**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO**

# DESENVOLVIMENTO DE UM SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE DE PROCESSOS INDUSTRIAIS

**HERMES FRANCISCO VECCHI**

**BLUMENAU**
**2005**

**2005/1-23**

**HERMES FRANCISCO VECCHI**

# DESENVOLVIMENTO DE UM SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE DE PROCESSOS INDUSTRIAIS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Marcel Hugo - Orientador

**BLUMENAU**
**2005**

**2005/1-23**

# DESENVOLVIMENTO DE UM SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE DE PROCESSOS INDUSTRIAIS

Por

**HERMES FRANCISCO VECCHI**

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

| | |
|---|---|
| Presidente: | Prof. Marcel Hugo – Orientador, FURB |
| Membro: | Profa. Fabiane B. V. Benitti |
| Membro: | Prof. Everaldo Artur Grahl |

Blumenau, 13 de julho de 2005

Dedico este trabalho aos meus pais e às minhas tias, que tanto esperaram por este momento.

## AGRADECIMENTOS

À Deus, pela força e sabedoria.

Aos meus pais, Paulo e Rose, minhas tias, Edir, Amélia e Terezinha, meus irmãos Paulo e Juliana, que sempre me incentivaram e acreditaram no meu potencial.

Aos meus amigos, Renato e Célio, pela ajuda e incentivo.

À minha namorada, Jô, pela paciência, ajuda e companheirismo, principalmente neste período de muito trabalho.

Ao meu orientador, Marcel Hugo, por ter me mostrado o caminho, me deixando andar com as próprias pernas, tornando maior a satisfação ao descobrir as soluções.

A todos os amigos que sempre foram companheiros, incentivando e ajudando nessa jornada.

À *Softlution*, que através da CPU projetada proporcionou o tema principal desse trabalho.

A mente que se abre a uma nova idéia, jamais voltará ao seu tamanho original.

Albert Einstein.

# RESUMO

Este trabalho apresenta o desenvolvimento de um *software* de controle de processos industriais, que gerencia uma CPU micro-processada produzida pela *Softlution*. O software garante ao usuário um controle visual dos processos existentes em uma planta de fábrica. As disciplinas de requisitos, análise e projeto, e implementação do RUP foram aplicadas para o desenvolvimento do software, detalhando as etapas desse processo, amplamente utilizado pelas principais empresas de software do mercado.

Palavras-chave: Controle de processos industriais. RUP.

# ABSTRACT

This work presents the development of a software of control of industrial processes, that manages a micron-processed CPU produced by the Softlution. Software guarantees to the user a visual control of the existing processes in a plant plant. You discipline them of requirements, analysis and project, and implementation of the RUP had been applied for the development of software, detailing the stages of this process, widely used for the main companies of software of the market.



Key-Words: Control of Industrial Processes, RUP.

# LISTA DE ILUSTRAÇÕES

# LISTA DE SIGLAS

CPU – Unidade Central de Processamento

CLP – Controlador Lógico Programável

OMG – *Object Management Group*

OMT – *Object Modeling Technique*

PID – Proporcional Integral Derivativo

RTF – *Rich Text File*

RUP – *Rational Unified Process*

SCADA – *Supervisory Control and Data Acquisition*

SISO – *Single Input, Single Output*

TI – Tecnologia da Informação

UML – Linguagem de Modelagem Unificada

UP – *Unified Process*

VCL – *Visual Component Library*

# SUMÁRIO

# 1 INTRODUÇÃO

O impacto das tecnologias de informação sobre as formas de organização da produção implicaram mudanças significativas no âmbito microeconômico. O novo paradigma organizacional da produção tornou a capacidade de produzir, aperfeiçoar e inovar o principal ativo estratégico das empresas (LACERDA et al., 2001, p. 24). Com isso, as empresas procuram soluções práticas e baratas para aumentar a produtividade e a qualidade, diminuir custos e evitar desperdícios. Tudo isso pode ser alcançado através de um controle rigoroso dos processos industriais envolvidos na fabricação de um determinado produto, como por exemplo, o controle de um forno elétrico, que deve ser mantido na temperatura ideal para que o produto seja fabricado com qualidade. Para isso são instalados sensores de temperatura que quando detectam uma temperatura abaixo da desejada, soam um alarme, o qual pode ser sonoro, alertando para a necessidade do aumento da temperatura ou que fazem operar um atuador para re-equilibrar a mesma.

Atualmente este controle é feito de forma individualizada, com um sistema de controle específico para cada empresa, de acordo com os processos industriais utilizados na fábrica, o que dificulta a padronização e comercialização de soluções para os mais variados casos existentes no mercado, tornando caro seu desenvolvimento.

Para agilizar este controle e diminuir seu custo, a *Softlution – Software Solutions*, que segundo seu proprietário Renato Gozdziejewski Jr, é uma empresa que atua desde 1999 no mercado da informática, no ramo de consultoria e desenvolvimento de soluções em tecnologia da informação (TI), visando diversos setores da atividade industrial, sempre respeitando a individualidade de cada cliente, desenvolveu um protótipo de um controlador de processos industriais de baixo custo e que se adapta aos processos conforme as características da empresa, visando atender empresas que tenham esta necessidade, mas sem a disponibilidade

de grandes investimentos.

O equipamento consiste em uma unidade central de processamento (CPU) micro-processada e pode ser modularizado, com canais de entradas e saídas digitais, analógicas, saídas de potência, comunicação serial, que controla não só sensores como atuadores, mandando comandos para os processos, como por exemplo, aumentar a temperatura. Esta CPU gerencia uma rede de equipamentos micro-processados que poderão estar ligados a qualquer processo industrial, tais como, controle e aquisição de dados de temperatura, pH, vazão, nível, umidade relativa, peso, entre outros.

Para facilitar o gerenciamento de um processo industrial monitorado por tal CPU, este trabalho desenvolveu um *software* que controla a CPU e manipula as informações obtidas para prover o gerenciamento do processo industrial completo de forma visual, propiciando ao usuário uma interface amigável para o controle da sua planta.

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é o desenvolvimento de um sistema que gerencie um processo industrial através das informações provenientes do controlador de processos industriais.

Os objetivos específicos do trabalho são:

a) disponibilizar um *software* gerenciador para a referida CPU, desenvolvido utilizando as disciplinas de requisitos, análise e projeto, e implementação do processo de desenvolvimento RUP;
b) disponibilizar o gerenciamento do processo industrial completo de forma visual;
c) avaliar a adequação das disciplinas de requisitos, análise e projeto, e implementação do RUP para o desenvolvimento deste tipo de aplicação.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos: Introdução, Fundamentação Teórica, Desenvolvimento do Trabalho e Conclusões.

No primeiro capítulo é introduzida a idéia principal do trabalho e os objetivos a serem alcançados.

No segundo capítulo é apresentada uma fundamentação teórica que aborda as metodologias utilizadas no desenvolvimento do trabalho, bem como a especificação da CPU para a qual o protótipo apresentado foi desenvolvido.

No terceiro capítulo é demonstrado todo o processo de desenvolvimento do protótipo, desde sua especificação até a versão final, mostrando os artefatos gerados para sua documentação, exemplos de códigos de programação e demonstração do funcionamento do protótipo.

No quarto capítulo são apresentadas conclusões e sugestões para extensões do trabalho.

# 2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são relacionadas ferramentas e padrões que auxiliaram na montagem do *software* e que propiciaram um estudo detalhado do RUP, bem como uma visão dos documentos gerados de acordo com o método para o melhor controle do desenvolvimento.

## 2.1 CONTROLE DE PROCESSOS

Um sistema de controle é uma coleção de componentes que trabalham juntos no sentido de propiciar alguma inteligência a uma máquina. Na maioria dos casos, os circuitos eletrônicos fornecem a inteligência, e os componentes eletromecânicos, tais como sensores e motores, fornecem a relação com o mundo físico. Um bom exemplo é o automóvel moderno.

Os vários sensores fornecem ao computador de bordo informações sobre a condição dos instrumentos. O computador então calcula a quantidade de combustível que precisa ser injetado no motor e ajusta o sincronismo de ignição. As partes mecânicas do sistema incluem o motor, transmissão, rodas, e assim por diante. Para projetar, diagnosticar, ou reparar estes sistemas sofisticados, deve-se compreender a eletrônica, a mecânica, e princípios do sistema de controle. No passado, as máquinas automatizadas, ou os processos, eram controlados por circuitos eletrônicos analógicos, ou por circuitos que usavam interruptores, relés, e temporizadores. A partir do advento do microprocessador de baixo custo, mais e mais sistemas redesenharam dispositivos para incorporar um controlador microprocessado. Os exemplos incluem copiadoras, os robôs, e os controladores de processo industrial (DELMAR, 2004, p. 12).

O controle de processos industriais é uma das aplicações de sistemas de controle amplamente usada, que supervisiona cada processo de modo que uma saída uniforme correta seja mantida. Isto é feito monitorando e ajustando os parâmetros de controle (tais como a taxa da temperatura ou de fluxo) para assegurar-se de que o produto de saída seja o desejado. O exemplo clássico do controle de processo é um sistema que mantém uma temperatura especificada em um forno elétrico. O sistema é composto por um aquecedor, que aumenta a temperatura do forno, um termostato, para controle da temperatura, e um par termelétrico, dispositivo que converte a temperatura em tensão. O controlador regula o aquecedor de tal maneira que sustenta a temperatura (como relatado pelo par termelétrico) no valor especificado no sistema (DELMAR, 2004, p. 12).

O controle do processo pode ser classificado como sendo um processo de grupo ou um processo contínuo. Em um processo contínuo há um fluxo material contínuo. Um processo de grupo tem um começo e uma extremidade (que é executada geralmente repetidamente). Os exemplos de processos de grupo incluem misturar um grupo da massa de pão e o

carregamento de caixas em um carrinho. Em uma planta grande, tal como uma refinaria, muitos processos estão ocorrendo simultaneamente e devem ser coordenados, porque a saída de um processo é a entrada de outro (DELMAR, 2004, p. 13).

## 2.2 REPRESENTAÇÃO VISUAL DE PROCESSOS

Conforme Lopez (2002, p. 15), *Supervisory Control and Data Acquisition* (SCADA) são os sistemas de supervisão de processos industriais que coletam dados do processo através de estações remotas, principalmente Controladores Lógicos Programáveis (CLP), formatam estes dados, e os apresentam ao operador em uma multiplicidade de formas. O objetivo principal dos sistemas SCADA é propiciar uma interface de alto nível do operador com o processo, informando-o "em tempo real" de todos os eventos de importância da planta.

