serviços da empresa;

f) controle de qualidade: é responsável pela inspeção dos produtos ou serviços executados para verificar se estão de acordo com as especificações da engenharia de produto;

g) manutenção: cuida da preservação dos recursos físicos e materiais da empresa, isto é, dos prédios, instalações, máquinas e equipamentos.

2.2 SISTEMAS DE PRODUÇÃO

As empresas, assim como todos os organismos vivos, funcionam como sistemas. Um sistema é um conjunto de partes inter-relacionadas Cada parte de um sistema pode ser um órgão, um departamento ou um subsistema. Em outras palavras, todo sistemas é constituído de vários subsistemas. Por outro lado, todo sistema faz parte integrante de um sistema maior. Segundo [CHI1991], os principais componentes de um sistema são:

a) entradas (*inputs*): as entradas constituem tudo o que ingressa em um sistema para permitir que ele funcione;

b) saídas (*outputs*): as saídas ou resultados constituem tudo aquilo que o sistema produz e devolve ao ambiente externo;

c) processamento: o processamento ou transformação é o que o sistema realiza com as estradas para proporcionar as saídas. É o funcionamento interno do sistema;

d) retroação (*feedback*): a retroação é a influência que as saídas do sistema exercem sobre suas entradas no sentido de ajustá-las ou regulá-las ao funcionamento do sistema.

Cada empresa adota um sistema de produção para realizar as suas operações e produzir seus produtos ou serviços da melhor forma possível e, com isto, garantir sua eficiência e eficácia. O sistema de produção é a maneira pela qual a empresa organiza seus órgãos e realiza suas operações de produção, adotando uma interdependência lógica entre as etapas do processo produtivo, desde o momento em que os materiais e as matérias-primas saem do almoxarifado até chegar ao depósito como produto acabado. As entradas e insumos que vêm dos fornecedores ingressam no sistema de produção através do almoxarifado de materiais, sendo ali estocados até sua eventual utilização pela produção. A produção processa e transforma os materiais e matérias-primas em produtos acabados, que são estocados no depósito de produtos acabados até sua entrega aos clientes e consumidores. A interdependência entre o almoxarifado, a produção e o depósito é muito grande, fazendo com que qualquer alteração em um deles provoque influências sobre os demais.

Existem três tipos de sistemas de produção:

a) sistema de produção sob encomenda: é o sistema de produção utilizado pela empresa que somente produz após ter recebido o pedido ou a encomenda de seus produtos. Apenas após o contrato ou encomenda de um determinado produto é que a empresa vai produzi-lo;

b) sistema de produção em lotes: é o sistema utilizado por empresas que produzem um quantidade limitada de um tipo de produto de cada vez. Essa quantidade limitada é denominada lote de produção, e cada lote é dimensionado para atender a um determinado volume de vendas previsto para um período;

c) sistema de produção contínua: é utilizado por empresas que produzem determinado produto, sem modificações, por um longo período de tempo. O ritmo de produção é acelerado e as operações são executadas sem interrupção ou mudança. Como o produto é sempre o mesmo e o processo produtivo não sofre mudanças, o sistema pode ser aperfeiçoado continuamente.

2.3 PLANEJAMENTO E CONTROLE DA PRODUÇÃO

O planejamento e controle da produção (PCP) planeja e programa a produção e as operações da empresa, bem como as controla adequadamente para tirar melhor proveito possível em termos de eficiência e eficácia [CHI1991].

A finalidade do PCP é aumentar a eficiência e eficácia do processo produtivo da empresa. Tem, portanto, dupla finalidade: atuar sobre os meios da produção para aumentar a eficiência e cuidar para que os objetivos de produção sejam plenamente alcançados para aumentar a eficácia.

Para atender essa dupla finalidade, o PCP tem uma dupla função: planejar a produção e controlar seu desempenho. O PCP estabelece antecipadamente o que a empresa deverá produzir, planejando o processo produtivo, programando materiais, máquinas, pessoas e estoques, bem como monitora e controla o desempenho da produção em relação ao que foi planejado, corrigindo eventuais desvios ou erros que possam surgir, controlando o funcionamento do processo produtivo para mantê-lo de acordo com o que foi planejado.

Para poder funcionar satisfatoriamente, o PCP exige um enorme volume de informações. Na realidade, o PCP recolhe dados e produz informações incessantemente. É um centro de informações para a produção. Neste sentido, o PCP apresenta quatro fases principais, que são o projeto de produção, a coleta de informações, o planejamento da produção e o controle da produção.

O projeto da produção constitui a primeira fase do PCP e é também denominado pré-produção ou planejamento de operações. Nesta primeira fase, procura-se definir como o sistema de produção funciona e quais as suas dimensões para se estabelecer os parâmetros do PCP. O projeto de produção é relativamente permanente e sofre poucas mudanças com o tempo, a não ser que o sistema de produção passe por alterações com a aquisição de novas máquinas, mais pessoal ou novas tecnologias, e, toda vez que ocorram, tais mudanças devem alterar o projeto de produção.

A segunda fase do PCP é a coleta de informações para que o esquema do projeto de produção possa ser devidamente montado, quantificado e dinamizado. No fundo, a coleta de informações constitui o detalhamento da primeira fase, isto é, do projeto de produção.

O planejamento da produção constitui a terceira fase do PCP e visa estabelecer o que a empresa deverá produzir em um determinado período de tempo, tendo em vista, de uma lado, a sua capacidade de produção e, de outro, a previsão de vendas que deve ser atendida. O planejamento da produção visa compatibilizar a eficácia e a eficiência e procura coordenar e integrar máquinas, pessoas, matérias-primas e processos produtivos em um todo sistêmico e harmonioso. O planejamento da produção se assenta das primeiras fases do PCP sobre o processo produtivo, e é realizado em três etapas: formulação do plano de produção, implementação do plano de produção através da programação da produção e a execução do plano de produção através das emissões de ordens.

O plano de produção representa aquilo que a empresa pretende produzir dentro de um determinado exercício ou período de tempo. Geralmente, esse exercício ou período é de um ano, quando se trata de produção contínua ou em lote. Quando se trata de produção sob encomenda, o plano de produção cobre o tempo necessário para a execução do produto. A elaboração do plano de produção depende do sistema de produção utilizado pela empresa.

A partir da formulação do plano de produção, o PCP passa a cuidar da programação da produção. A programação da produção é o detalhamento do plano de produção para que ele possa ser executado de maneira integrada e coordenada pelos diversos órgãos produtivos. Programar a produção é determinar quando deverão ser realizadas as tarefas e operações de produção e quanto deverá ser feito. A programação da produção detalha e fragmenta o plano de produção para que possa ser executado no dia-a-dia da empresa. Para tanto, ela faz estabelece a seqüência do processo produtivo e estabelece as data de início e fim de cada atividade.

Programada a produção, os diversos órgãos envolvidos direta ou indiretamente no processo produtivo devem executá-la de maneira integrada e coordenada. Para que isso possa acontecer, a programação da produção transforma o plano de produção em uma infinidade de ordens que deverão ser executadas pelos diversos órgãos da empresa, como produção, compras, depósito, entre outros. Para tanto existem os seguintes tipos de ordens:

a) ordem de produção: é a comunicação para produzir que é enviada à seção produtiva, autorizando-a a executar determinado volume de produção;

b) ordem de montagem: corresponde a uma ordem de produção destinada aos órgãos produtivos de montagem ou de acabamento;

c) ordem de compra: comunicação para comprar matéria-prima ou material enviada ao órgão de compras;

d) ordem de serviço: comunicação sobre prestação interna de serviços;

e) requisição de materiais: solicita matéria-prima ou material do almoxarifado para alguma seção produtiva.

Com a emissão das ordens, todos os órgãos envolvidos direta ou indiretamente no processo produtivo passam a trabalhar em conjunto. As ordens representam decisões que cada órgão deverá executar para que todo o processo produtivo se desenvolva da melhor maneira possível.

2.4 CONTROLE DA PRODUÇÃO

Segundo [CHI1991], o controle visa guiar e regular as atividades da empresa a fim de garantir o alcance dos objetivos almejados. Se as coisas ocorressem de acordo com o que foi planejado, não haveria necessidade de controle. O controle existe porque alguma coisa pode sair diferente daquilo que foi planejado. O controle é a função administrativa que consiste em medir e corrigir o desempenho para assegurar que os objetivos da empresa sejam plenamente atingidos. A tarefa do controle é verificar se tudo está sendo feito de conformidade com o que foi planejado e organizado, de acordo com as ordens dadas, para identificar os erros ou desvios, a fim de corrigi-los e evitar sua repetição, ou seja, avaliar se tudo ocorreu conforme os padrões estabelecidos.

Um padrão é uma norma ou critério que serve de base para a avaliação ou comparação de alguma coisa, é o ponto de referência para aquilo que está sendo feito. O controle da produção geralmente avalia a produção de acordo com padrões de quantidade, padrões de qualidade, padrões de tempo e padrões de custo.

Os principais padrões de quantidade são:

a) volume de produção: significa a quantidade de produtos ou serviços produzidos dentro de um determinado período. Representa a saída do sistema de produção, isto é, o resultado final da atividade produtiva;

b) número de horas trabalhadas: representa a quantidade de trabalho efetuado em número de horas. Geralmente utiliza apenas a mão-de-obra direta como referência e é importante para dar a dimensão do esforço de trabalho realizado no período.

Qualidade é a adequação a alguns padrões previamente definidos. Esses padrões são denominados especificações, e quando as especificações não são bem definidas, a qualidade torna-se ambígua e a aceitação ou rejeição do produto passa a ser discutível. Diz-se que um produto é de alta qualidade quando ele atende exatamente aos padrões estabelecidos e exibe as exatas especificações adotadas.

Os padrões de tempo consistem no tempo-padrão de produção e nos padrões de rendimento. O tempo-padrão de produção representa o nível satisfatório de produção determinado para cada trabalhador em determinado período de tempo. O tempo-padrão é obtido através do estudo que trabalho, que é a verificação de um trabalho realizado por um trabalhador para medir o tempo gasto na sua execução e melhorar sua eficiência. O padrão de rendimento pode ser estabelecido com a determinação dos tempos-padrão, os quais servirão de base para o acompanhamento da produção.

3 UNIFIED MODELING LANGUAGE – UML

Segundo [FUR1998], a UML é uma tentativa de padronizar a modelagem orientada a objetos de forma que qualquer sistema, seja qual for o tipo, possa ser modelado corretamente, com consistência, fácil de se comunicar com outras aplicações, simples de ser atualizado e compreensível.

Diante da diversidade de conceitos das diversas metodologias de orientação a objetos, Grady Booch, James Rumbaugh e Ivar Jacobson decidiram criar uma Linguagem de Modelagem Unificada. Eles disponibilizaram inúmeras versão preliminares da UML para a comunidade de desenvolvedores e a resposta incrementou muitas novas idéias que melhoraram ainda mais a linguagem.

Em janeiro de 1997 a UML foi submetida ao consórcio *Object Management Group* (OMG) como uma proposta para um padrão de notação para técnicas de análise e projeto orientados a objetos. Este documento foi aprovado em novembro de 1997.

Os objetivos da UML são:

a) a modelagem de sistemas (não apenas software) usando os conceitos da orientação a objetos;

b) estabelecer uma união fazendo com que métodos conceituais sejam também executáveis;

c) criar uma linguagem de modelagem usável tanto pelo homem quanto pela máquina.

A UML está destinada a ser dominante, a linguagem de modelagem comum usada nas indústrias. Ela está totalmente baseada em conceitos e padrões extensivamente testados provenientes das metodologias existentes anteriormente, e também é muito bem documentada com toda a especificação da semântica da linguagem representada em meta-modelos.

Existem cinco fases no desenvolvimento de sistemas de software em UML: análise de requisitos, análise, projeto, programação e testes. Estas cinco fases não devem ser executadas nesta ordem, mas concomitantemente de forma que problemas detectados numa

certa fase modifiquem e melhorem as fases desenvolvidas anteriormente de forma que o resultado global gere um produto de alta qualidade e performance.

As fases de análise de requisitos, análise e projeto utilizam-se em seu desenvolvimento cinco tipos de visões, nove tipos de diagramas e vários modelos de elementos que serão utilizados na criação dos diagramas e mecanismos gerais. Todos, em conjunto, especificam e exemplificam a definição do sistema, tanto no que diz respeito à funcionalidade estática quanto dinâmica do desenvolvimento de um sistema.

Antes de se abordar cada um destes componentes separadamente, serão definidas as partes que compõem a UML [FUR1998]:

a) *visões*: As visões mostram diferentes aspectos do sistema que está sendo modelado. A visão não é um gráfico, mas uma abstração consistindo em uma série de diagramas. Definindo um número de visões, cada uma mostrará aspectos particulares do sistema, dando enfoque a ângulos e níveis de abstrações diferentes e uma figura completa do sistema poderá ser construída. As visões também podem servir de ligação entre a linguagem de modelagem e o método/processo de desenvolvimento escolhido;

b) *modelos de elementos*: Os conceitos usados nos diagramas são modelos de elementos que representam definições comuns da orientação a objetos como as classes, objetos, mensagem, relacionamentos entre classes incluindo associações, dependências e heranças;

c) *mecanismos gerais*: Os mecanismos gerais provém comentários suplementares, informações, ou semântica sobre os elementos que compõem os modelos. Eles provém também mecanismos de extensão para adaptar ou estender a UML para um método/processo, organização ou usuário específico;

d) *diagramas*: Os diagramas são os gráficos que descrevem o conteúdo em uma visão. A UML possui nove tipos de diagramas que são usados em combinação para prover todas as visão do sistema.

3.1 ANÁLISE DE REQUISITOS

Esta fase captura as intenções e necessidades dos usuários do sistema a ser desenvolvido através do uso de funções chamadas casos de uso. Através do desenvolvimento de casos de uso, as entidades externas ao sistema (em UML chamam-se de atores externos) que interagem e possuem interesse no sistema são modelados entre as funções que eles requerem, funções estas chamadas de casos de uso [FUR1998].

Os atores externos e os casos de uso são modelados com relacionamentos que possuem comunicação associativa entre eles ou são desmembrados em hierarquia. Cada caso de uso modelado é descrito através de um texto, e este especifica os requerimentos do ator externo que utilizará estes casos de uso.

O diagrama de casos de uso mostrará o que os atores externos, ou seja, os usuários do futuro sistema deverão esperar do aplicativo, conhecendo toda sua funcionalidade sem importar como este será implementado. A análise de requisitos também pode ser desenvolvida baseada em processos de negócios, e não apenas para sistemas de software.

3.2 ANÁLISE

Segundo [FUR1998], a fase de análise está preocupada com as primeiras abstrações (classes e objetos) e mecanismos que estarão presentes no domínio do problema. As classes são modeladas e ligadas através de relacionamentos com outras classes e são descritas no Diagrama de Classe. As colaborações entre classes também são mostradas neste diagrama para desenvolver os casos de uso modelados anteriormente. Esta colaborações são criadas através de modelos dinâmicos em UML.

Na análise, só serão modeladas classes que pertençam ao domínio principal do problema do software, ou seja, classes técnicas que gerenciem banco de dados, interface, comunicação, concorrência e outros não estarão presentes neste diagrama.

3.3 PROJETO

Na fase de projeto, o resultado da análise é expandido em soluções técnicas. Novas classes serão adicionadas para prover uma infra-estrutura técnica: a interface do usuário e de periféricos, gerenciamento de banco de dados, comunicação entre outros sistemas, dentre outros. As classe do domínio do problema modeladas na fase de análise são mescladas nessa nova infra-estrutura técnica tornando possível alterar tanto o domínio do problema quanto a infra-estrutura. O projeto resulta no detalhamento das especificações para a fase de programação do sistema [FUR1998].

3.4 PROGRAMAÇÃO

Na fase de programação, as classes provenientes do projeto são convertidas para o código da linguagem orientada a objetos escolhida (a utilização de linguagens procedurais é extremamente não recomendada). Dependendo da capacidade da linguagem usada, essa conversão pode ser uma tarefa fácil ou muito complicada. No momento da criação de modelos de análise e projeto em UML, é melhor evitar traduzi-los mentalmente em código.

Nas fases anteriores, os modelos criados são o significado do entendimento e da estrutura do sistema. Então, no momento da geração do código onde o analista conclua antecipadamente sobre modificações em seu conteúdo, seus modelos não estarão mais demonstrando o real perfil do sistema. A programação é uma fase separada e distinta onde os modelos criados são convertidos em código [FUR1998].

3.5 DIAGRAMAS DA UML

Na UML o modo para descrever os vários aspectos de modelagem, é através da notação definida pelos seus vários tipos de diagramas. Um diagrama é uma apresentação gráfica de uma coleção de elementos de modelo, freqüentemente mostrado como um gráfico conectado por arcos (relacionamentos) e vértices (outros elementos do modelo)

3.5.1 DIAGRAMAS DE CASOS DE USO

[FUR1998] propõe como primeiro passo para a análise, definir os casos de uso que descrevam o que o sistema irá fornecer em termos de funcionalidade. Um diagrama de caso de uso é um gráfico de atores, um conjunto de casos incluídos por um limite de domínio, comunicação, partição e associações entre atores, assim como generalizações entre casos de uso.

Na UML, deve-se verificar quatro elementos básicos em um diagrama de caso de uso:

a) ator: é um agente que deve interagir com o sistema, um tipo de usuário ou categoria com papel definido, podendo incluir seres humanos, máquinas, dispositivos ou outros sistemas. Atores não precisam ser forçosamente pessoas, tipicamente são clientes, usuários, gerentes, computadores, etc;

b) caso de uso: a UML define como um conjunto de seqüências de ações que um sistema desempenha para produzir um resultado observável de valor à um ator específico. O caso de uso é a interação típica entre o sistema e um usuário, um modo específico de utilização a partir de um ponto de vista segmentado de funcionalidade;

c) interação: o ator comunica-se com os sistemas através do envio e recebimento de mensagens, sendo que um caso de uso é sempre iniciado a partir do momento que um ator envia a sua mensagem (estímulo);

d) sistema: é visto como uma caixa-preta que fornece situações de aplicação (os casos de uso). Não é importante na fase de análise compreender como o sistema implementa os casos de uso ou como ocorre o funcionamento interno.

3.5.2 DIAGRAMA DE CLASSES

O diagrama de classes é a essência da UML, resultado de uma combinação de diagramas propostos pela OMT, Booch e vários outros métodos. A metodologia define esse

diagrama como uma estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo, como classes, tipos e seus respectivos conteúdos e relações.

O diagrama de classes consiste de Objetos, Classes, Operações, Associações e Restrições. [FUR1998] sugere o seguinte para a definição dos diagramas de classe:

a) ter todas as informações necessárias para uma análise completa de um modelo de objetos é geralmente improvável na prática, muito menos interpretar e representar todas as interações corretamente. O procedimento recomendado consiste na identificação dos comportamentos iniciais, definindo os objetos que exibem esses comportamentos, classificando os objetos, identificando os relacionamentos entre eles e modelando os seus ciclos de vida. Uma análise cuidadosa, focando as abstrações comportamentais, promove uma redução de objetos para calçar o caminho de uma boa construção hierárquica e hereditariedade e o seu uso prudente pode ajudar a produzir uma clara e compreensível estrutura de aplicação;

b) embora seja importante ter conhecimento completo acerca da teoria envolvida para a construção de diagramas de classe, não se deve tentar empregar inicialmente todos os tipos de notação oferecidos. Deve-se começar de forma simples: classes, associações, atributos e generalizações; deve-se apenas introduzir outras notações quando forem necessárias;

c) compreender a perspectiva na qual se está trabalhando. Em análise deve-se construir modelos conceituais, ao trabalhar com software deve-se concentrar em modelos de especificação. Os modelos de implementação devem ser criados somente quando estiver ilustrado uma técnica de implementação particular;

d) não desenhar modelos para tudo, e concentrar-se apenas nas áreas chaves. É melhor ter poucos diagramas que serão muito utilizados do que muitos diagramas que serão pouco utilizados e esquecidos.

3.5.3 DIAGRAMA DE INTERAÇÃO

Diagrama de interação é um termo genérico que se aplica a vários tipos de diagramas que enfatizam interações de objetos. Uma interação é uma especificação comportamental que inclui uma seqüência de trocas de mensagens entre um conjunto de objetos dentro de um contexto para realizar um propósito específico, tal como a realização de um caso de uso.

Para especificar uma interação, é necessário definir um contexto de caso de uso e estabelecer os objetos que interagem e seus relacionamentos. Com isso, diagramas de interação são aplicados para mostrar a realização de caso de uso, e as possíveis seqüências de interação são especificadas em uma descrição única que contém condicionais (ramificações ou sinais condicionais) ou através de descrições múltiplas, cada uma detalhando um caminho particular pelos possíveis caminhos de realização [FUR1998].

Diagramas de interação são apresentados sob duas formas na UML:

a) *diagrama de seqüência*: expõe o aspecto do modelo que enfatiza o comportamento dos objetos em um sistema, incluindo suas operações, interações, colaborações e histórias de estado em seqüência temporal de mensagem e representação explícita de ativação de operações;

b) *diagrama de colaboração*: mostra o contexto completo de uma interação, inclusive os objetos e seus relacionamentos pertinentes a uma interação particular, sendo freqüentemente melhores para propósitos de desenho. Um diagrama de colaboração é um contexto, um gráfico de objetos e vínculos com fluxo de mensagem anexado.

3.5.4 DIAGRAMA DE ESTADO

Para [FUR1998], a contribuição dessa técnica está em prover uma definição formal explícita de comportamento, permitindo uma verificação dos eventos e transições de estados aos quais um sistema está sujeito.

A UML propõe o emprego do diagrama de estado de maneira individualizada para cada classe, com o objetivo de tornar o estudo simples o bastante para se ter uma diagrama de estado compreensível.

Para a definição do diagrama de estados deve-se observar o ciclo de vida de objetos de uma classe, observar como são gerados, as posições que assumem durante suas vidas, se dão origem a outros objetos em classes relacionadas e sua destruição.

Um caso especial de diagrama de estado, proposto na UML, é o diagrama de atividade. Nele tudo, ou a maioria dos estados é estado de ação e a maioria das transições é ativada por conclusão de ações nos estados precedentes. É útil quando se pretende descrever um comportamento paralelo ou mostrar como interagem comportamentos em vários casos de uso.

3.5.5 DIAGRAMAS DE IMPLEMENTAÇÃO

Na UML, aspectos de implementação física são modelados através de diagramas de implementação, que são derivados do módulo de Booch e diagramas de processo, mas modificados para centrar em especificações de comportamentos interconectados.

O diagrama de implementação da UML existe sob duas formas:

a) *diagrama de componente*: mostra a estrutura do próprio código fonte;

b) *diagrama de implantação*: mostra a estrutura do sistema de tempo real.

4 TECNOLOGIAS E FERRAMENTAS UTILIZADAS

4.1 MICROSOFT COMPONENT OBJECT MODEL – COM

O modelo de programação *Component Object Model* (COM) é um modelo cliente/servidor baseado em objetos, designado a promover a interoperabilidade entre softwares. O primeiro mérito do COM é prover um meio para objetos clientes fazerem uso de objetos servidores, sem se preocupar com o fato de que os dois podem ter sido desenvolvidos por companhias diferentes, usando linguagens de programação diferentes, em datas diferentes. Para permitir este nível de interoperabilidade COM define um padrão binário, o qual especifica como um objeto é carregado na memória em tempo de execução. Por definir como um objeto é carregado na memória, COM permite a qualquer linguagem que seja capaz de reproduzir o leiaute de memória requerido, criar um objeto COM [RED1997].

O objetivo principal do COM é fornecer interoperabilidade básica entre objetos clientes e servidores no nível binário. Segundo [RED1997], COM também tem vários outros objetivos:

a) prover uma solução para a evolução dos novos problemas;

b) prover uma visão sistêmica dos objetos;

c) prover um modelo de programação único;

d) prover suporte para processamento distribuído.

No modelo de programação COM, clientes COM conectam-se com um ou mais objetos COM, os quais estão contidos em servidores COM. Aqui, um cliente é qualquer peça de software que faz uso dos serviços fornecidos por um objeto COM. Cada objeto COM disponibiliza seus serviços através de uma ou mais interfaces, as quais são essencialmente relações de funções semanticamente agrupadas. A implementação compilada de cada objeto COM fica contida dentro de um módulo binário (Executável - EXE ou Biblioteca de Vínculo Dinâmico – *Dynamic Link Library* - DLL) chamado de servidor COM. Um simples servidor COM é capaz de conter a implementação compilada de vários objetos COM diferentes. O modelo de programação COM define o que um servidor COM precisa fazer para expor

objetos COM, o que um objeto COM precisa fazer para expor seus serviços, e o que um cliente COM precisa fazer para usar os serviços de um objeto COM.

4.1.1 OBJETOS COM

Um objeto COM, assim como qualquer outro objeto, é uma instância feita em tempo de execução de uma classe definida. Contudo, diferente da maioria dos outro objetos, os quais são identificados por um nome que tem algum significado, objetos COM são identificados por uma Identificação de Classe (*Class Identifier – CLSID*). CLSIDs são parte de um grupo especial de identificadores chamados Identificadores Globais Únicos (*Globally Unique Identifiers – GUIDs*). GUIDs são valores de 128 bits estatisticamente garantidos como únicos através do tempo e espaço. O Quadro 1 ilustra como uma GUID foi definida.