Hoje os sistemas de supervisão oferecem três funções básicas que são definidas a seguir:

a) funções de supervisão: inclui todas as funções de monitoramento do processo tais como: sinóticos animados, gráficos de tendência de variáveis analógicas e digitais, relatórios em vídeo e impressos, entre outro.

b) funções de operação: atualmente os sistemas SCADA substituíram com vantagens as funções da mesa de controle. As funções de operação incluem: ligar e desligar equipamentos e seqüência de equipamentos, operação de malhas proporcional integral derivativo (PID), mudança de modo de operação de equipamentos, etc.

c) funções de controle: alguns sistemas de supervisão possuem uma linguagem que permite definir diretamente ações de controle, sem depender de um nível intermediário de controle representado por ações remotas inteligentes. Todas as operações de entrada e saída são executadas diretamente através de cartões de I/O ligados diretamente ao barramento do micro. Os dados são amostrados, um

algoritmo de controle como um controlador PID, por exemplo, é executado, e a saída é aplicada ao processo (ação direta sobre uma variável manipulada). Isto, entretanto só é possível quando a velocidade do processo assim o permite. Em alguns casos requisitos de confiabilidade tornam desaconselhável este tipo de solução.

## 2.3 PROTÓTIPO DESENVOLVIDO PARA O CONTROLE DE PROCESSOS

Conforme descrição da *Softllution*, o protótipo do equipamento de controle de processos que será gerenciado pelo *software* desenvolvido neste trabalho consiste em uma CPU com as seguintes características:

a) processador PIC18F452, 40MHz da *Microchip* (arquitetura RISC);
b) 3 ou 6 canais analógicos de entrada de 16 bits de resolução;
c) 4 canais analógicos de saída de 8, 12 ou 16 bits de resolução de 0-5V ou em *loop* de corrente 4-20mA ou 0-20mA;
d) 4 canais configuráveis (entrada ou saída) digitais opto-isoladas (isolamento ótico);
e) 3 saídas de acionamento por relês;
f) 1 saída de potência (relê de estado sólido);
g) *display* lcd 16 X 2;
h) teclado 4 teclas;
i) comunicação serial EIA-232/485;
j) *Real Time Clock*;
k) até 128 Kbytes de memória EEPROM;
l) protocolo de comunicação *ModBus*.

O equipamento pode ser utilizado para aplicações genéricas de supervisão, controle e aquisição de dados ou SCADA, o que não impede que seja programado para atender

aplicações específicas (customizadas) dependendo da complexidade e necessidade de automação do processo.

Para a aquisição de dados, o equipamento pode fazer as leituras dos 6 ou 3 canais analógicos de tempos em tempos conforme a configuração do tempo de amostragem de cada canal.

A leitura é convertida para a unidade desejada através das configurações de calibração. A calibração, em síntese, é a transformação do valor do sinal amostrado para a grandeza desejada. Por exemplo, se o sinal proveniente do sensor de temperatura variar entre 4 e 20 mA, na calibração é relacionado o valor mínimo de 4 mA a 0°C e o valor máximo de 20 mA a temperatura de 1700°C.

Estes valores amostrados são gravados em uma área de memória do equipamento que ficará disponível para alguma solicitação de leitura do computador servidor ou exibição em *display*.

No caso do controle de processos, o equipamento poderá ser configurado para especificação de malhas de controle do tipo *single input, single output* (SISO). A malha de controle SISO especifica qual grandeza medida no processo, através dos sensores (canais de entrada analógicos), deverá ser controlada, ou seja, mantida no valor desejado ou valor de *set point*, através de um algoritmo de controle.

O equipamento utiliza o algorítmo PID, e através de suas saídas analógicas ou de potência manipula algum atuador (válvula, resistência elétrica, por exemplo). Este elemento final de controle atua no comportamento do processo realizando o gerenciamento.

O computador servidor, através da aplicação visual, pode acessar ou manipular qualquer canal do controle, sendo possível a ligação de até 256 equipamentos em rede (EIA-232/485) conforme especificação do protocolo *Modbus*.

## 2.4 LINGUAGEM DE MODELAGEM UNIFICADA – UML

A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas complexos de software que proporciona uma forma-padrão para a preparação de planos de arquitetura de projetos de sistema. A UML inclui aspectos conceituais tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de *software* reutilizáveis (BOOCH; RUMBAUGH; JACOBSON, 2000, p. XIII).

A UML começou em 1994, com um esforço conjunto de Grady Booch e Jim Rumbaugh para combinar as notações diagramáticas dos seus dois difundidos e populares métodos – o *Booch*, que era expressivo durante as fases de projeto e construção de sistemas e *Object Modeling Technique* (OMT), que era mais útil para análise e sistemas de informações com uso intensivo de dados. Posteriormente, se juntou a eles Ivar Jacobson, criador do método *Objectory*, que fornecia excelente suporte para os casos como uma maneira de controlar a captura de requisitos, a análise e o projeto em alto nível. O grupo era conhecido como os três amigos (LARMAN, 2004, p. 34).

Os documentos da primeira versão da UML foram lançados em 1996, desde então, com a adesão de novos parceiros, outras versões de aprimoramento foram lançadas até chegar na versão 2.0 usada atualmente, tornando a UML, para muitas empresas, um recurso estratégico para seus negócios (BOOCH; RUMBAUGH; JACOBSON, 2000, p. XVII). A UML foi adotada, em 1997, como padrão pelo *Object Management Group* (OMG), uma instituição de criação de padrões para indústria de software.

A UML é uma linguagem visual de programação, mas seus modelos podem ser diretamente conectados a várias linguagens de programação já que é capaz de representar tudo que é melhor expresso em termos gráficos, enquanto as linguagens de programação representam o que é melhor expresso em termos textuais (BOOCH; RUMBAUGH;

JACOBSON, 2000, p. 16).

Essa conexão permite a realização de uma engenharia de produção, que é a geração de código a partir de um modelo em UML para uma linguagem de programação e o processo inverso, reconstruindo um modelo a partir de sua implementação, processo chamado de engenharia reversa (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 16).

Para permitir a visualização de um sistema sob diferentes aspectos, a visão parcial dos elementos que compõem o sistema é documentada através de diagramas, geralmente representados como gráficos de vértices (itens) e arcos (relacionamentos). A UML 1.3 inclui os seguintes diagramas (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 25):

a) diagrama de classes: encontrados com maior freqüência em sistemas de modelagem orientados a objeto, exibem um conjunto de classes, interfaces, colaborações e seus relacionamentos. Os diagramas de classe abrangem uma visão estática da estrutura do sistema;
b) diagrama de objetos: representa retratos estáticos de instâncias de itens encontrados em diagramas de classes, exibindo um conjunto de objetos e seus relacionamentos. Como nos diagramas de classes, abrangem a visão estática da estrutura ou processo de um sistema, mas sob perspectiva de casos reais ou de protótipos;
c) diagrama de casos de uso: são importantes principalmente para a organização e a modelagem de comportamentos do sistema. Exibem um conjunto de casos de uso, atores (papéis desempenhados por usuários ou equipamentos ao interagir com casos de uso) e seus relacionamentos. Abrangem a visão estática de casos de uso do sistema;
d) diagramas de interação: abrangem a visão dinâmica de um sistema, exibindo uma interação, consistindo de um conjunto de objetos e seus relacionamentos, incluindo

as mensagens trocadas entre eles. Tanto o diagrama de seqüências, cuja ênfase está na ordenação temporal das mensagens, quanto o diagrama de colaborações, cuja ênfase está na organização estrutural dos objetos que enviam e recebem mensagens, são tipos de diagramas de interação. Os dois diagramas são isomórficos, permitindo que um diagrama de um tipo seja transformado em um diagrama de outro tipo;

e) diagrama de gráfico de estados: são importantes principalmente para a modelagem de comportamentos de interface, classe ou colaboração, dando ênfase a comportamentos de um objeto ordenados por eventos. Exibe uma máquina de estados, formada por estados, transições, eventos e atividades e abrange a visão dinâmica de um sistema;

f) diagrama de atividades: é importante para a modelagem funcional de um sistema, dando ênfase ao fluxo de controle entre objetos. Abrange a visão dinâmica de um sistema;

g) diagrama de componentes: exibe as organizações e as dependências existentes em um conjunto de componentes. Estão relacionados ao diagrama de classes, já que normalmente os componentes estão mapeados para classes, interfaces ou colaborações;

h) diagrama de implantação: mostra a configuração dos nós de processamento em tempo de execução e os componentes neles existentes, por isso está relacionado aos diagramas de componentes. Abrange a visão estática do funcionamento de uma arquitetura (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 26).

Mas a UML não é apenas um linguagem gráfica. Para cada notação gráfica existe uma especificação capaz de fornecer uma declaração textual da sintaxe e da semântica do respectivo bloco de construção. Por exemplo, relacionada a uma construção de classe, existe

uma especificação que fornece o conjunto de atributos, operações e comportamentos da classe (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 28).

A combinação dos blocos de construção e das especificações dos mesmos possibilita a construção de modelos de maneira incremental, desenhando diagramas e depois acrescentando uma semântica às especificações (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 28).

Segundo Booch, Rumbaugh e Jacobson (2000, p. 34), a UML não se limita ao ciclo de vida de desenvolvimento de determinado *software*, porém, para obter o máximo proveito da UML, será preciso que o processo tenha as seguintes características:

a) orientado por caso de uso,

b) centrado na arquitetura,

c) processo iterativo e incremental.

## 2.5 RATIONAL UNIFIED PROCESS (RUP)

O *Rational Unified Process* (RUP) é um processo de desenvolvimento de *software* inspirado no processo que atende pelo nome de Processo Unificado (ou UP do inglês *Unified Process*), e faz uso extensivo da notação UML (SCOTT, 2003, p. 19).

Ele oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Sua meta é garantir a produção de software de alta qualidade que atenda às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis (RATIONAL, 2002).