Quadro 1: Definição da estrutura de uma GUID

```
typedef struct GUID
{
  DWORD   Data1; //32 bits
  WORD    Data2; //16 bits
  WORD    Data3; //16 bits
  WORD    Data4[8]; //64 bits
}GUID;
```

Fonte: [RED1997]

A estrutura interna de uma GUID dificilmente precisa ser acessada diretamente, exceto por questões de depuração ou quando as GUIDs são transmitidas entre máquinas com ordenação de bytes diferentes.

Para entender como é necessário que a especificação COM use CLSIDs para identificar classes de objetos, pode-se considerar o seguinte cenário: imagine que alguém desenvolveu um objeto COM e o identificou usando um nome comum, como “MeuObjeto”. Então esse objeto, na sua forma binária, foi distribuído para centenas de desenvolvedores, os quais rapidamente instalaram-no. Se um desses desenvolvedores já possuir um outro objeto chamado “MeuObjeto” em seu sistema, ele não conseguirá resolver o conflito de nomes gerado porque os objetos estão na sua forma binária. Por isso, para prevenir este tipo de

conflito de nomes, COM faz uso de CLSIDs para ter um identificador único para cada classe de objeto [RED1997].

Ao invés de criar uma central responsável por controlar e distribuir GUIDs, COM disponibilizou uma função em sua API (*Application Programming Interface*) chamada CoCreateGuid, a qual é usada por várias ferramentas geradoras de GUIDs, como GUIDGEN.EXE e UUIDGEN.EXE [RED1997].

Enquanto CLSIDs são considerados adequados para identificar uma classe como única, eles não são muito bons para se usar em sua forma básica. Para se fazer uso deles de maneira mais eficiente e mais parecido com o desenvolvimento orientado a objetos, pode-se associar a ele uma forma legível e fácil de usar dentro das aplicações pelo desenvolvedor, onde se atribui o valor do CLSID à uma constante, conforme indicado no Quadro 2 [RED1997].

Quadro 2: Atribuição de um CLSID a um nome legível

```
const CLSID CLSID_MeuObjeto = {0x7af31102, 0x7a1b, 0x11d0,
     {0xba, 0xdc, 0x00, 0x80, 0xc7, 0xb2, 0x48, 0x80}};
```

Fonte: [RED1997]

4.1.2 INTERFACES

COM define meios padrão completos para criar componentes e controlar a comunicação entre estes componentes e seus clientes. Diferente dos ambientes tradicionais de programação, este mecanismo é independente das aplicações que utilizam o componente e das linguagens de programação usadas para criar estes objetos. COM então define um padrão de interoperabilidade binário assim como o de uma linguagem qualquer. Utilizando-se de COM, aplicações interagem com qualquer outra e com o sistema, através de coleções de chamadas de funções conhecidas como *Interfaces*. Em COM, uma interface é um tipo forte de contrato entre um componente de software e um cliente e provê um, relativamente pequeno, mas eficiente grupo de funções semanticamente parecidas. Isto é uma articulação de um comportamento e responsabilidades esperados, e ele fornece aos programadores e designers uma entidade concreta para usar quando estes se referem ao componente. Embora não exista

requerimentos rígidos do modelo, interfaces precisam ser divididas em partes semelhantes para que possam ser reusadas em uma variedade de contextos [EDD1998].

Cada interface é identificada por um identificador único chamado de identificador de interface (*Interface Identifier*) ou IID, similar ao meio como um objeto COM é identificado por um CLSID único. Parecido com o CLSID, IIDs também são GUIDs e também é bom associar a cada IID uma constante com forma legível e fácil de usar, atribuindo o valor do IID à uma constante [RED1997]. O Quadro 3 ilustra como fazer a associação de um IID a um nome legível:

Quadro 3: Atribuição de um IID a um nome legível

```
const IID IID_MinhaInterface = {0x7af31102, 0x7a1b, 0x11d0,
     {0xba, 0xdc, 0x00, 0x80, 0xc7, 0xb2, 0x48, 0x80}};
```

Fonte: [RED1997]

Interfaces são essenciais para a programação COM porque elas são o único meio de interagir com o objeto COM. Em vez de obter um ponteiro para todo o objeto COM, um cliente COM precisa obter apenas um ponteiro para uma interface em particular, a qual é então usada para acessar as funções definidas como parte desta interface em particular. A única maneira de acessar funções de uma interface em particular é através do ponteiro para esta interface [RED1997].

O uso de interfaces COM serve de instrumento para criar componentes de software adaptáveis. COM fornece as funcionalidades necessárias para os componentes se desenvolverem com o tempo. Um dos maiores problemas para este esforço em softwares baseados em componentes está na publicação de novas versões. Versões de componentes de software podem ser problemáticas quando um componente tem parte de seu código modificado. Este é um problema comum com as *Dynamic Link Libraries* (DLLs) no Windows – um programa que depende de uma DLL em particular pode falhar se ele encontrar uma versão mais nova ou mais velha da DLL que ele estava esperando. Problemas similares também acontecem com os componentes de software. Com o tempo, se os desenvolvedores do componente fizerem alguma alteração nele e modificarem alguma interface, as aplicações que dependem deste componente podem ser afetadas [EDD1998].

COM resolve este problema ditando que componentes baseados em COM jamais podem ter suas interfaces modificadas após estes terem sido disponibilizados ao mercado. A única maneira de implementar novas funcionalidades em um componente, neste estágio, é publicar uma nova interface. Assim clientes preparados para esta nova interface podem fazer uso de suas funcionalidades, enquanto as aplicações antigas poderão trabalhar com este software através de sua interface antiga [EDD1998].

Segundo [RED1997] existem três características essenciais que definem uma interface:

a) o número de funções suportadas;

b) a declaração de cada função;

c) a ordem em que as funções são declaradas.

[RED1997] complementa dizendo que: “Modificando uma dessas características efetivamente modificará a interface, e porque as interfaces são a única maneira de manipular um objeto COM, a partir do momento que uma interface é exposta para uso por parte dos clientes, ela jamais poderá ser modificada”.

A habilidade de um simples objeto de suportar múltiplas interfaces simultaneamente é uma característica importante do COM. Esta característica permite que um objeto ganhe novas interfaces para suportar novas funcionalidades em versões sucessivas do objeto, enquanto ao mesmo tempo retém completa compatibilidade binária com aplicações que dependem destes objetos [EDD1998]. Em outras palavras, mudar um objeto adicionando novas ou mesmo não relatadas funcionalidades não vão requerer recompilação por parte de qualquer cliente existente.

4.1.2.1 NAVEGAÇÃO ENTRE INTERFACES

Todo objeto COM precisa suportar no mínimo uma interface: *IUnknown*. Quando um cliente inicialmente ganha acesso a um objeto, este cliente receberá no mínimo um ponteiro para *IUnknown* (a mais fundamental Interface), através da qual ele poderá apenas controlar o tempo de vida do objeto e chamar a função *QueryInterface*. *QueryInterface* é a

função básica em COM através da qual um pedaço de software determina quais interfaces são suportadas por um componente [EDD1998].

Para suportar a navegação entre interfaces, toda interface precisa implementar a função *QueryInterface*, a qual possui dois parâmetros, um para especificar o IID da interface desejada, e o outro para receber o ponteiro atual para a interface. Se o objeto COM implementa a interface identificada pelo IID, a chamada para *QueryInterface* será sucedida e retornará um ponteiro para a interface no segundo parâmetro. Supondo que se possui um ponteiro para *IMinhaInterface* no objeto *MeuObjeto* e que o mesmo possua duas *Interfaces*, o Quadro 4 mostra como conseguir um ponteiro para *IMinhaInterface2* fazendo uma chamada a *QueryInterface* através do ponteiro para *IMinhaInterface*.

Quadro 4: Utilização da função QueryInterface

```
HRESULT hr;
IMinhaInterface2* pIMinhaInterface2; //

//pMinhaInterface é um ponteiro para IminhaInterface
hr = pMinhaInterface->QueryInterface(IDD_MinhaInterface2,
      &pIMinhaInterface2);
if(SUCCEEDED(hr))
{
  //pIMinhaInterface2 aponta para a interface MinhaInterface2
}
else
{
  //pIMinhaInterface2 contém um valor nulo
}
```

Fonte: [RED1997]

4.1.2.2 TEMPO DE VIDA DE UM COMPONENTE

Além da função *QueryInterface* a interface *IUnknown* ainda define outras duas funções: *AddRef* e *Release* através das quais é possível controlar o tempo de vida de um objeto. Tipicamente, o cliente de um objeto é responsável por controlar o tempo de vida do mesmo. O cliente cria o objeto quando ele precisa do mesmo, usa o objeto, e destrói o mesmo quando ele não é mais necessário. No entanto, objetos COM podem ter múltiplos clientes independentes um do outro. Para prevenir um cliente de destruir um objeto COM e deixar os

outros com ponteiros inválidos para suas interfaces, ambos, o cliente e o objeto COM, dividem a responsabilidade de controlar o seu tempo de vida, o qual é gerenciado através de um processo chamado contagem de referencia (*reference counting*).

Segundo [RED1997] todo objeto mantém uma variável interna para contagem de referências, conforme a definição do Quadro 5.

Quadro 5: Variável interna para contagem de referências

```
class CQualquerObj : IUnknown
{
private:
  ULONG m_cRef; //Contagem de referencias
  .
  .//Outras variáveis membro
  .
}
```

Fonte: [RED1997]

Quando um componente COM é criado pela primeira vez sua variável de contagem interna é setada para zero. Toda vez que o objeto COM fornece um ponteiro de uma interface – como resultado de uma chamada para *QueryInterface*, por exemplo – é responsabilidade do próprio objeto COM chamar *AddRef* naquela interface [RED1997], conforme Quadro 6.

Quadro 6: Controle interno de contagem de referência

```
HRESULT CQualquerObj::QueryInterface(REFIID iid, LPVOID *ppv)
{
  *ppv = NULL;
  if(IID_IUnknown == iid)
    *ppv = (LPVOID)(IUnknown *)this;
  else if (...)
   .
   . //outras interfaces suportadas
   .
  else
    return E_NOINTERFACE; //interface não suportada

  //incrementa contagem de referencia
  (IUnknown*)*ppv)->AddRef();

  return NOERROR;
}
```

Fonte: [RED1997]

AddRef serve para incrementar o valor da variável interna de contagem de referências em 1 (um). Um exemplo típico desta função pode ser visto no Quadro 7.

Quadro 7: Incremento da variável interna de contagem de referências

```
ULONG CQualquerObj::AddRef(void)
{
  return ++m_cRef;
}
```

Fonte: [RED1997]

Toda vez que um cliente termina de usar uma interface, é responsabilidade do mesmo chamar a função *Release* daquela interface. A função *Release* serve para decrementar o valor da variável interna de contagem de referencias em 1 (um). Quando esta variável chega em 0 (zero) significa que não existe mais nenhum cliente usando esta interface, e então é responsabilidade do objeto COM de se auto-destruir [RED1997]. O Quadro 8 mostra o código padrão para esta função.

Quadro 8: Uso da função Release para controlar o tempo de vida da Interface

```
ULONG CQualquerObj::Release(void)
{
  m_cRef--;
  if(m_cRef == 0)
  {
    delete this;
    return 0;
  }
  return m_cRef;
}
```

Fonte: [RED1997]

4.1.3 TIPOS DE COMPONENTES

Segundo [EDD1998] existem três tipos de componentes – *in-process*, *local* ou *remote* – dependendo da estrutura do seu módulo de código e do seu relacionamento com o processo cliente que o está usando.

Servidores *in-process* são carregados no espaço do processo cliente porque eles são implementados como DLLs. Como se sabe, DLLs são bibliotecas de código que são carregados em tempo de execução (dinamicamente) pelo sistema operacional em função dos programas que precisam chamar funções da DLL, e estas são sempre carregadas no mesmo espaço de memória do processo que a chamou. Isto é importante porque no Windows 9x e Windows NT cada programa (processo) é carregado em seu endereço privado de 32 bits por questões de segurança e estabilidade. Tendo em vista, então, que normalmente não é possível acessar posições de memória além deste espaço privado de memória, as DLLs precisam ser carregadas junto com os processos.

A principal vantagem dos servidores *in-process* é a sua rapidez. Como os objetos são carregados junto com os processos, nenhuma troca de contexto é necessária para acessar os seus serviços. A única desvantagem de servidores *in-process* é que como eles são DLLs e não executáveis (EXE), eles somente podem ser usados no contexto do programa que o chamou e não como uma aplicação independente. Controles ActiveX, por exemplo, são implementados como servidores *in-process*.

Um servidor local roda como um processo separado na mesma máquina do programa cliente. Este tipo de servidor é um executável completo, por isso ele é um processo separado. É significativamente mais lento acessar um servidor local do que um servidor *in-process* porque o sistema operacional precisa alternar entre os processos e copiar os dados que precisam ser transferidos entre as aplicações cliente e servidora. Uma vantagem dos servidores locais é que como eles são arquivos .EXE, podem rodar independentemente de uma aplicação cliente. Um exemplo de um servidor local é o Microsoft Internet Explorer, que pode ser executado pelo usuário para surfar na Internet ou pode ser chamado para servir uma outra aplicação, como o Visual Basic.

Servidores remotos rodam em máquinas separadas conectados em uma rede. Esta funcionalidade pode ser implementada usando *Distributed* COM (DCOM) e não é necessário nenhuma programação especial para conseguir esta funcionalidade. Se não fosse este suporte especial para a comunicação, os programadores teriam que encarar a assustadora tarefa de escrever códigos especializados para cada componente e protocolo de rede ao qual fosse necessário ter suporte.

DCOM fornece o suporte para objetos distribuídos, isto é, ele permite que os desenvolvedores dividam uma simples aplicação em vários componentes diferentes, cada um podendo rodar em um computador diferente. DCOM trabalha usando um *proxy* para interceptar as chamada às interfaces de um objeto e então gerar uma *Remote Procedure Call* (RPC) para a instância real do objeto que está rodando em um processo separado em um outro computador. Um ponto importante é que o cliente faz esta chamada exatamente como ele faria para um objeto *in-process* e que o DCOM trabalha entre estes processos fazendo as chamadas através da rede de forma transparente, isto significa que os objetos não precisam aparecer na rede para serem localizadas em suas máquinas, e a rede pode parecer um único grande computador com um enorme poder de processamento.

4.1.4 COM LIBRARY

Uma das maiores qualidades do COM é que ele isola um desenvolvedor que está usando um componente das diferenças entre servidores do tipo *in-process, local* e *remote*. É visível que mecanismos muito diferentes são usados para chamar um *remote server* em um outro computador do que o que é necessário para carregar uma DLL *in-process*. No entanto,

é objetivo do COM “esconder” estas diferenças dos usuários de um objeto para que todos estes tipos de servidores pareçam iguais [EDD1998].

Portanto é bom esclarecer que COM não é simplesmente uma especificação em papel. COM também envolve algum código a nível de sistema, ou seja, alguma implementação daquilo a que ele se propõe. Esta implementação está contida dentro da *COM Library* e é fornecida através de uma DLL que inclui alguns elementos como descritos na especificação COM:

a) um pequeno número de funções fundamentais na forma de uma API que facilita a criação de aplicações COM, para clientes e servidores. Para os clientes COM fornece funções básicas para instanciar um objeto e para os servidores COM fornece facilidades para a exposição dos objetos do componente;

b) *implementation locator service* (implementação de um serviço localizador) para permitir que o COM determine através de um identificador de classe (CLSID) que servidor implementa aquela classe e onde o servidor está localizado;

c) chamadas *RPC (Remote Procedure Call)* transparentes, isto é, não há diferenciação na forma de chamar um objeto quando ele está rodando em um servidor do tipo *local* ou *remote*;

d) um mecanismo padrão para permitir que uma aplicação controle como a memória é alocada junto com o seu processo.

4.2 INTRANET

A intranet surgiu em meados de 1995 por fornecedores de produtos de rede para se referirem ao uso dentro das empresas privadas de tecnologias projetadas para a comunicação por computadores entre empresa. Com pouco tempo de existência, a intranet já ganhava mais rapidamente adeptos que sua irmã mais velha, a Internet, esses novos adeptos são usuários especiais: as Corporações. Apesar da internet interligar milhões de computadores

em todo o mundo, com acesso fácil e imediato, as informações se apresentarem de forma gráfica, agradável, permitindo a qualquer pessoa ou empresa poder buscar e fornecer informações, havia a necessidade de oferecer as empresas soluções aos problemas de distribuição de documentos e dados, pois em muitos casos as soluções patenteadas não conseguiam atingir o objetivo de proporcionar um acesso flexível e integrado as informações. Outra necessidade era em aprimorar a capacidade de uso dos softwares e dinamizar o treinamento de usuários através de padrões comuns. A intranet estaria oferecendo todas essas vantagens oferecendo ainda mais uma segurança às suas redes particulares de informação com a estrutura de comunicações da internet. Essa estrutura de comunicações que é utilizada pela intranet consiste nos mesmos padrões de comunicação de dados da internet pública. Os padrões que a intranet se baseia são o TCP/IP (*Transport Control Protocol*/*Internetworking Protocol*). O TCP/IP é um protocolo de rede que permite um computador enderece e envie dados de forma confiável a outro computador. O IP trata do endereçamento e o TCP é de garantir a transmissão, independente da possível ocorrência de falhas na rede intermediária.

A internet surgiu com finalidades militares e posteriormente liberada para estudo, desenvolvimento pelas Universidades e outros órgãos. O surgimento da intranet, muitos anos mais tarde, surgiu com a necessidade de facilitar a vida das corporações oferecendo algumas utilidades a mais que a internet, sendo utilizada principalmente como uma 'ferramenta' para disponibilizar a reapresentação de uma mesma realidade para muitas pessoas. E é exatamente por isso que ela se estabelece como uma explosão de remodelamento empresarial e se transforma tão rapidamente, de um sistema de integração pública, a uma estratégia de comunicação corporativa.

4.3 HYPERTEXT MARKUP LANGUAGE – HTML

O *HyperText Markup Language* (HTML) é uma linguagem de formatação usada para construir páginas na internet/intranet. Possui algumas regras de sintaxe que devem ser seguidas para obter o efeito desejado. Em uma página HTML, podem existir textos, imagens, itens multimídia e hipertextos, que são ligações para outras páginas ou imagens [UNI2000].

As páginas criadas em HTML são arquivos de texto simples, isto é, não contém informações específicas de determinado programa ou plataforma e podem ser lidas por

qualquer programa editor de textos [LEM1998]. Um arquivo HTML contém dois componentes:

a) o texto propriamente dito;

b) as *tags* HTML, que indicam elementos da página, a estrutura, a formatação e vínculos de hipertexto para outras páginas.

Geralmente as tags tem o seguinte formato: <NomeTag>texto</NomeTag>. As tags HTML normalmente têm uma tag de abertura e outra de fechamento, que delimitam o texto a que se referem. A tag de abertura ativa um recurso e a tag de fechamento o desativa. O nome das tags de fechamento são precedidos por uma barra (/). Mas nem todas elas tem uma abertura e um fechamento, algumas são unilaterais.

Quando uma página em HTML é analisada por um *browser*, toda a formatação feita manualmente, como espaços extras, tabulações e retornos, são ignoradas. O único elemento capaz de formatar uma página em HTML é a tag. Um pequeno exemplo de página HTML pode ser visto no Quadro 9.

A HTML define três tags que são usadas para descrever as estrutura geral da página e que oferecem algumas informações simples de cabeçalho. Essas três tags identificam a página para os *browsers* ou ferramentas HTML e fornecem, ainda, informações sobre a página antes de carregá-la inteiramente. As tags de estrutura da página não afetam o aspecto visual que ela terá quando for apresentado. Essas tags são:

a) <HTML>: é a primeira tag de estrutura de toda página HTML, que indica que o conteúdo do arquivo encontra-se codificado em HTML. Todo o texto e comandos contidos na página HTML deverão estar delimitados pelas tags HTML de abertura e fechamento;

b) <HEAD>: geralmente há poucas tags nesta parte, onde é usada a tag <TITLE>, que define o título da página;

a) <BODY>: o restante da página HTML é delimitado por esta tag. Marca o início do documento propriamente dito.

Quadro 9: Página HTML

```
<HTML>
<HEAD>
<TITLE>Titulo da pagina</TITLE>
</HEAD>
<BODY>
corpo do documento
</BODY>
</HTML>
```

Fonte: [UNI2000]

4.4 ACTIVE SERVER PAGES – ASP

Cative Server Pajés (ASP) é uma tecnologia de *scripts* que rodam no servidor, que pode ser usada para criar aplicações *web* dinâmicas e interativas. Uma página ASP é uma página *HyperText Markup Language* (HTML) com extensão .asp que contém *server-side scripts* que são processados pelo servidor web antes de serem enviadas ao *browser* do cliente. Todo o código de programação existente em páginas ASP é executado no servidor, e este retorna ao cliente somente respostas em HTML padrão, o que faz com que aplicações ASP possam ser acessadas por qualque *browser* existente no mercado. Uma aplicação ASP pode ainda conter linhas de *client-side script*, que são executados na estação cliente [UNI2000].

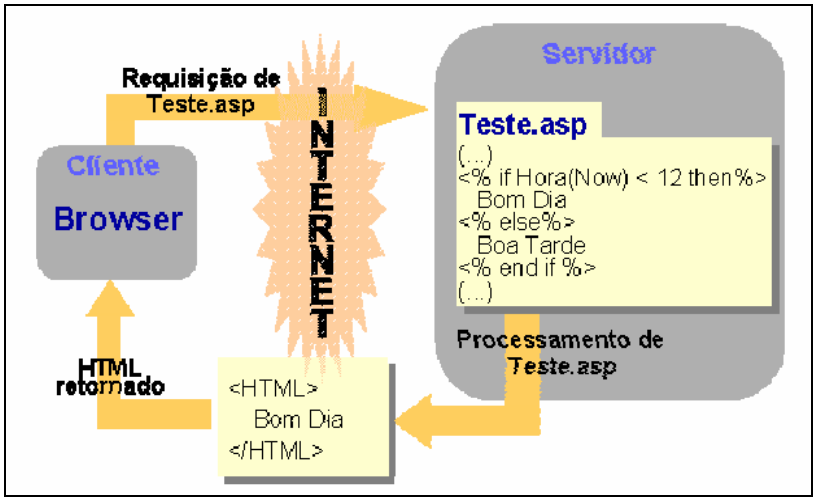
Os c*lient-side scripts* são códigos de programa que são processados pelo *browser* do cliente. Geralmente em aplicações voltadas à internet ou intranet, o código que é executado no cliente trata apenas pequenas consistências de telas e validações de entrada de dados. Em se tratando de páginas web, os *client-side scripts* terão de ser processados pelo *browser*. O maior problema de se utilizar este tipo de artifício em uma aplicação é a incompatibilidade de interpretação da linguagem entre os *browsers*. O Microsoft Internet Explorer, por exemplo, é capaz de interpretar o VBScript, porém o Netscape Navigator não o faz sem o auxílio de um *plug-in*, desenvolvido por terceiros. Há ainda o problema de versões muito antigas de navegadores, que não suportam nenhum tipo de *script*.

Em grande parte das situações, não é possível exigir que o usuário final disponha de determinado produto para acessar a aplicação. Portanto, é importante pesar todos esses fatores ao planejar uma aplicação com *client-side scripts*. A linguagem de *script* mais

adequada para se usar do lado cliente é o JavaScript, devido a sua compatibilidade com a maioria dos *browsers* atuais.

Server-side scripts são códigos de programa que são processados pelo servidor. Devido a este fato, não é necessário preocupar-se com a linguagem que o código é criado: o servidor é quem se encarrega de interpretá-lo e devolver a resposta em HTML padrão para o cliente, como visto na Figura 1. Em páginas ASP são esse códigos os maiores responsáveis pelos resultados apresentados, sendo sua linguagem de *script* padrão o VBScript.

Figura 1: Requisição de uma página ASP

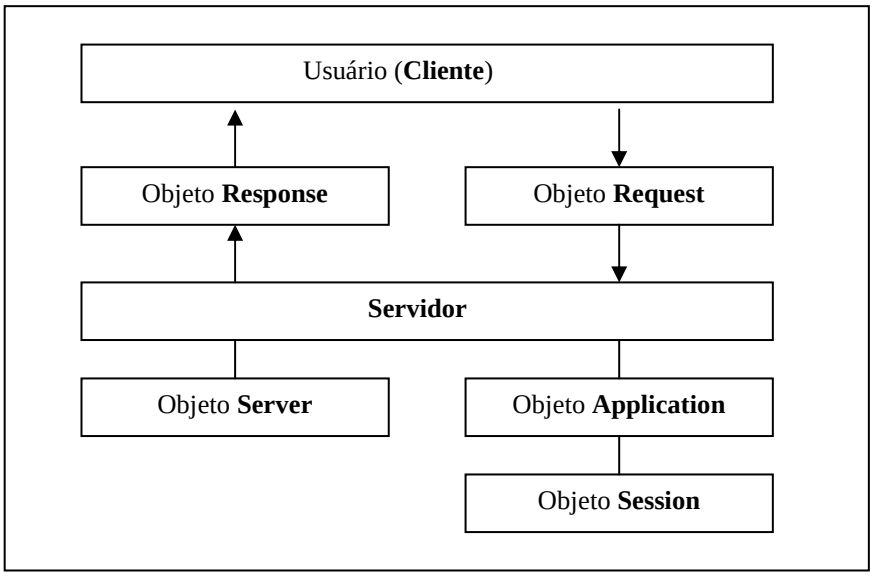


Fonte: [UNI2000]

4.4.1 OBJETOS DO ASP

O ASP possui alguns objetos básicos que operam as funcionalidades de uma aplicação. Cada objeto possui seus próprios métodos e eventos. A hierarquia dos objetos ASP pode ser vista na Figura 2.