O RUP tem as principais características para obter o máximo proveito da UML:

a) orientado por caso de uso: refere-se ao fato de utilizar os casos de uso para dirigir todo o trabalho de desenvolvimento, desde a captação dos requisitos até a

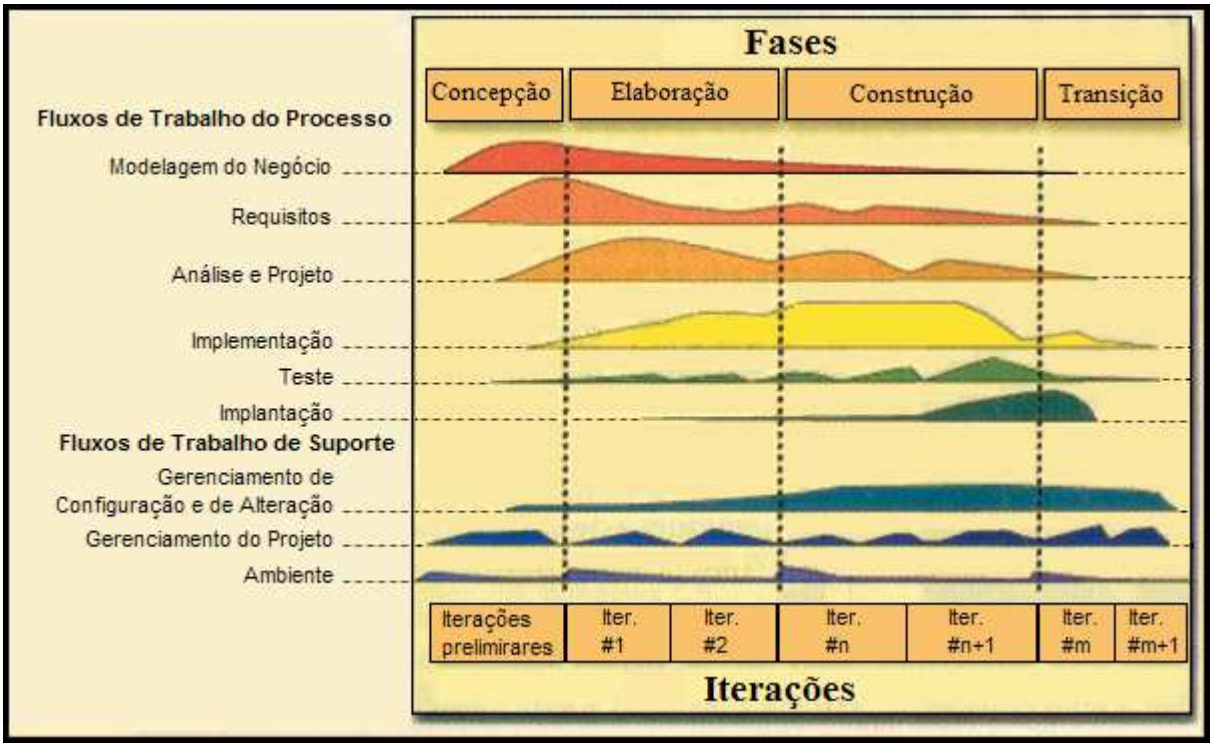
aceitação do código. Os casos de uso são expressos em língua natural (língua nativa dos clientes), são intuitivamente óbvios para o leitor, oferecendo uma habilidade consideravelmente maior para a compreensão dos requisitos reais do sistema do que em textos convencionais. O RUP emprega os casos de uso como força condutora de desenvolvimento(SCOTT, 2003, p. 21).

b) centrado na arquitetura: a arquitetura é a organização fundamental do sistema como um todo, incluindo aspectos como elementos estáticos, elementos dinâmicos, o modo como estes elementos trabalham juntos e o estilo arquitetônico total que guia a organização do sistema. No RUP, durante as iterações iniciais, propõe-se uma arquitetura candidata que ofereça a base para uma fundação sólida. Durante as iterações posteriores, expande-se a visão para uma arquitetura completa, que influencia a maior parte, senão todas, as tarefas de desenvolvimento realizadas como parte de uma dada iteração (SCOTT, 2003, p. 22).

c) iterativo e incremental: envolve o gerenciamento de seqüências de versões executáveis e integração contínua da arquitetura do sistema para a produção dessas versões, de maneira que cada versão incorpora os aprimoramentos incrementais em relação às demais (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 34). Segundo SCOTT (2003, p. 24-25), o RUP estabelece a quebra do sistema em unidades de construção, onde cada construção é uma versão operacional de alguma parte significativa do sistema completo. Ao focar um conjunto limitado de casos de uso e fazer uso de protótipos, a equipe de projetos e os clientes podem negociar requisitos de modo progressivo. O objetivo é que os problemas sejam isolados dentro das iterações, lidando com eles em uma escala relativamente pequena, em vez de permitir que se alastrem.

Segundo SCOTT (2003, p. 26), a vida de um sistema de software pode ser

representada com uma série de ciclos, onde cada ciclo termina com a liberação de uma versão do sistema para os clientes. No RUP, cada ciclo contém quatro fases, conforme figura 1. Uma fase é o intervalo de tempo decorrido entre dois importantes marcos do processo, quando um conjunto bem definido de objetivos é alcançado, artefatos são concluídos e decisões são tomadas para se passar à fase seguinte.




Fonte: Booch, Rumbaugh e Jacobson (2000, p. 18)

Figura 1 – Fases do RUP

A arquitetura geral do RUP tem duas dimensões:

a) o eixo horizontal mostra os aspectos do ciclo de vida do processo em relação ao tempo em que se desenvolve. Representa o aspecto dinâmico do processo e é expresso em termos de fases, iterações e marcos;

b) o eixo vertical representa as disciplinas que agrupam as atividades de maneira lógica, por natureza e representa o aspecto estático do processo, sendo descrito em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis do processo.

## 2.6 CONCEPÇÃO

Segundo LARMAN (2004, p. 58), a idéia desta fase é fazer uma investigação para formar uma opinião racional e justificável da finalidade geral e da viabilidade potencial do novo sistema, para então decidir se vale a pena investir em uma exploração mais profunda.

A fase de concepção deve ser relativamente curta, com poucas semanas de duração para a maioria dos projetos. Se for definido de antemão que o projeto será feito e for claramente viável, então a fase será breve. Ela pode incluir os primeiros *workshops* sobre requisitos, o planejamento da primeira iteração, passando rapidamente para a fase de elaboração.

## 2.7 ELABORAÇÃO

O principal objetivo desta fase é estabelecer a capacidade para a construção do novo sistema, dadas as restrições financeiras e de cronograma e outros tipos de restrições com que o desenvolvimento do projeto se defronta (SCOTT, 2003, p. 28).

Uma das coisas mais importantes a ser feita na fase de elaboração é descobrir todos os casos de uso potenciais para o sistema, baseados nos requisitos levantados.

Sabe-se que a fase de elaboração foi completada quando todos os casos de uso foram levantados. Os desenvolvedores podem se sentir à vontade para dar estimativas de quanto tempo levará para construir cada caso de uso e todos os riscos significativos foram identificados e compreendidos ao ponto de saber como lidar com eles (LARMAN, 2004, p. 38).

## 2.8 CONSTRUÇÃO

Segundo Martin e Scott (2000, p. 40), a construção elabora o projeto em uma série de iterações. Cada iteração é um mini-projeto, onde são feitas análise, projeto, codificação, teste

e integração para os casos de uso designados para cada iteração.

Nesta fase o modelo de casos de uso é concluído (isso significa que todos os interessados estão de acordo com os requisitos funcionais e referenciados no sistema), bem como todos os outros modelos: análise, projeto, instalação, implementação e teste. A arquitetura foi modificada conforme necessário para garantir que o sistema disponibilizado reflita a intenção especificada na descrição da arquitetura (SCOTT, 2003, p. 117).

## 2.9 TRANSIÇÃO

O objetivo principal na transição é entregar uma versão beta do sistema aos clientes, enquanto continuam os esforços para resolver defeitos que os clientes descobrem no mesmo, bem como as sugestões de funcionalidades que não se qualificam como defeitos, desde que sejam pequenas e absolutamente essenciais (SCOTT, 2003, p. 129).

Para isso é necessário que todos os materiais de apoio estejam prontos, como guias do usuário, materiais de treinamento e outros, que são confeccionados durante a fase de construção.

## 2.10 ITERAÇÕES E INCREMENTOS

Cada fase do processo ainda pode ser dividida em iterações. Uma iteração é um ciclo completo de desenvolvimento, resultando em uma versão de um produto executável que constitui um subconjunto do produto final e cresce de modo incremental de uma iteração para outra para se tornar o sistema final. Cada iteração passa pelos vários fluxos de trabalho do processo, embora sua ênfase seja diferente a cada fase (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 446).

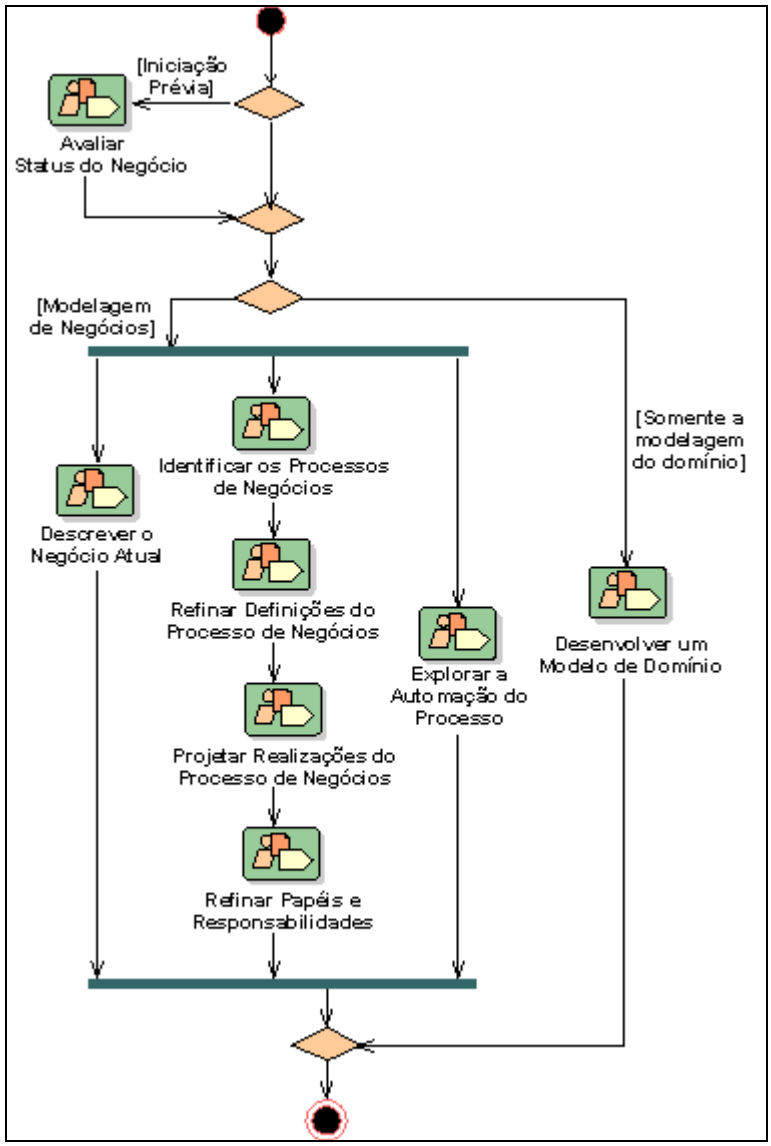
## 2.11 DISCIPLINAS DO RUP

Cada disciplina do RUP tem um fluxo de trabalho que define a seqüência de atividades realizadas na disciplina. As atividades são unidades de trabalho que um indivíduo – ou conjunto de indivíduos trabalhando em equipe – pode ser solicitado a executar. O comportamento e as responsabilidades de cada indivíduo dentro do contexto de uma organização de engenharia de *software* são definidos através de papéis (RATIONAL, 2002).

### 2.11.1 Modelagem de negócios

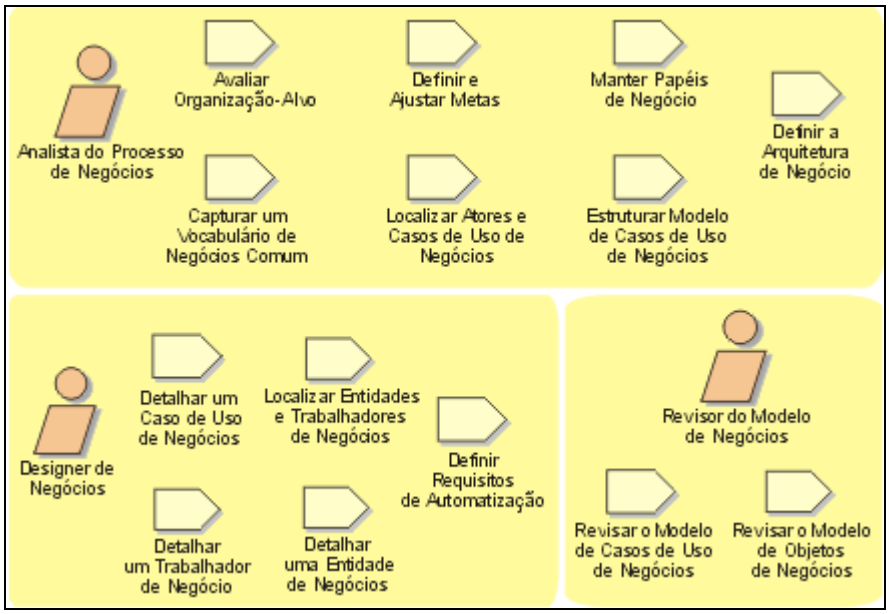
A modelagem de negócios descreve como desenvolver uma visão da nova organização-alvo e, com base nesta visão, definir os processos, os papéis e as responsabilidades dessa organização em um modelo de casos de uso de negócios e em um modelo de objetos de negócios (RATIONAL, 2002).

Na figura 2 é mostrado o fluxo de trabalho genérico apresentado no RUP para a disciplina de modelagem de negócios. As atividades e os papéis envolvidos na disciplina são mostrados na figura 3. A figura 4 mostra os artefatos produzidos na disciplina e os papéis envolvidos na confecção dos mesmos.



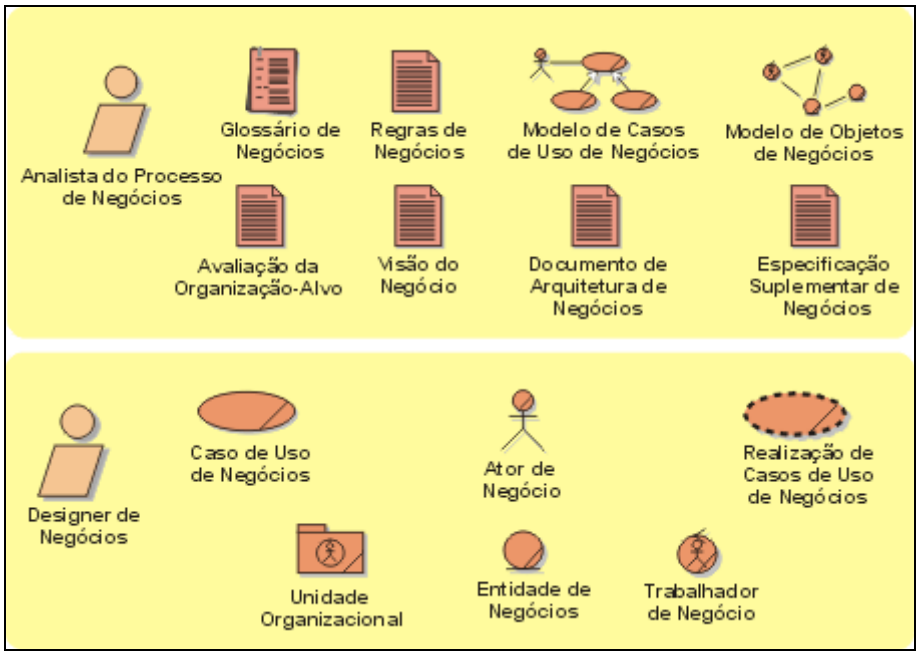

Fonte: Rational (2002)

Figura 2 – Fluxo de trabalho genérico para a disciplina de modelagem de negócios




Fonte: Rational (2002)

Figura 3 – Atividades da disciplina de modelagem de negócios




Fonte: Rational (2002)

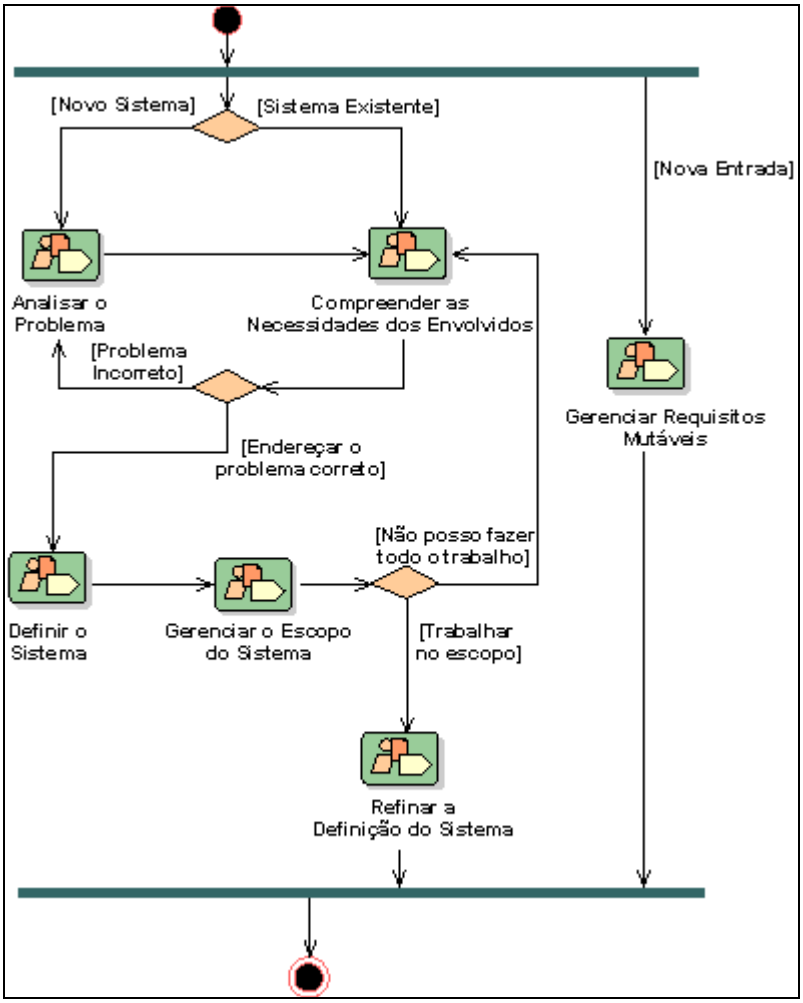
Figura 4 – Artefatos desenvolvidos na disciplina de modelagem de negócios

### 2.11.2 Requisitos

Na disciplina de requisitos é importante, antes de tudo, compreender a definição e o

escopo do problema que se tenta resolver com este sistema. As regras de negócios, o modelo de casos de uso de negócios e o modelo de objetos de negócios desenvolvidos durante a modelagem do negócio servirão como informações importantes nesse trabalho. Os envolvidos são identificados e as solicitações dos principais envolvidos são recolhidas, reunidas e analisadas (RATIONAL, 2002).

A figura 5 mostra o fluxo de trabalho genérico do RUP para a disciplina. Na figura 6 e 7 são mostradas as atividades e os artefatos desenvolvidos na disciplina, respectivamente, bem como, os papéis envolvidos.

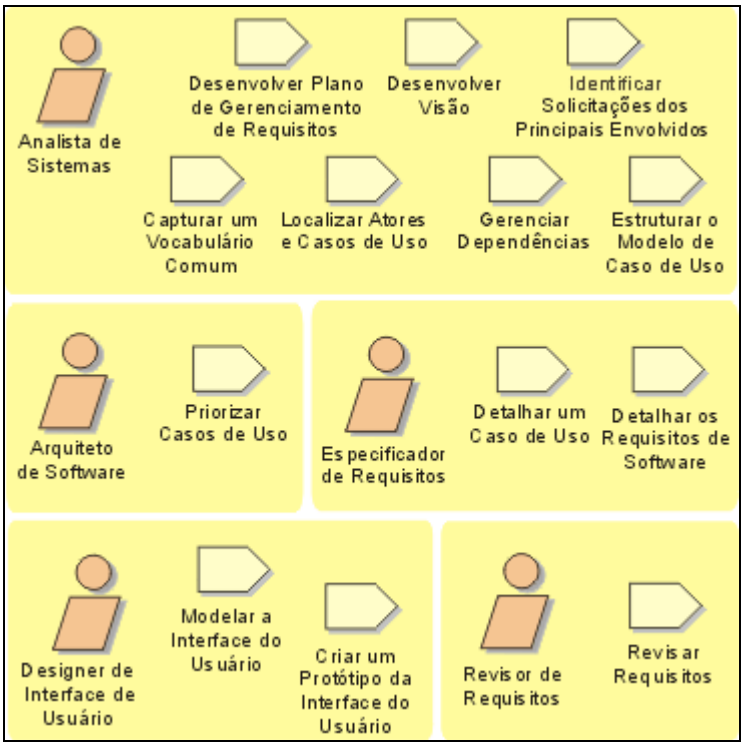



Fonte: Rational (2002)