Figura 2: Hierarquia dos objetos ASP



Fonte: [MAR1999]

4.4.1.1 OBJETO APPLICATION

Pode-se entender como aplicação todo o conjunto de páginas ASP e HTML que formam o programa como um todo. Pode-se, com este objeto, disponibilizar determinadas informações que serão acessíveis em qualquer parte do programa e por qualquer usuário.

Para declarar uma variável *Application* usa-se a seguinte sintaxe:

Application("nomvar") = conteúdo

Como as variáveis do tipo Application são compartilhadas e podem ser alteradas em qualquer página ASP, deve-se prevenir eventuais problemas com alterações simultâneas de valores, mantendo, deste modo, os dados sempre consistentes. Para isso, usa-se os métodos *lock* e *unlock*. O objetivo do método *lock* é deixar disponíveis as variáveis apenas para um usuário, impedindo o acesso aos demais usuários. O método *unlock* desbloqueia o acesso às variáveis que foram bloqueadas pelo método *lock*.

No Quadro 10, o método *lock* foi utilizado para que o número de visitantes seja sempre exibido de forma correta, não ocorrendo nenhum problema no caso de haverem duas tentativas simultâneas de modificação no valor da variável.

Quadro 10: Objeto *Application*

```
<HTML>
<BODY>
<%
Application.Lock
Application("visitantes") =
   Application("visitantes") + 1
Application.Unlock
%>
Esta página foi acessada
<%=Application("visitantes")%>
</BODY>
</HTML>
```

Fonte: [UNI2000]

4.4.1.2 OBJETO *SERVER*

O objeto *Server* é capaz de interagir com o serviço HTTP, criando uma interface programável de seus métodos e propriedades. Outra função deste objeto é instanciar componentes no servidor. Os principais métodos deste objeto são:

a) *CreateObject*: este método cria uma instância de um componente no servidor. A capacidade de utilizar componentes amplia as potencialidades de uma aplicação ASP. Por padrão, os objetos criados por este método tem o escopo da página, ou seja, eles são automaticamente destruídos pelo servidor assim que ele termina de processar a página ASP. Sua sintaxe é: Server.CreateObjetct("nomecomponente.nomeinterface"), onde *nomecomponente* é o nome da biblioteca do componente e *nomeinterface* é a interface deste componente da qual se deseja criar o objeto;

b) *Execute*: este método chama um arquivo ASP e o processa como se fosse parte do *script* ASP que o chamou. É semelhante a se fazer uma chamada de procedimento em muitas linguagens de programação. Sintaxe: Server.Execute(arquivo)

4.4.1.3 OBJETO *SESSION*

Entende-se por sessão o tempo que um usuário utiliza uma aplicação. Cada vez que a aplicação é acessada por um usuário, uma sessão no servidor é aberta para ele. Quando a aplicação termina, a sessão é finalizada. Dessa forma, informações disponibilizadas no escopo da sessão estarão disponíveis durante toda a aplicação para um determinado usuário.

Este objeto é baseado em *cookies*, portando será acessível somente aos *browsers* que os suportam e estejam habilitados a os aceitarem. Os *cookies* são arquivos que são gravados no computador cliente onde são armazenadas informações, como identificação do usuário ou preferências, que são enviadas ao computador servidor. Para declarar uma variável Session usa-se a seguinte sintaxe: Session("nomvar") = conteúdo

4.4.1.4 OBJETO RESPONSE

O Objeto *Response* controla os dados que serão enviados para o cliente. Estes dados podem ser HTML, *cookies*, valores de variáveis, etc. Suas principais coleções e métodos são:

a) coleção *Cookies*: É através dessa coleção que se pode enviar *cookies* para um cliente. Com *cookies* pode-se armazenar temporariamente valores que são gravados em arquivos texto pelo *browser* cliente. As linhas de código que gravam o cookie devem sempre vir antes de qualquer tag HTML, pois não há como gravar informações em um cookie depois de qualquer código HTML ser enviado ao cliente.
Sintaxe: Response.Cookies("nomecookie")("nomedachave") = conteúdo, onde *nomecookie* é o nome do arquivo texto que será gravado no cliente e *nomedachave* o nome do campo a ser armazenado;

Quadro 11: *Cookies*

```
<%
Response.Cookies("Teste")("Campo1") = "Primeiro
   valor"
Response.Cookies("Teste")("Campo2") = "Segundo
   valor"
%>
<HTML>
<BODY>
Teste
</BODY>
</HTML>
```

Fonte: [UNI2000]

b) método *Write*: Este método é utilizado para enviar texto à página HTML. Sintaxe: Response.Write conteúdo. Pode-se utilizar como conteúdo valores de variáveis, funções ou mesmo textos;

Quadro 12: Método *Write*

```
<HTML>
<%Session("teste") = "Teste"%>
<BODY>
<%
Response.Write ("Hello world")
//envia o texto “Hello world” à
//página HTML

Response.Write (Session("teste"))
//envia o conteúdo da variável
//de sessão “teste” à página HTML
%>
</BODY>
</HTML>
```

Fonte: [UNI2000]

c) método *Redirect*: Utilizado para redirecionar o *browser* para outra URL. Este método deve ser colocar antes de qualquer tag <HTML> ser enviada ao cliente. Sintaxe: Response.Redirect URL

4.4.1.5 OBJETO *REQUEST*

O objeto *Request* é capaz de receber informações do cliente, como, por exemplo, ler *cookies*, receber dados digitados em formulários HTML. Suas coleções são:

a) *Cookies*: Esta coleção é responsável por ler as informações armazenadas nos cookies existentes nos clientes. Sintaxe: Request.Cookies("nomecookie")("nomechave");

b) *Form*: Permite que se obtenha dados digitados em formulários HTML enviados pelo método HTTP *Post*. Sintaxe: Request.Form("nomecampo");

c) *QueryString*: É utilizada para se obter informações vindas da string de pesquisa HTTP. Esta string se encontra após o ponto de interrogação (?) na linha da URL, como, por exemplo, em http://www.empresa.com.br?cliente=1. Request.QueryString("nomevar");

d) *ServerVariables*: Esta coleção retorna valores de variáveis de ambiente predeterminadas. Algumas dessas variáveis são:

- LOGON_USER: A conta de usuário com a qual o cliente está logado;

- REMOTE_ADDR: Endereço IP do cliente que fez a requisição;

- SERVER_NAME: O nome, alias DNS ou endereço IP do servidor.

4.4.1.6 DIRETIVA *INCLUDE*

Um outro recurso do ASP é o uso do *include* no lado do servidor. Este recurso permite incluir arquivos dentro de uma página ASP durante a execução. Isto é extremamente útil para a criação de funções globais, cabeçalhos, rodapés ou outros elementos que precisam ser reutilizados em várias páginas. Quando é necessário alterar alguma dessas funções, pode-se fazê-lo de uma só vez, sendo essa alteração refletida automaticamente em todas as páginas que fazem referência a este arquivo. Sintaxe: <!-- #include file=”nomedoarquivo” -->

Quadro 13: Diretiva *Include*

```
<HTML>
<BODY>
Exemplo de utilizacao da diretiva
include
<!--#include file="rodape.htm"-->
</BODY>
</HTML>
```

Fonte: [UNI2000]

4.5 OPEN DATABASE CONNECTIVITY – ODBC

O *Open DataBase Connectivity* (ODBC) é um meio utilizado para acessar um banco de dados em ambiente Windows. Utilizando o ODBC, o desenvolvedor não precisa se preocupar com as particularidades dos bancos de dados que irá acessar e trabalhar. O ODBC é uma interface que permite à aplicações acessarem qualquer banco de dados que possua um driver ODBC. O ODBC fornece uma API que permite à aplicação ser independente do sistema de gerenciamento de banco de dados (DBMS). Quando um banco de dados é acessado através do ODBC, este banco deve estar registrado como uma origem de dados ODBC (*DSN - Data Source Name*). Registrando o banco como uma origem de dados, a aplicação apenas precisa saber o nome desta origem de dados. A localização ou o tipo do banco de dados não faz diferença para a aplicação.

O ODBC possui os seguintes componentes:

a) ODBC API: uma biblioteca de funções, um conjunto códigos de erros e uma linguagem SQL padrão para acessar os dados do DBMS;

b) ODBC *Driver Manager*: uma DLL (ODBC32.DLL) que carrega e gerencia os drivers ODBC dos bancos de dados que a aplicação utilizar;

c) ODBC *Database Drivers*: uma ou mais DLL’s que processam as chamadas ODBC para um banco de dados específico;

d) ODBC *Cursor Library*: uma DLL (ODBCCR32.DLL) que atua enre o ODBC Driver Manager e os drivers do banco de dados que trata da rolagem dos dados;

e) ODBC *Administrator*: uma ferramenta usada para configurar um DBMS para torná-lo disponível como *data source* para uma aplicação.

5 PROTÓTIPO

Como base para o trabalho proposto, observou-se o problema atual de uma empresa na região de Brusque. Esta empresa trabalha no ramo metalúrgico, com a fabricação de máquinas e equipamentos sob encomenda para a indústria alimentícia e de pescado.

5.1 LEVANTAMENTO DE INFORMAÇÕES

Observou-se que a empresa tem alguns processos e controles automatizados, mas mantém um controle manual sobre as horas gastas nas ordens de produção. Este controle é necessário para o cálculo da mão-de-obra empregada em cada projeto executado, visto que, por serem produtos sob encomenda, demandam tempos diferentes na manufatura. Alguns problemas observados com esse processo são a falta de legibilidade das anotações pelos funcionários, o tempo gasto na digitação dos dados e possíveis erros nos cálculos.

Para cada projeto solicitado, o sistema atual emite uma ordem de produção, que é encaminhado ao setor de produção, que é então repassada aos funcionários. Cada funcionário mantém uma ficha de controle individual, mostrada na Figura 3, onde cada um marca o tempo empregado em cada tarefa/ordem, e respectivo centro de custo, que pode ser, funilaria, pintura, acabamento, elétrico, entre outros. Periodicamente esta ficha é encaminhada a um responsável por digitar essas informações numa planilha eletrônica, para então se obter o total de horas por projeto, conforme Figura 4.

Figura 3: Ficha de Controle Individual

Ficha de Produção – Operador: Francisco

Custo CC: C-Corte F-Funilaria E-Estrutura U-Usinagem P-Pintura A-Acabamento

Dia/Mês	OP	CC	Horas Gastas	Dia/Mês	OP	CC	Horas Gastas
13/07	1587	F	3:30	15/07	1586	F	1:00
13/07	1556	C	1:00	15/07	1593	C	0:30
13/07	1556	F	3:30	16/07	1303	C	0:45
14/07	1634	C	7:00	16/07	1430	C	0:45
14/07	1553	C	1:00	16/07	1303	F	0:45
15/07	1553	F	2:45	17/07	1430	C	0:45
15/07	1553	F	2:15	19/07	1565	F	5:00
15/07	1565	C	1:00	19/07	1651	F	3:00

Figura 4: Tabela de horas de produção

Custo Horas Produção					
Produto: Separador de gelo c/ esteira para seleção				Des. Nr. 01.20.12-0	
Corte	Estrutura	Funilaria	Usinagem	Acabamento	Pintura
20,50	10,50	138,00	7,75	4,25	5,75
6,50		22,50	0,25	2,00	
9,25		63,25	0,75	32,00	
		6,75	28,50	4,50	
			1,50		
36,25	**10,50**	**230,50**	**38,75**	**42,75**	**5,75**

TOTAL GERAL: 364,50 h

5.2 ESPECIFICAÇÃO

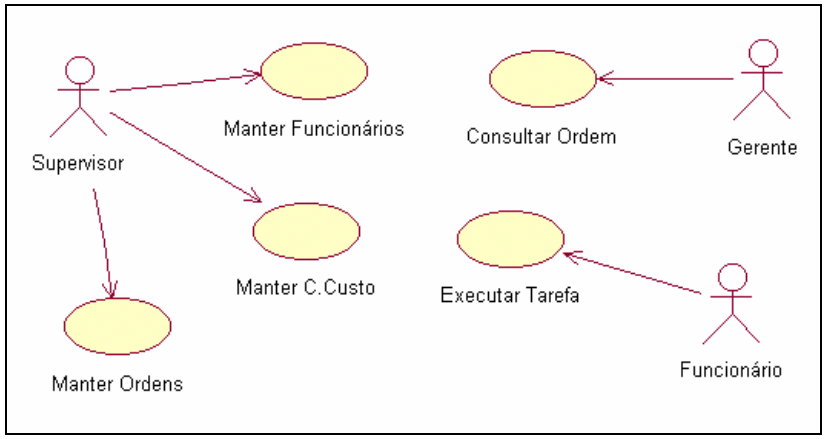
O protótipo desenvolvido propõe automatizar este processo de controle de horas, acabando com os problemas citados. A parte cliente do protótipo foi desenvolvido em ASP, onde será acessado via *browser*, através da intranet. As páginas ASP acessam um componente COM, desenvolvido em Visual C++, que será instalado no servidor.