Figura 5 – Fluxo de trabalho genérico para a disciplina de Requisitos

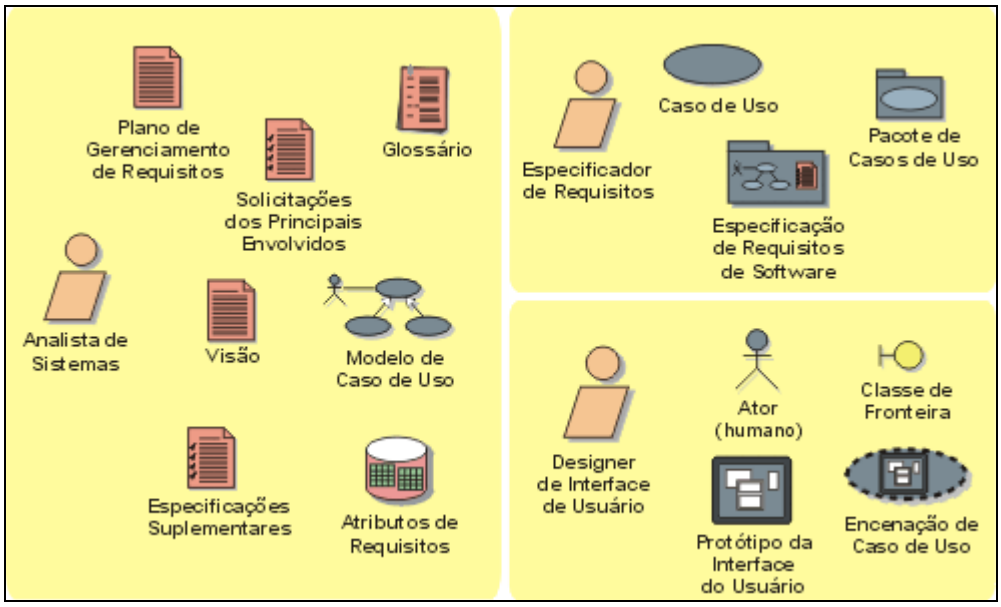
Cada detalhamento do fluxo de trabalho representa uma habilidade-chave que precisa ser aplicada para executar o gerenciamento eficiente de requisitos. *Analisar o Problema* e

*Compreender as Necessidades dos Envolvidos* são o foco durante a fase de *Concepção* de um projeto, ao passo que a ênfase recai em *Definir o Sistema* e *Refinar a Definição do Sistema* durante a fase de *Elaboração*. *Gerenciar o Escopo do Sistema* e *Gerenciar Requisitos Variáveis* são detalhes feitos continuamente ao longo do projeto (RATIONAL, 2002).




Fonte: Rational (2002)

Figura 6 – Atividades da disciplina Requisitos



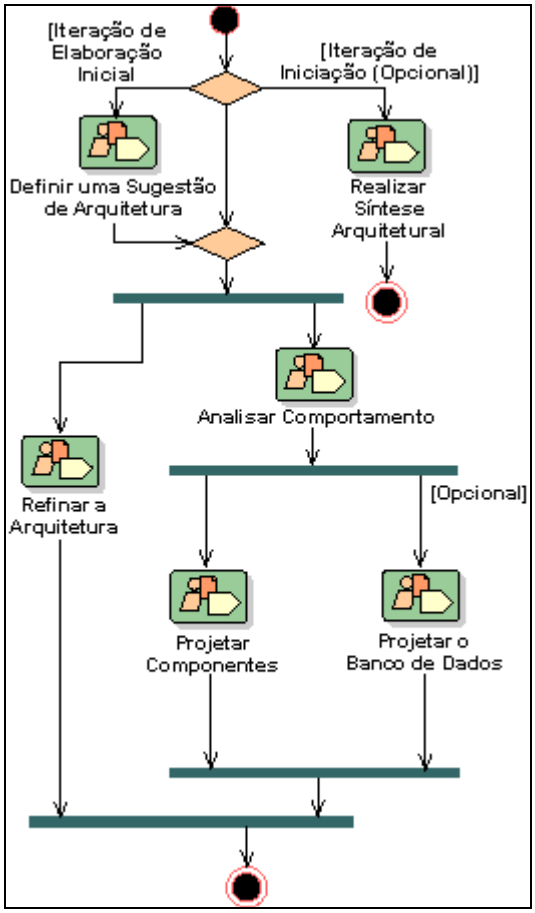

Fonte: Rational (2002)

Figura 7 – Artefatos desenvolvidos na disciplina de Requisitos

### 2.11.3 Análise e Projeto

A disciplina de análise e projeto tem como finalidade transformar os requisitos em um projeto do sistema a ser criado, desenvolver uma arquitetura sofisticada para o sistema e adaptar o projeto para que corresponda ao ambiente de implementação, projetando-o para fins de desempenho (RATIONAL, 2002).

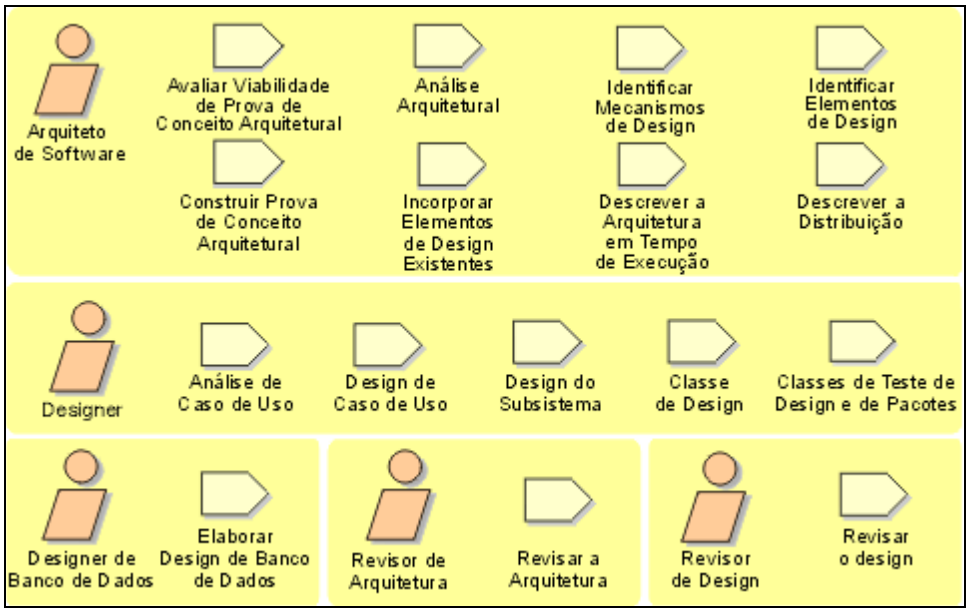
Nas figuras 8, 9 e 10 são mostrados o fluxo de trabalho genérico do RUP, as atividades e os artefatos desenvolvidos na disciplina de análise e projeto.




Fonte: Rational (2002)

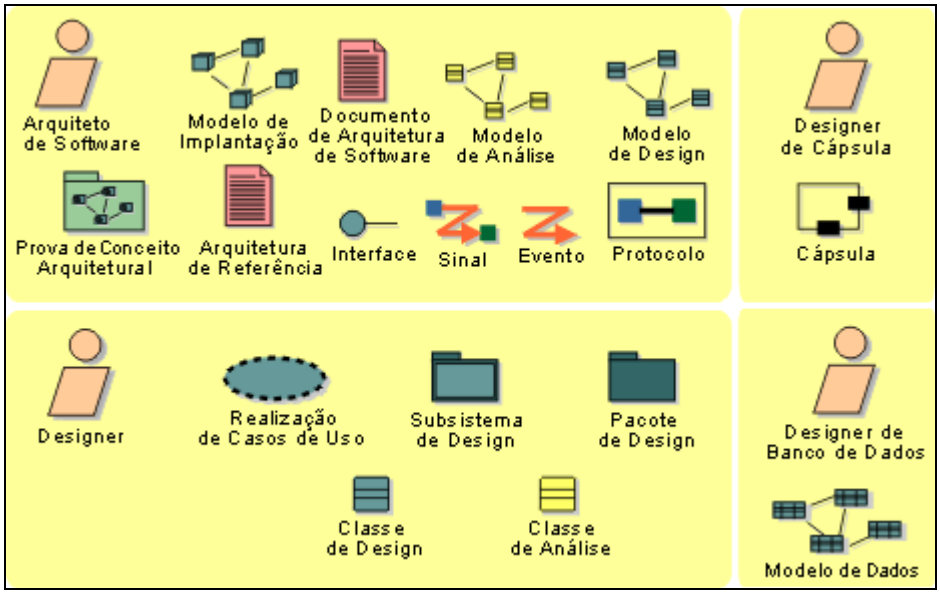
Figura 8 – Fluxo de trabalho genérico para a disciplina Análise e Projeto

Na fase de *Concepção*, função da disciplina de análise e projeto será determinar se o sistema é viável, conforme o previsto, e avaliar as tecnologias possíveis para a solução. Na fase de Elaboração, a disciplina enfatiza a criação de uma arquitetura inicial para o sistema, a fim de fornecer um ponto de partida para o trabalho de análise inicial. Se a arquitetura já existir, o enfoque do trabalho mudará e passará para o refinamento da arquitetura (Refinamento da Arquitetura) e para a análise do comportamento e a criação de um conjunto inicial de elementos que fornecerão o comportamento apropriado (Analisar Comportamento). Depois que os elementos iniciais forem identificados, eles serão posteriormente refinados (RATIONAL, 2002).




Fonte: Rational (2002)

Figura 9 – Atividades da disciplina Análise e Projeto




Fonte: Rational (2002)

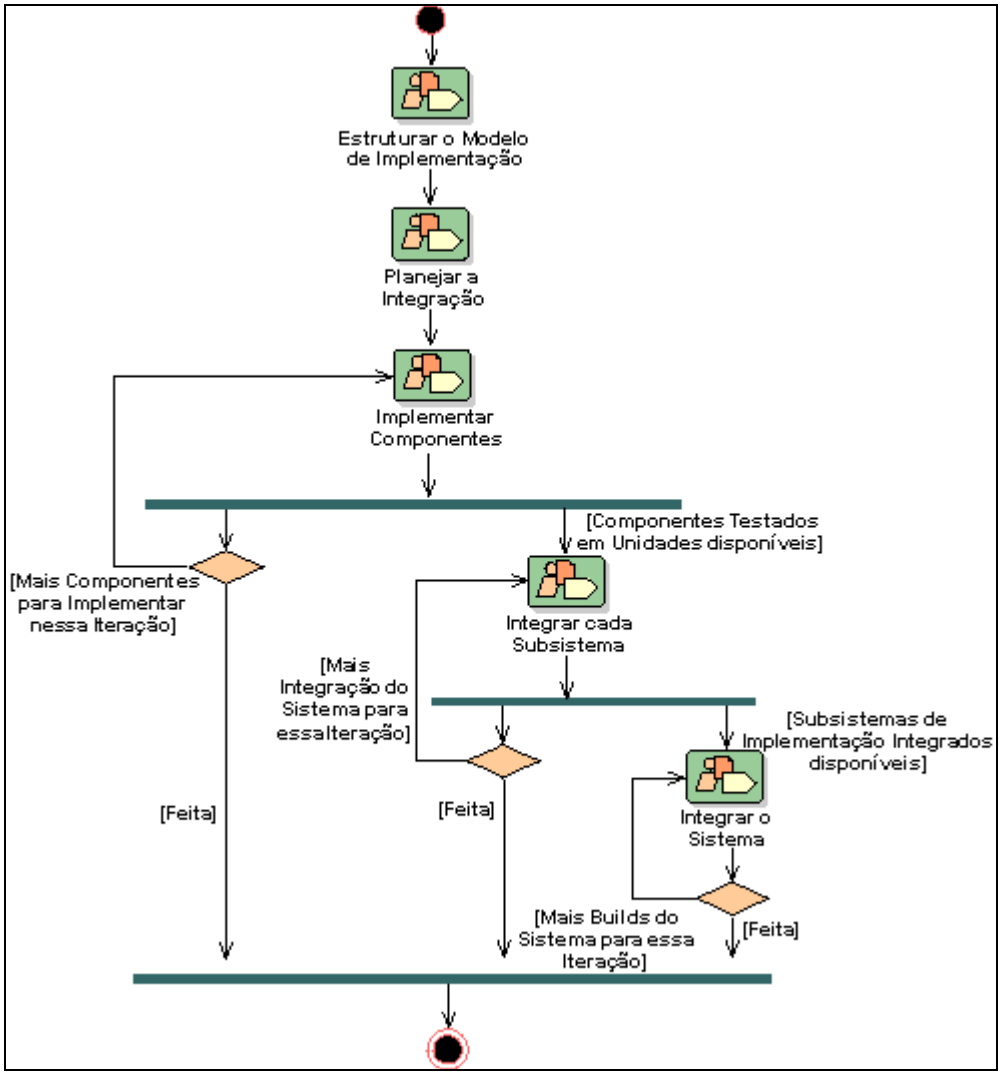
Figura 10 – Artefatos desenvolvidos na disciplina Análise e Projeto

### 2.11.4 Implementação

A Implementação tem como finalidade definir a organização do código em termos de subsistemas de implementação organizados em camadas, implementar classes e objetos em termos de componentes, testar os componentes desenvolvidos como unidades e integrar os

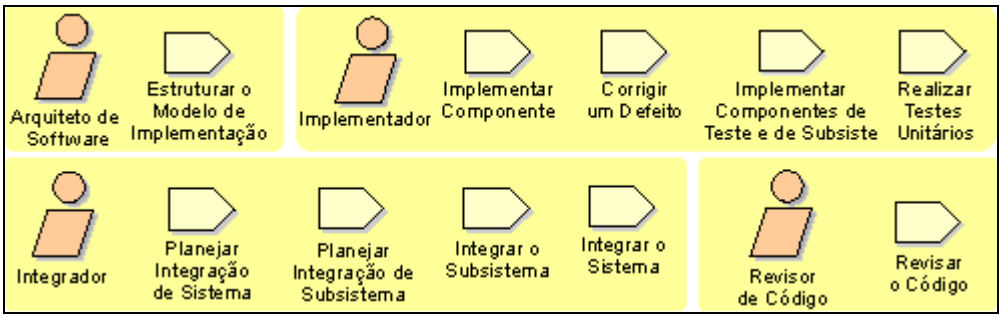
resultados produzidos por implementadores individuais (ou equipes) ao sistema executável (RATIONAL, 2002).

A figura 11 mostra o fluxo de trabalho genérico do RUP para a disciplina, onde a *Estrutura do Modelo* de *Implementação* é realizada na fase inicial de *Elaboração* e para cada iteração, começando na *Elaboração*, pode-se *Planejar a Integração*, *Implementar os Componentes*, *Integrar cada Subsistema* e, por fim, *Integrar o Sistema* (RATIONAL, 2002). As figuras 12 e 13 mostram as atividades da disciplina e os artefatos desenvolvidos, respectivamente.



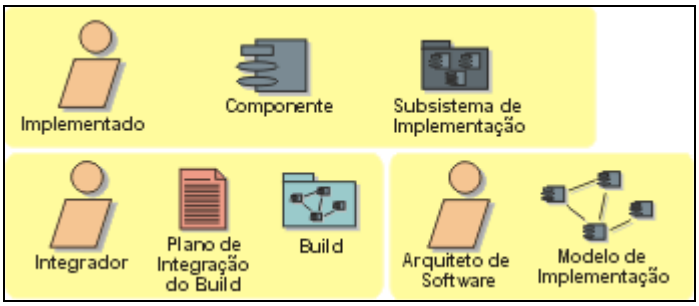

Fonte: Rational (2002)

Figura 11 – Fluxo de trabalho genérico para a disciplina Implementação




Fonte: Rational (2002)

Figura 12 – Atividades da disciplina Implementação



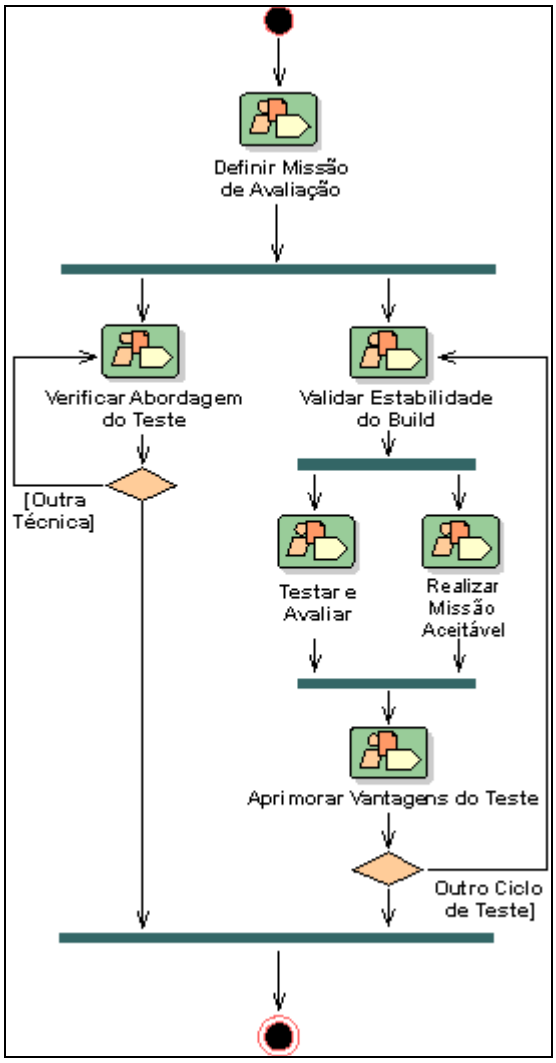

Fonte: Rational (2002)

Figura 13 – Artefatos desenvolvidos na disciplina de Implementação

### 2.11.5 Testes

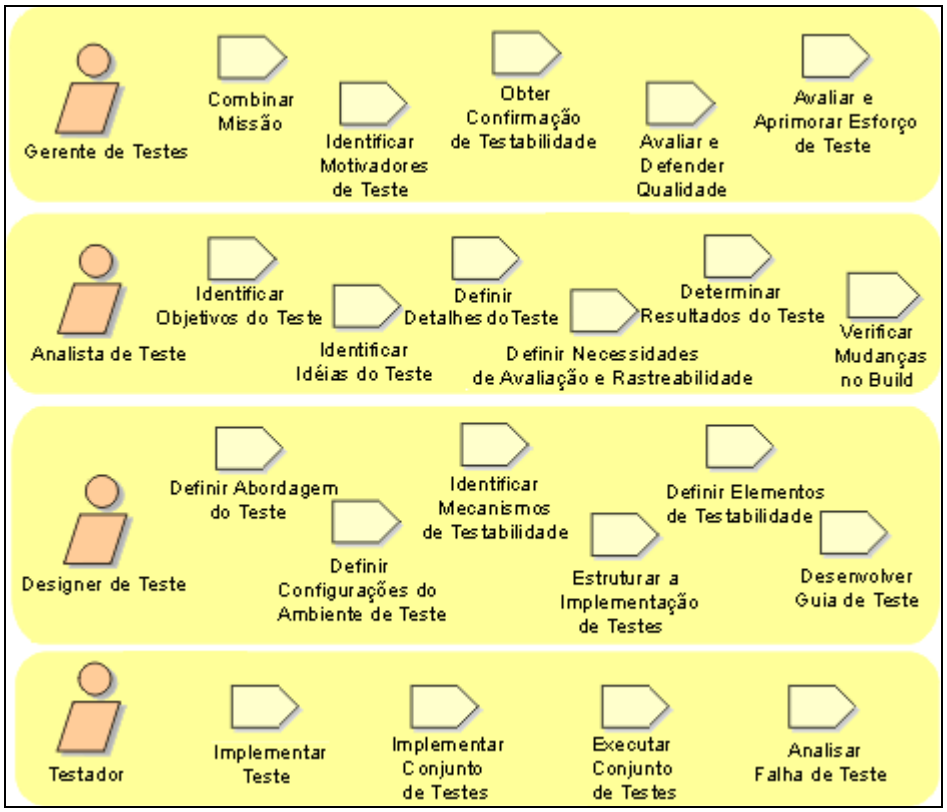
A principal finalidade do teste é localizar e expor os pontos fracos do software. O teste desafia as suposições, os riscos e as incertezas inerentes ao trabalho de outras disciplinas, tratando essas questões por meio de uma demonstração concreta e uma avaliação imparcial (RATIONAL, 2002).

Nas figuras 14, 15 e 16, são apresentados o fluxo de trabalho genérico do RUP para a disciplina, suas atividades e papéis, e os artefatos desenvolvidos na disciplina, respectivamente.