Para a utilização do sistema na área de produção, sugere-se utilizar um microcomputador rodando um *browser*, com um leitor de código de barras, onde a entrada do código do funcionário será feita através do crachá do mesmo, e do número da ordem pela ficha de produção emitida pelo sistema existente.

5.2.1 DIAGRAMA DE CASOS DE USO

Conforme a notação da UML [FUR1994], e utilizando-se a ferramenta Rational Rose, foi desenvolvido o diagrama de casos de uso conforme a Figura 5, onde destacam-se cinco casos de uso principais do sistema.

Figura 5: Diagrama de casos de uso



Os casos de uso definidos para o sistema são:

a) **manter funcionários**: inclusão, alteração e exclusão no cadastro de funcionários;

b) **manter centros de custo**: inclusão, alteração e exclusão no cadastro de centros de custo;

c) **manter ordens de produção**: inclusão, alteração e exclusão de ordens de produção;

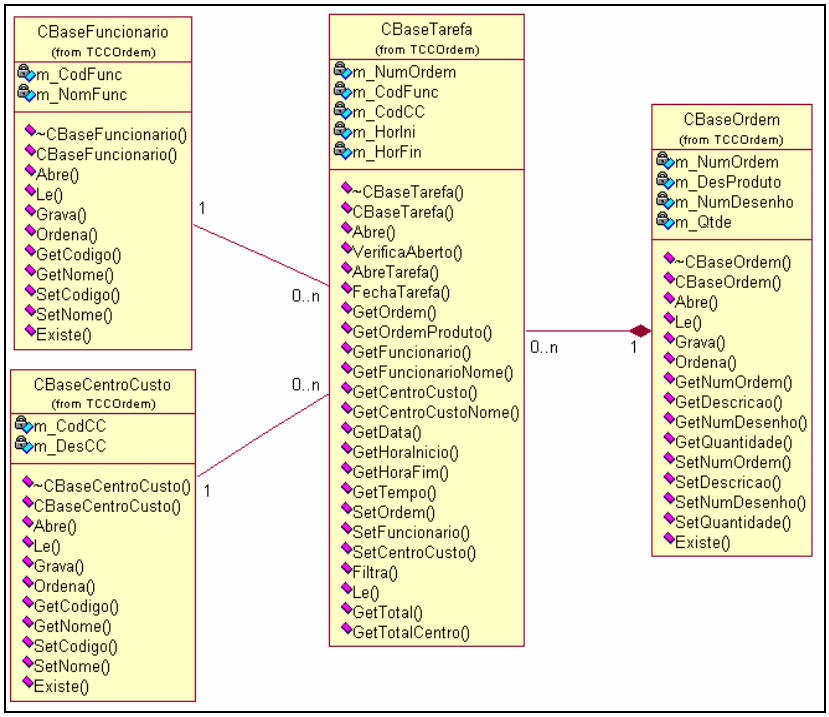
d) **executar tarefa**: quando um funcionário inicia uma tarefa, digita seu código ou passa seu cartão com código de barras, o código do centro de custo e o número da ordem para abrir a tarefa. Quando termina a tarefa, digita apenas o seu código para fechar a tarefa;

e) **consultar ordem**: gerente obtém totais de horas gastas por ordem de produção e centro de custo.

5.2.2 DIAGRAMA DE CLASSES

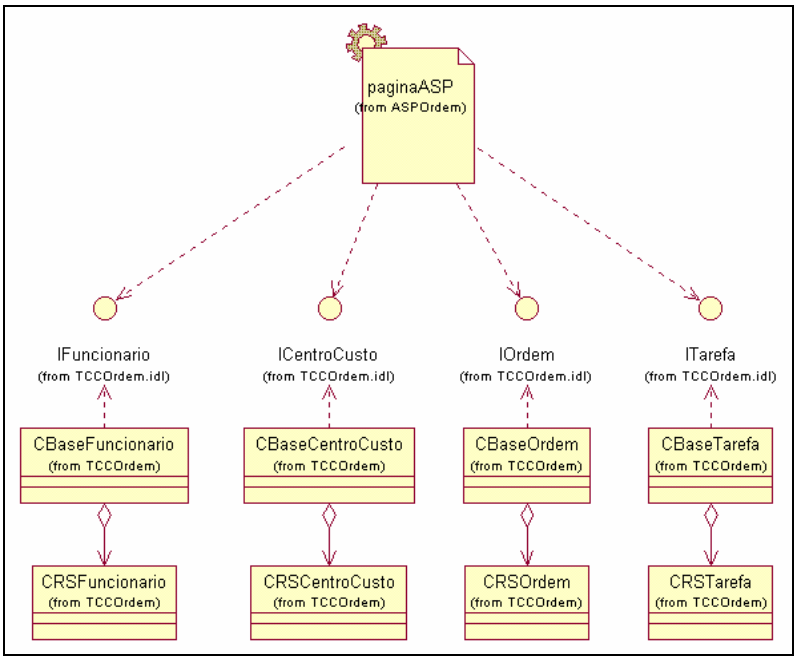
Inicialmente, na fase de análise, identificou-se as classes principais do sistema, seus atributos componentes e seus relacionamentos, conforme visto na Figura 6.

Figura 6: Diagrama de classes



Na fase de projeto, foram acrescentadas as classes para prover a infra-estrutura técnica, como as interfaces do componente e as classes de acesso a banco de dados (Figura 7). Também foi criada uma classe genérica chamada *paginaASP*, que representa a parte da aplicação desenvolvida em ASP, a qual acessará as interfaces do componente.

Figura 7: Diagrama de classes (projeto)

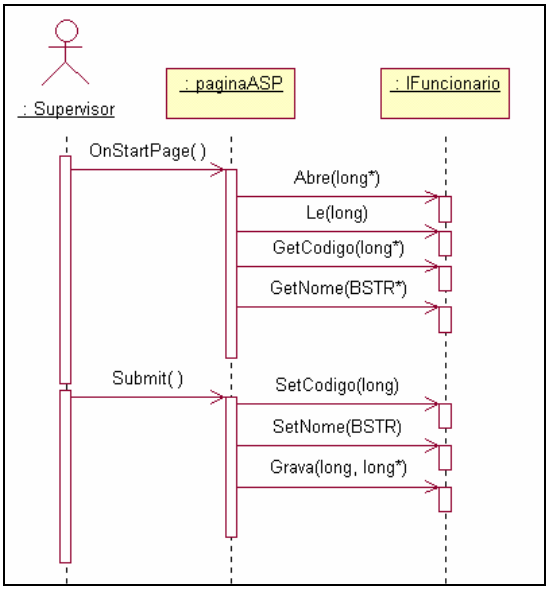


5.2.3 DIAGRAMA DE SEQÜÊNCIA

Nos diagramas de sequência pode-se observar o tempo de execução de cada objeto e verificar as mensagens que são trocadas entre os objetos. As principais seqüências encontradas no sistema são:

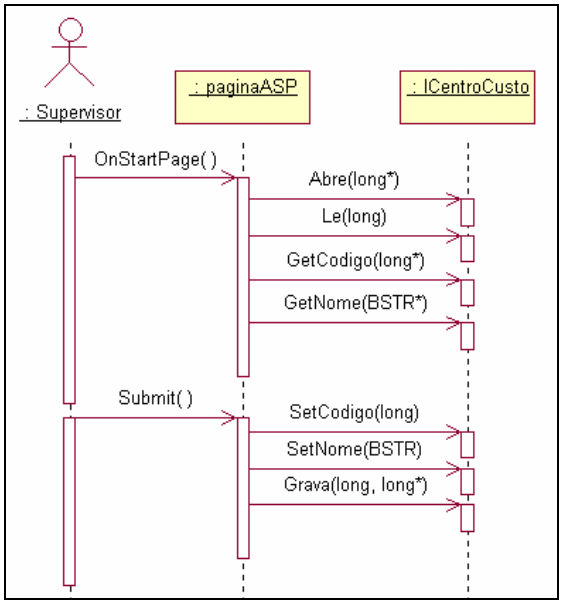
a) manutenção de funcionários (Figura 8). Quando o supervisor acessa a tela de manutenção de funcionários, é disparado o método *OnStartPage* da página ASP. Esta, por sua vez, acessa a interface *IFuncionario* e busca os dados dos funcionários para serem exibidos na tela. Quando o supervisor altera ou inclui um funcionário, é disparado o método *Submit* da página ASP, a qual envia os dados e chama o método *Grava* da interface *IFuncionario*;

Figura 8: Manutenção de funcionários



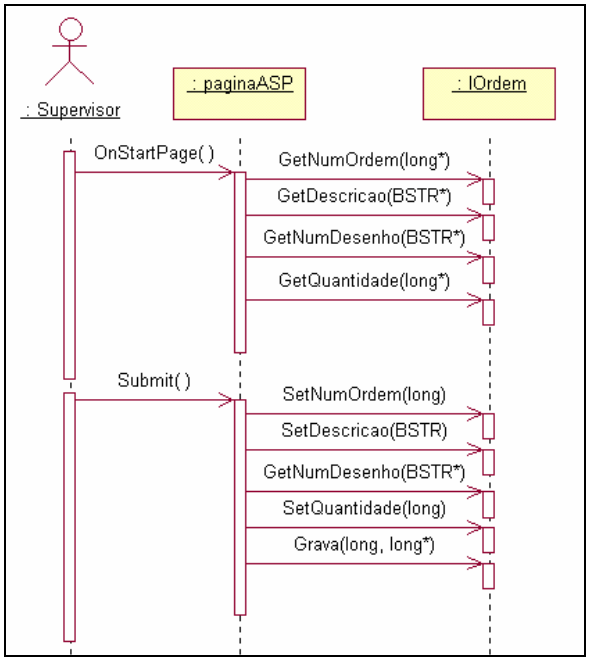
b) manutenção de centros de custo (Figura 9). Quando o supervisor acessa a tela de manutenção de centros de custo, é disparado o método *OnStartPage* da página ASP. Esta, por sua vez, acessa a interface *ICentroCusto* e busca os dados dos centros de custo para serem exibidos na tela. Quando o supervisor altera ou inclui um centro de custo, é disparado o método *Submit* da página ASP, a qual envia os dados e chama o método *Grava* da interface *ICentroCusto*;

Figura 9: Manutenção de centro de custo



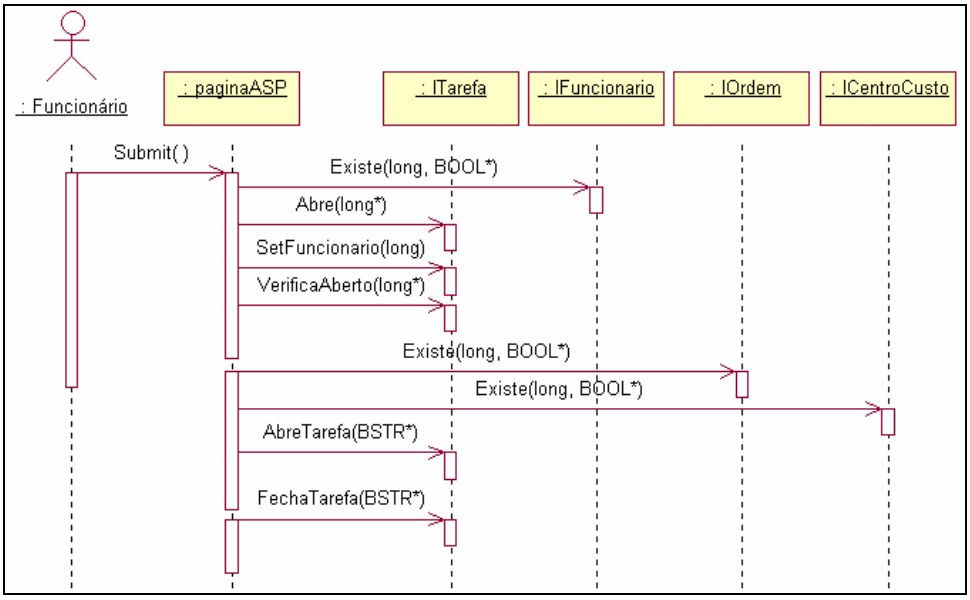
c) manutenção de ordens de produção (Figura 10). Quando o supervisor acessa a tela de manutenção de ordens de produção, é disparado o método *OnStartPage* da página ASP. Esta, por sua vez, acessa a interface *IOrdem* e busca os dados das ordens para serem exibidos na tela. Quando o supervisor altera ou inclui uma ordem de produção, é disparado o método *Submit* da página ASP, a qual envia os dados e chama o método *Grava* da interface *IOrdem*;