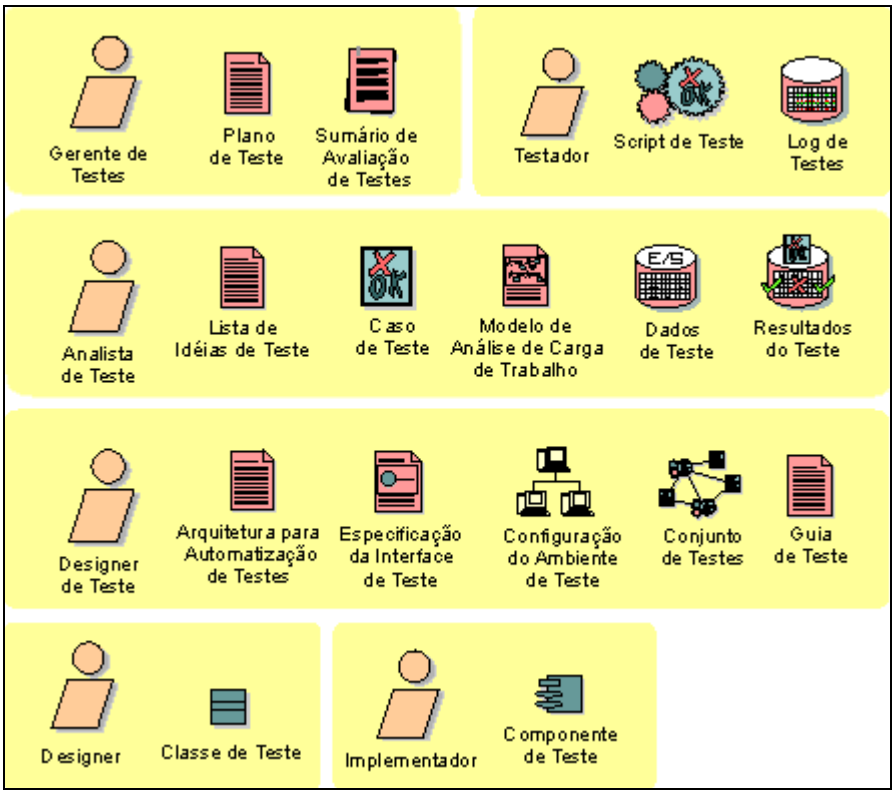

Fonte: Rational (2002)

Figura 14 – Fluxo de trabalho genérico para a disciplina de Testes




Fonte: Rational (2002)

Figura 15 – Atividades da disciplina Testes



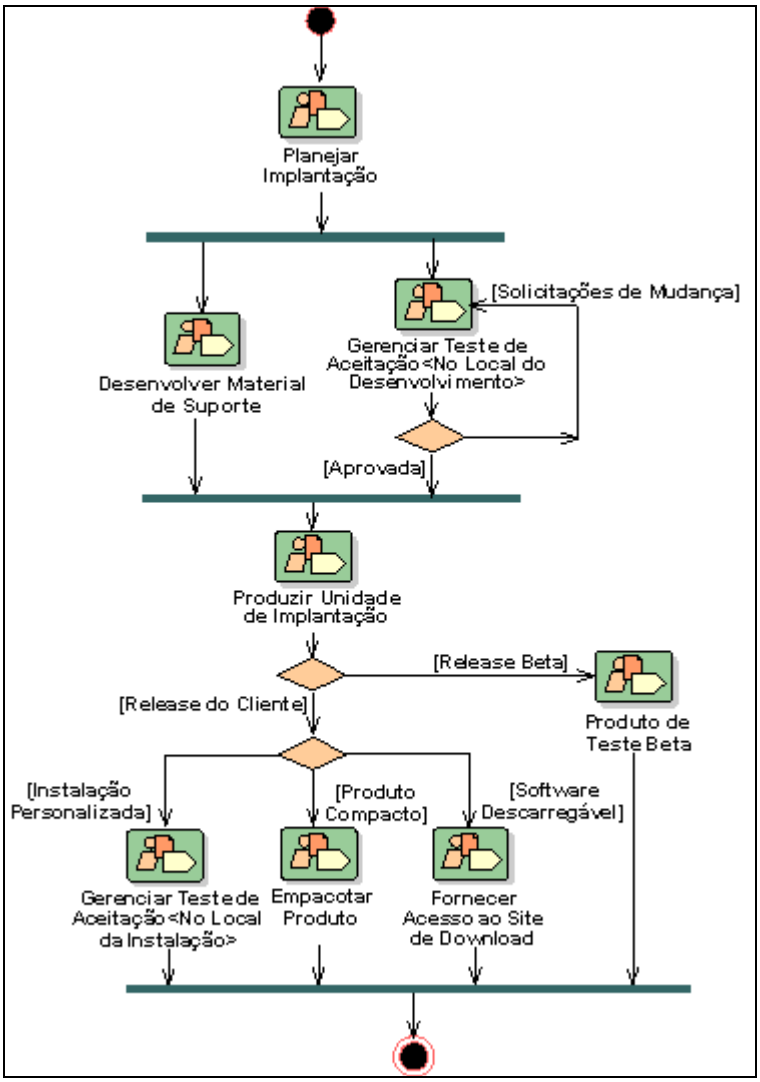

Fonte: Rational (2002)

Figura 16 – Artefatos desenvolvidos na disciplina Testes

### 2.11.6 Implantação

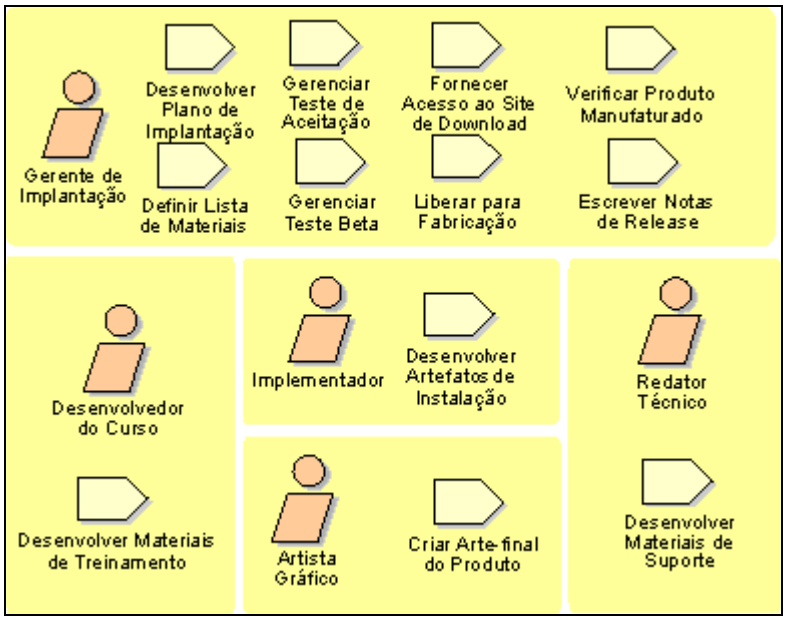
A Implantação descreve as atividades que garantem que o produto de software será disponibilizado a seus usuários finais, dando ênfase, a cada instância, em testar o produto no local de desenvolvimento, através de testes beta, antes do mesmo ser oferecido ao cliente (RATIONAL, 2002).

O fluxo de trabalho genérico do RUP para a disciplina de Implantação é mostrado na figura 17. Nas figuras 18 e 19 são mostradas as atividades e os artefatos desenvolvidos na disciplina, respectivamente.




Fonte: Rational (2002)

Figura 17 – Fluxo de trabalho genérico para a disciplina de Implantação




Fonte: Rational (2002)

Figura 18 – Atividades da disciplina Implantação


Fonte: Rational (2002)

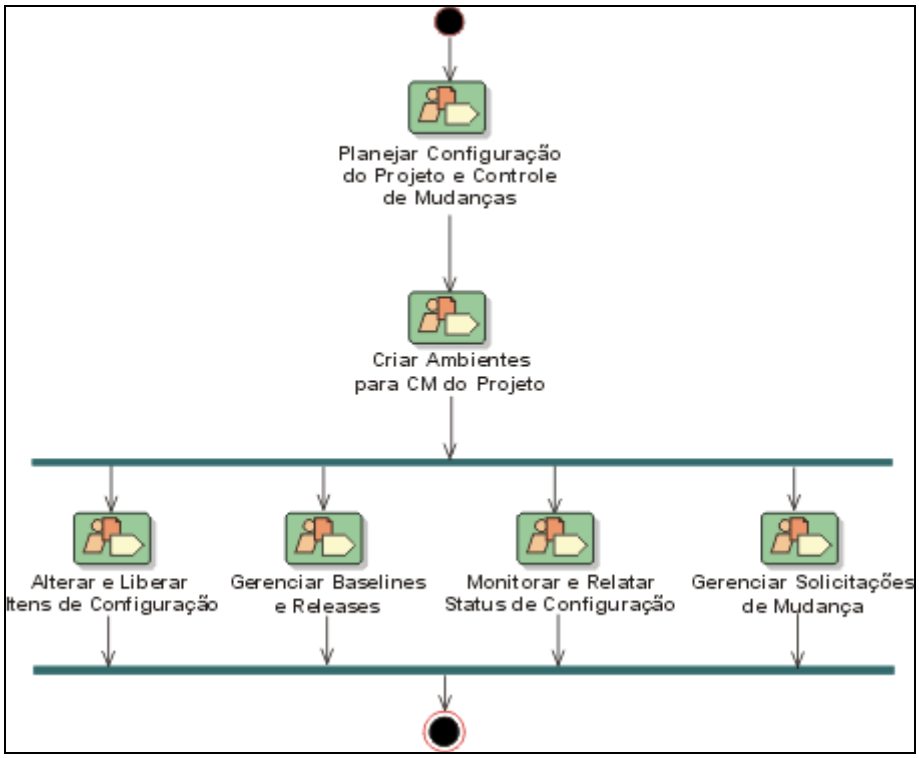
Figura 19 – Artefatos desenvolvidos na disciplina de Implantação

### 2.11.7 Gerenciamento de Configuração e Alteração

A disciplina de Gerenciamento de Configuração e Alteração é fundamental para controlar os inúmeros artefatos produzidos pelas muitas pessoas que trabalham em um mesmo projeto. O controle ajuda a evitar confusões dispendiosas e garante que os artefatos resultantes

não entrem em conflito devido a alterações simultâneas, notificação limitada e várias versões liberadas (RATIONAL, 2002).