Figura 10: Manutenção de ordens de produção



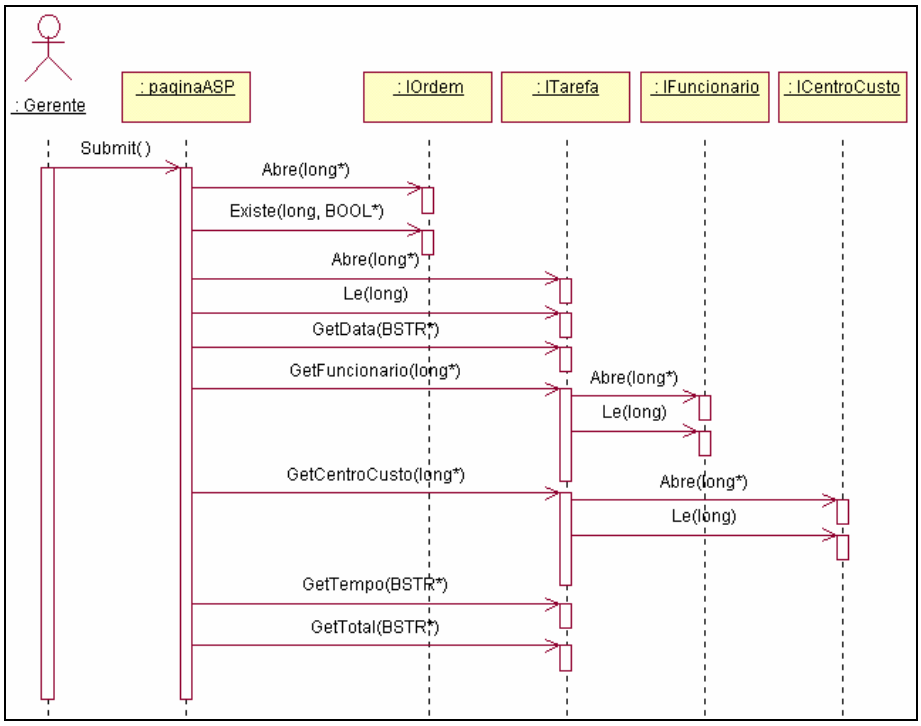
d) executar tarefa (Figura 11). Ao solicitar abertura ou fechamento de tarefa, o funcionário digita seu código na tela de abertura/fechamento de tarefa. A página ASP acessa a interface *IFuncionario* para verificar se o código do funcionário é válido, e em seguida chama o método *VerificaAberto* da interface *ITarefa*, para verificar se já existe uma tarefa em aberto para aquele funcionário. Se há tarefa aberta, é chamado o método *Fecha*, senão, é verificado se o código do centro de custo e o número da ordem de produção existem para, então, chamar o método *Abre*;

Figura 11: Executar tarefa



e) consultar ordem (Figura 12). Quando o gerente solicita a consulta, a página ASP verifica se o número da ordem é valido. Em seguida, busca os dados das tarefas (Data, Tempo) e os dados dos funcionários e centros de custo de cada tarefa. No final, é solicitado o tempo total gasto na ordem de produção.

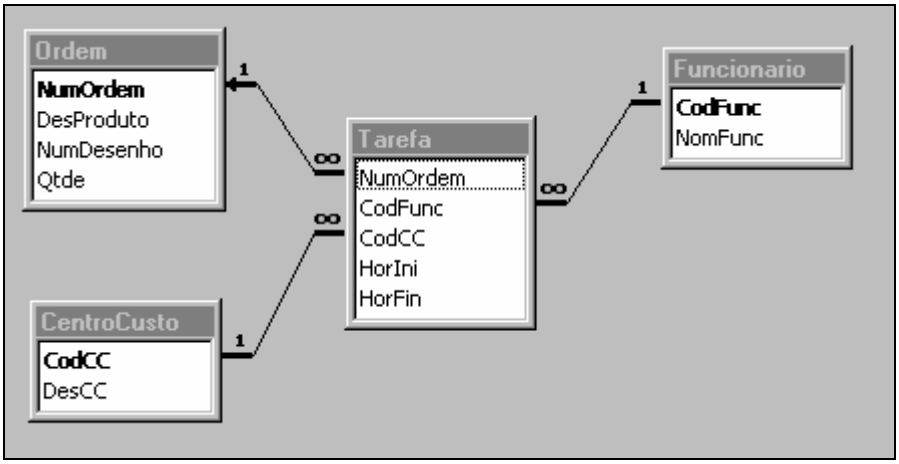
Figura 12: Consultar ordem



5.3 BANCO DE DADOS

Para armazenagem dos dados utilizou-se o banco de dados Microsoft Access. O sistema utiliza quatro tabelas, conforme descrito na Figura 13. As tabelas são acessadas pelo componente COM através de *Open DataBase Connectivity* (ODBC), utilizando classes derivadas da classe CRecordSet da *Microsoft Foundation Classes* (MFC). Um objeto CRecordSet representa um conjunto de registros selecionados de uma base de dados. Com o CRecordSet pode-se navegar entre os registros, atualizar, filtrar o conjunto, ordenar, entre outras operações.

Figura 13: Tabelas do aplicativo



5.4 IMPLEMENTAÇÃO

A seguir, são apresentados alguns detalhes da implementação. O Quadro 14 mostra a especificação das *interfaces* do componente e seus respectivos métodos. O Quadro 15 mostra um exemplo de acesso ao componente a partir de uma página ASP contida no sistema.

Quadro 14: *Interfaces* do componente

```
interface IOrdem : IDispatch {
  [id(1)] HRESULT Abre([out,retval] long* lQtde);
  [id(2)] HRESULT Le([in] long lPos);
  [id(3)] HRESULT Grava([in] long lTipo, [out,retval] long* lRetorno);
  [id(4)] HRESULT Existe([in] long lNumOrdem, [out,retval] BOOL* bExiste);
  [id(5)] HRESULT Ordena([in] long lCampo);
  [id(6)] HRESULT GetNumOrdem([out, retval] long* lNumOrdem);
  [id(7)] HRESULT GetDescricao([out,retval] BSTR* bsDescricao);
  [id(8)] HRESULT GetNumDesenho([out,retval] BSTR* bsNrDes);
  [id(9)] HRESULT GetQuantidade([out,retval] long* lQuantidade);
  [id(10] HRESULT SetNumOrdem([in] long lNumOrdem);
  [id(11] HRESULT SetDescricao([in] BSTR bsDescricao);
  [id(12] HRESULT SetNumDesenho([in] BSTR bsNrDes);
  [id(13] HRESULT SetQuantidade([in] long lQuantidade);
};
interface IFuncionario : IDispatch {
  [id(1)] HRESULT Abre([out,retval] long* lQtde);
  [id(2)] HRESULT Le([in] long lPos);
  [id(3)] HRESULT Grava([in] long lTipo, [out,retval] long* lRetorno);
  [id(4)] HRESULT Existe([in] long lCodigo, [out,retval] BOOL* bExiste);
  [id(5)] HRESULT Ordena([in] long lCampo);
  [id(6)] HRESULT GetCodigo([out, retval] long* lCodigo);
  [id(7)] HRESULT GetNome([out,retval] BSTR* bsNome);
  [id(8)] HRESULT SetCodigo([in] long lCodigo);
  [id(9)] HRESULT SetNome([in] BSTR bsNome);
};
interface ICentroCusto : IDispatch {
  [id(1)] HRESULT Abre([out,retval] long* lQtde);
  [id(2)] HRESULT Le([in] long lPos);
  [id(3)] HRESULT Grava([in] long lTipo, [out,retval] long* lRetorno);
  [id(4)] HRESULT Existe([in] long lCodigo, [out,retval] BOOL* bExiste);
  [id(5)] HRESULT Ordena([in] long lCampo);
  [id(6)] HRESULT GetCodigo([out, retval] long* lCodigo);
  [id(7)] HRESULT GetNome([out,retval] BSTR* bsNome);
  [id(8)] HRESULT SetCodigo([in] long lCodigo);
  [id(9)] HRESULT SetNome([in] BSTR bsNome);
};
interface ITarefa : IDispatch {
  [id(1)] HRESULT Abre([out,retval] long *lQtde);
  [id(2)] HRESULT Le([in] long lPos);
  [id(3)] HRESULT Filtra([in] BSTR bsFiltro, [out,retval] long* lQtde);
  [id(4)] HRESULT VerificaAberto([out,retval] long *lRetorno);
  [id(5)] HRESULT AbreTarefa([out,retval] BSTR *bsHora);
  [id(6)] HRESULT FechaTarefa([out,retval] BSTR *bsHora);
  [id(7)] HRESULT GetOrdem([out, retval] long *lCodigo);
  [id(8)] HRESULT GetOrdemProduto([out, retval] BSTR *bsNome);
  [id(9)] HRESULT GetFuncionario([out, retval] long *lCodigo);
  [id(10)] HRESULT GetFuncionarioNome([out, retval] BSTR *bsNome);
  [id(11)] HRESULT GetCentroCusto([out, retval] long *lCodigo);
  [id(12)] HRESULT GetCentroCustoNome([out, retval] BSTR *bsNome);
  [id(13)] HRESULT GetData([out,retval] BSTR *bsData);
  [id(14)] HRESULT GetHoraInicio([out,retval] BSTR *bsHora);
  [id(15)] HRESULT GetHoraFim([out,retval] BSTR *bsHora);
  [id(16)] HRESULT GetTempo([out,retval] BSTR *bsHora);
  [id(17)] HRESULT SetOrdem([in] long lCodigo);
  [id(18)] HRESULT SetFuncionario([in] long lCodigo);
  [id(19)] HRESULT SetCentroCusto([in] long lCodigo);
  [id(20)] HRESULT GetTotal([out,retval] BSTR *bsHoras);
  [id(21)] HRESULT GetTotalCentro([out,retval] BSTR *bsHoras);
};
```

Quadro 15: Exemplo de acesso ao componente

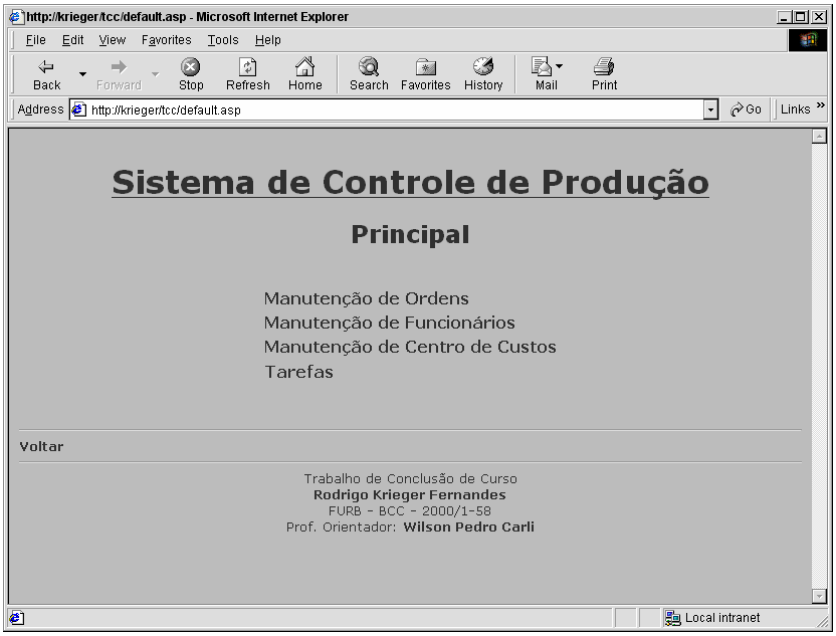
```
<%
   //Busca dados do formulario
   lNumOrdem = Request.Form("NumOrdem")
   sProduto = Request.Form("Produto")
   iQuanti = Request.Form("Quanti")
   sDesNR = Request.Form("DesNR")
   iTipo = Request.Form("Tipo")

   var sResultado
   //Instancia objeto 'Ordem' do componente 'TCCOrdem'
   obj = new ActiveXObject("TCCOrdem.Ordem")
   obj.Abre()   //Abre banco de dados
   obj.SetNumOrdem(lNumOrdem)

   //iTipo -> 1 - Alteracao / 0 - Exclusao
   if (iTipo == 1)
   {
      //alteracao
      obj.SetDescricao(sProduto)
      obj.SetNumDesenho(sDesNR)
      obj.SetQuantidade(iQuanti)
      if (obj.Grava(1))
         sResultado = "Alterado com sucesso"
      else
         sResultado = "Erro"
   }
   else
   {
      iResult =
      if (obj.Grava(2))
         sResultado = "Excluido com sucesso"
      else
         sResultado = "Erro"
   }
%>
```

5.5 DESCRIÇÃO DAS TELAS

Em seguida, uma breve descrição das telas do sistema.

A Figura 14 mostra a tela principal do protótipo. A partir dela se tem acesso a processos de manutenção de ordens de produção, manutenção de funcionários, manutenção de centros de custo e o controle de tarefas.

Figura 14: Tela Principal



A Figura 15 mostra a lista de ordens de produção. A partir dessa lista o usuário seleciona uma ordem e poderá alterar, excluir ou consultar o totais de horas. A partir dela também pode-se incluir uma ordem nova.

Figura 15: Lista de ordens de produção

Na Figura 16 são feitas as inclusões, alterações e exclusões de ordens de produção. Os campos disponíveis são o número da ordem, a descrição do produto, a quantidade e o número do desenho. Este último campo é apenas informativo e serve para cadastrar o número do desenho na base de dados de projetos.

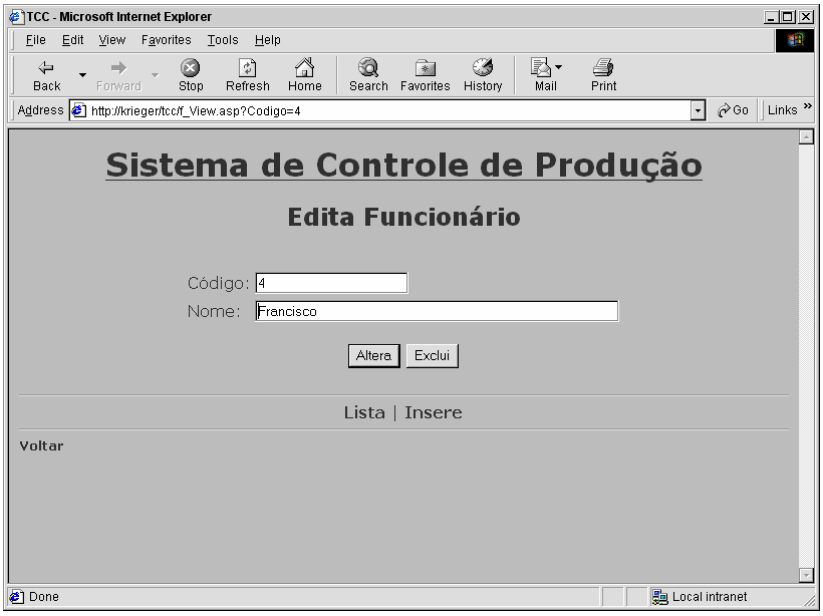
Figura 16: Cadastro de ordens

A Figura 17 exibe a lista de funcionários para o usuário selecionar o item para alterar ou excluir.

Figura 17: Lista de funcionários

A Figura 18 mostra a tela de cadastro de funcionários. Os campos disponíveis são o código e o nome do funcionário.

Figura 18: Cadastro de funcionários

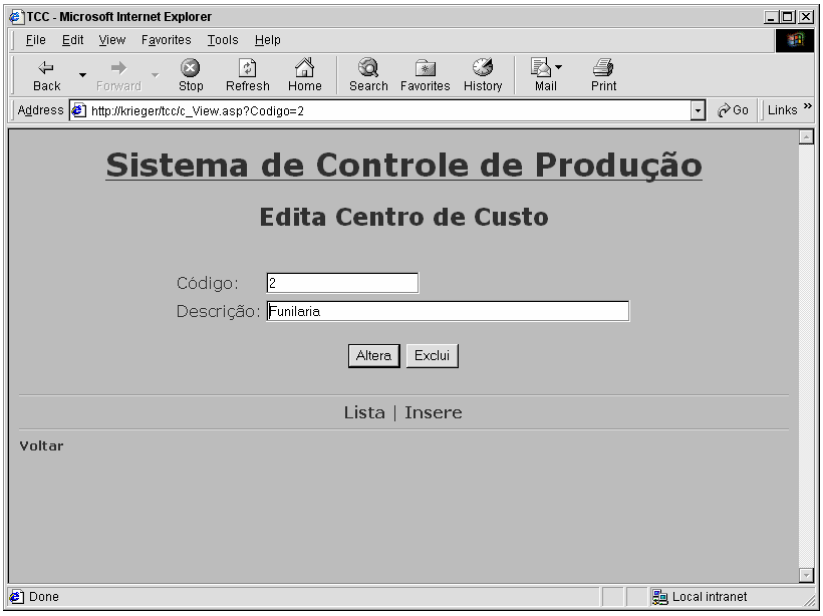


A Figura 19 exibe a lista de centros de custo para respectiva seleção.

Figura 19: Lista de centros de custo

A Figura 20 mostra a tela de cadastro de centros de custo, onde constam o código do centro de custo e sua descrição.

Figura 20: Cadastro de centro de custo

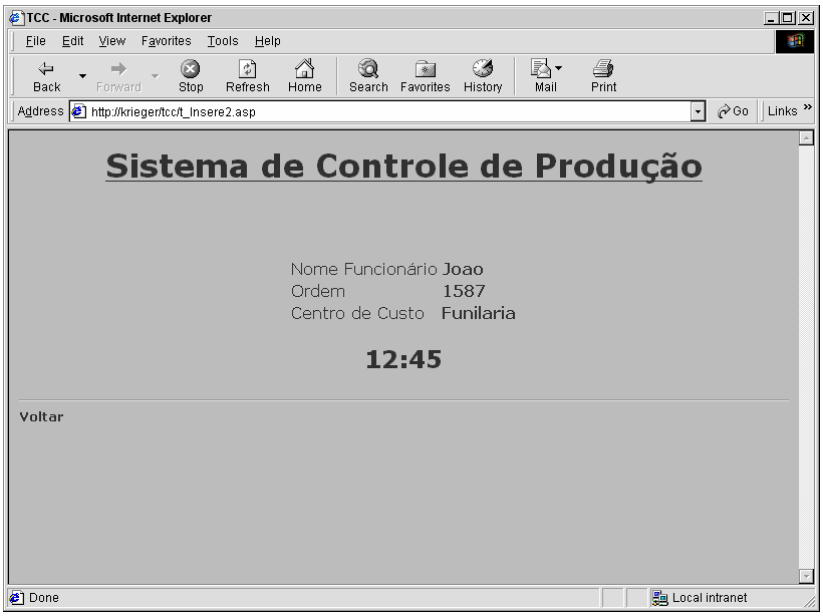


Na Figura 21 é feito a entrada do código ou cartão do funcionário. Se o funcionário não tem nenhuma tarefa em aberto, é exibida a tela da Figura 22 para entrada do código de centro de custo e número da ordem, senão, a tarefa é fechada, exibindo a ordem e centro de custo na qual ele estava trabalhando, e o horário de fechamento, conforme Figura 23.

Figura 21: Abre tarefa

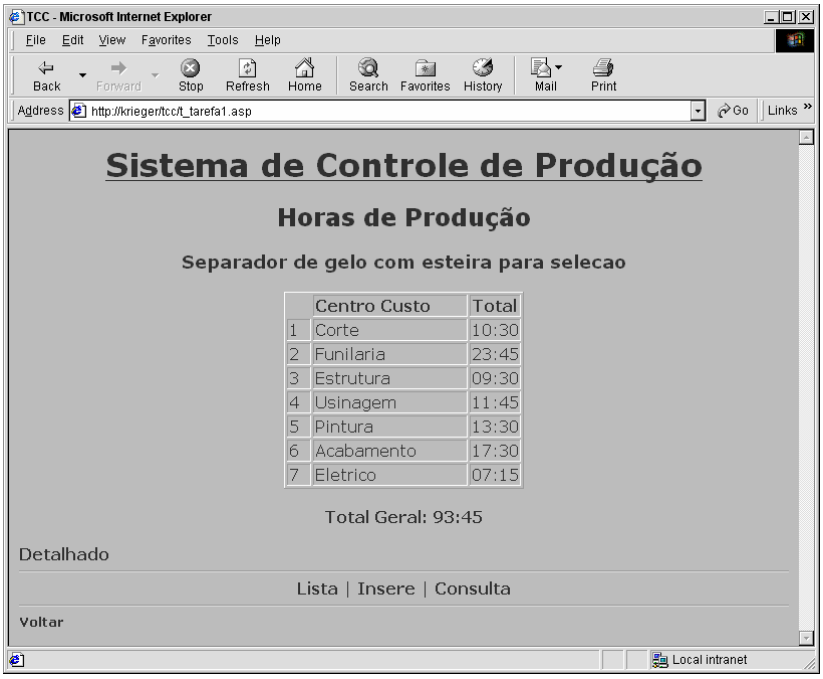
Figura 22: Dados da tarefa

Figura 23: Fechamento da tarefa



A Figura 24 mostra a tela do resultado da consulta de total de horas de produção de determinada ordem. O acesso a esta consulta se dá através do item “manutenção de ordens”.

Figura 24: Consulta



6 CONCLUSÕES

Com o desenvolvimento deste trabalho pode-se concluir que a utilização do ASP com componentes servidores COM expande muito a capacidade de criação de aplicações *web*, onde os desenvolvedores não precisam estar limitados apenas às linguagens de *script* disponíveis ao ASP.

Outra vantagem observada da utilização de um componente para realização do processamento principal de uma aplicação é com relação à segurança. No caso de uma falha de segurança no sistema, se algum usuário tiver acesso aos arquivos da aplicação, este conseguirá saber muito pouco a respeito da aplicação, pois o ASP só chama as funções de um componente.

As experiências anteriores ao desenvolvimento do protótipo eram em Visual C++ e COM, entretanto as noções de acesso a banco de dados utilizando este ambiente eram poucas. Portanto, em primeiro lugar foi necessário o estudo desta técnica.

O conhecimento em ASP e HTML também eram quase nulos. Estudou-se então essas tecnologias para se obter a base necessária para alcançar o objetivo principal, que era uma aplicação em ASP que utilize um componente COM para fazer os principais processamentos da aplicação.

Com o término do protótipo, o sistema será instalado na empresa para os primeiros testes, para posterior implantação definitiva, sendo que a mesma também pretende automatizar outros controles que atualmente mantém na forma manual.

6.1 LIMITAÇÕES E PROBLEMAS

De acordo com as observações realizadas, prevê-se que se esta aplicação fosse totalmente escrita utilizando ASP, já que este possui acesso a banco de dados. Aqui, seria obtido um melhor desempenho do que utilizando COM, pois este tem um grande *overhead* quando é instanciado. Entretanto a vantagem tende a crescer à medida que a complexidade da aplicação aumenta, pois um código binário tem um desempenho muito maior do que um código interpretado, como é o caso do ASP.

6.2 SUGESTÕES

Como sugestões para trabalhos futuros podem ser citados a automatização de outros processos manuais, como o controle de materiais; a integração com sistemas já existentes; e também a disponibilização de informações através da internet, levando-se em consideração principalmente a segurança.

7 REFERÊNCIAS BIBLIOGRÁFICAS

[AND1999] ANDERSON, Richard; HOMER, Alex; ROBINSON, Simom. **Beginning components for ASP**. Birmingham : Wrox, 1999.

[BEN1997] BENNET, Gordon. **Intranets: como implantar com sucesso na sua empresa**. Rio de Janeiro : Campus, 1997.

[CAM1996] CAMERON, Debra. **Intranets: technical issues and business applications**. Charleston : Computer Technology Research, 1996.

[CHI1991] CHIAVENATO, Idalberto. **Iniciação à administração da produção**. São Paulo : Makron, 1991.

[EDD1998] EDDON, Guy. **Inside Distributed COM**. Redmond: Microsoft Press, 1998.

[FUR1998] FURLAN, José Davi. **Modelagem de objetos através da UML**. São Paulo : Makron Books, 1998.

[GRI1998] GRIMES, Richard; et al. **Beginning ATL COM programming**. Birmingham : Wrox, 1998.

[KRU1997] KRUGLINSKI, David J. **Inside Visual C++**. Redmond : Microsoft Press, 1997.

[LEM1998] LEMAY, Laura. **Aprenda em 1 semana HTML 4**. Rio de Janeiro : Campus, 1998.

[MAR1999] MARCORATTI, José Carlos. **ASP, ADO e banco de dados na internet**. Florianópolis : Visual, 1999.

[MIC2000] MICROSOFT Corporation. **Microsoft Developer's Network**. 2000. Endereço eletrônico: http://msdn.microsoft.com

[MUL1997] MULLER, Pierre-Alain. **Instant UML**. Birmingham : Wrox, 1997.

[RED1997] REDMOND III, Frank E..**DCOM:** Microsoft Distributed Component Object Model. Chicago, Indianapolis : IDG Books World Wide, 1997.

[RUM1994] RUMBAUGH, James; et al. **Modelagem e projetos baseados em objetos**. Rio de Janeiro : Campus, 1994.

[UNI2000] UNICAMP, Centro de Computação da. **Desenvolvimento de aplicações para internet**. 2000. http://www.ccuec.unicamp.br

[ZAR1997] ZARATIAN, Beck. **Microsoft Visual C++ owner's manual**. Redmond : Microsoft Press, 1997