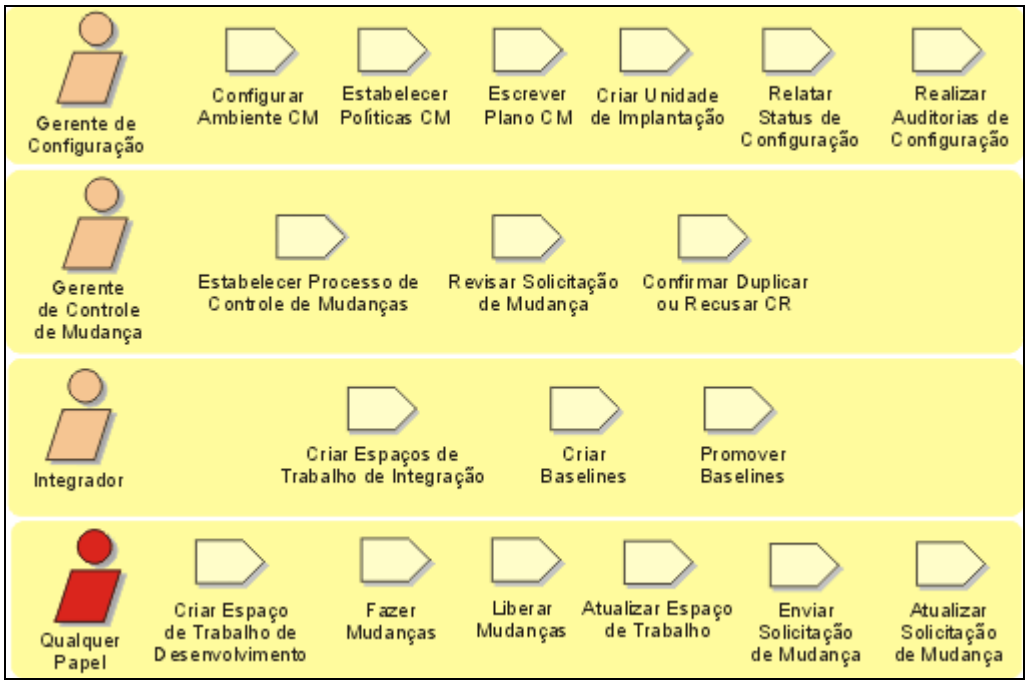
As figuras 20, 21 e 22 mostram, respectivamente, o fluxo de trabalho genérico para a disciplina, as atividades e papéis da disciplina, e os artefatos desenvolvidos na disciplina.




Fonte: Rational (2002)

Figura 20 – Fluxo de trabalho genérico para a disciplina Gerenciamento de Configuração e Alteração

Os dois primeiros detalhamentos do fluxo de trabalho são ativados no início de um projeto. Os demais são ativados de modo contínuo no decorrer do ciclo de vida do projeto (RATIONAL, 2002).




Fonte: Rational (2002)

Figura 21 – Atividades da disciplina Gerenciamento de Configuração e Alteração


Fonte: Rational (2002)

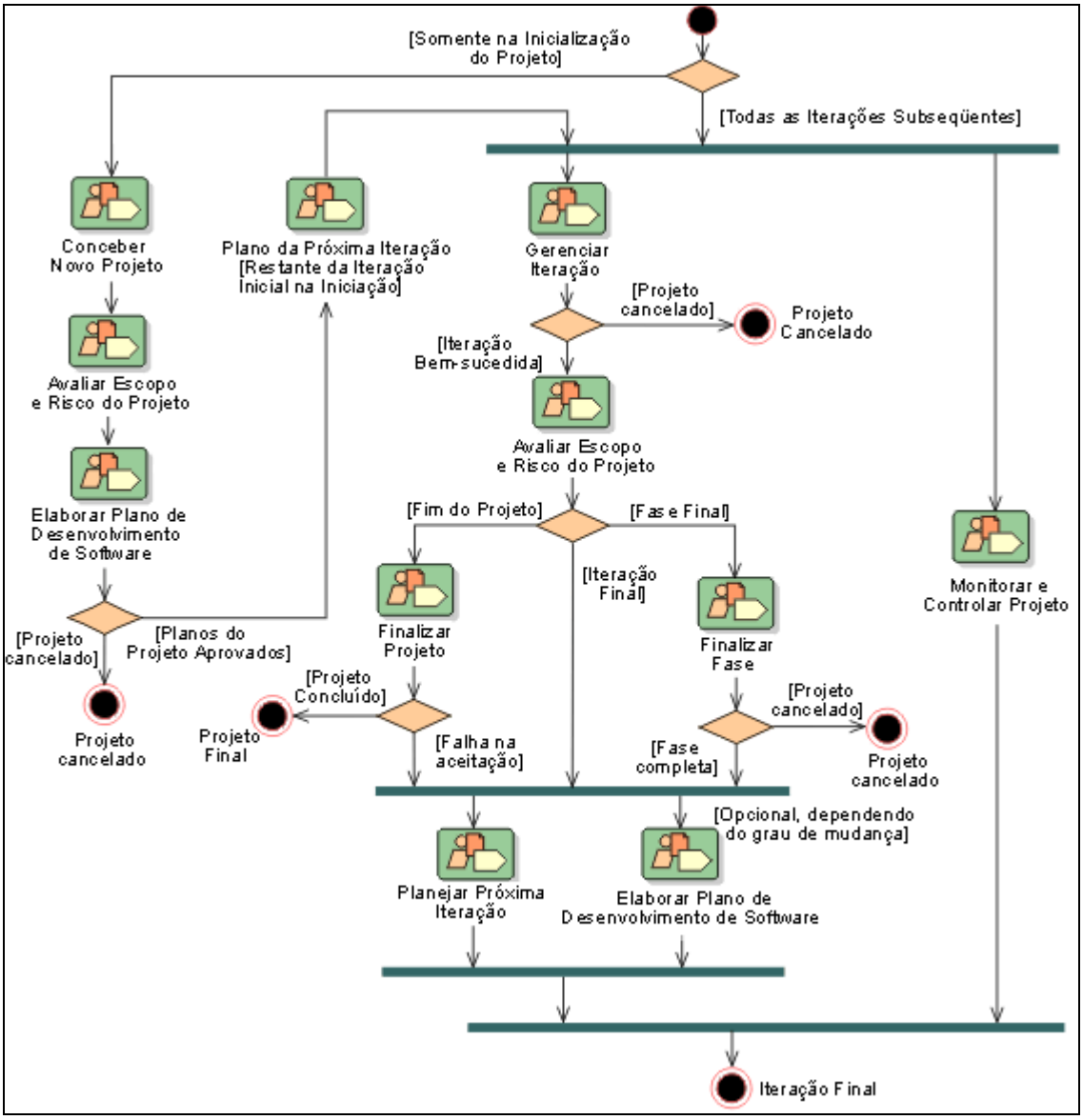
Figura 22 – Artefatos desenvolvidos na disciplina Gerenciamento de Configuração e Alteração

### 2.11.8 Gerenciamento de Projeto

O Gerenciamento de Projeto de Software é a arte de confrontar os objetivos da concorrência, gerenciar riscos e superar obstáculos para liberar com êxito um produto que

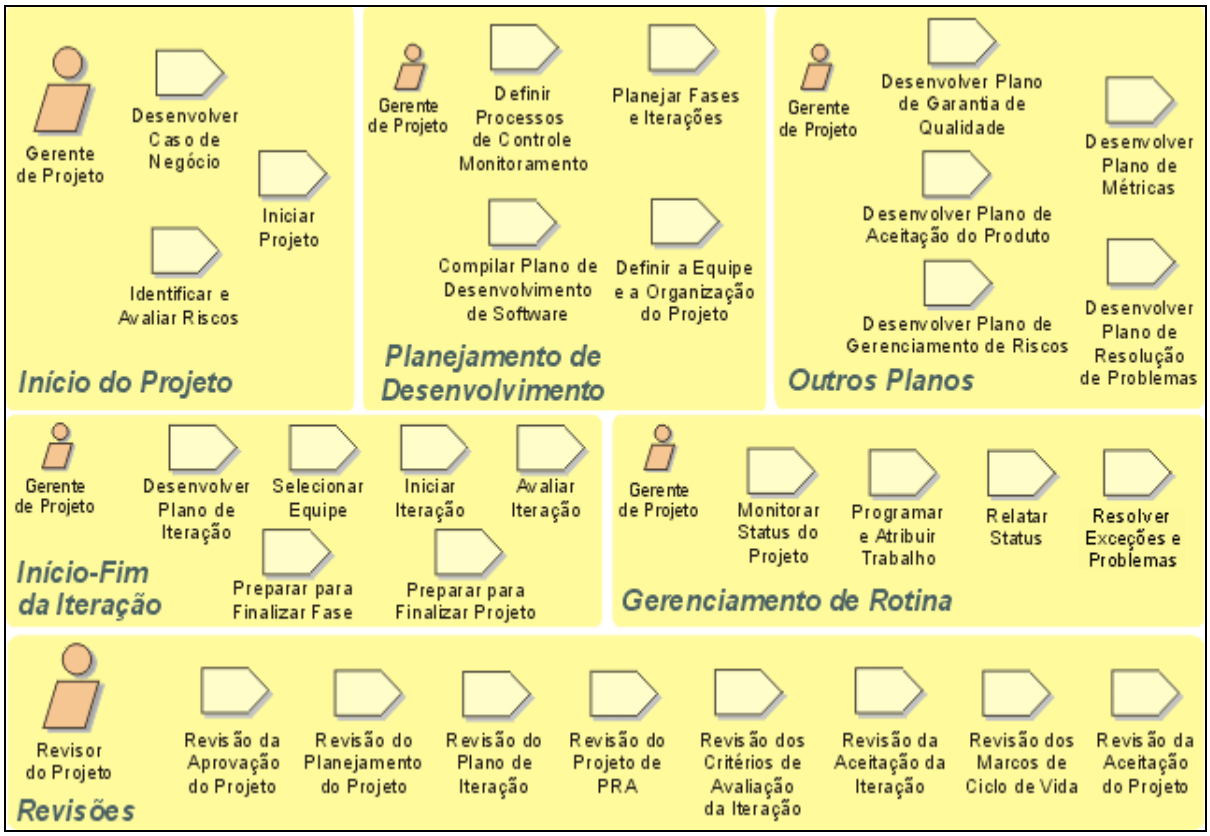
atenda às necessidades dos clientes e dos usuários. A principal finalidade do Gerenciamento de Projeto é fornecer um *framework* para gerenciar projetos intensivos de *software*, fornecer diretrizes práticas para planejar, montar a equipe, executar e monitorar os projetos e gerenciar os riscos (RATIONAL, 2002).

Na figura 23 é apresentado o fluxo de trabalho genérico do RUP para a disciplina. As atividades e papéis são mostrados na figura 24. Os artefatos desenvolvidos são apresentados na figura 25.



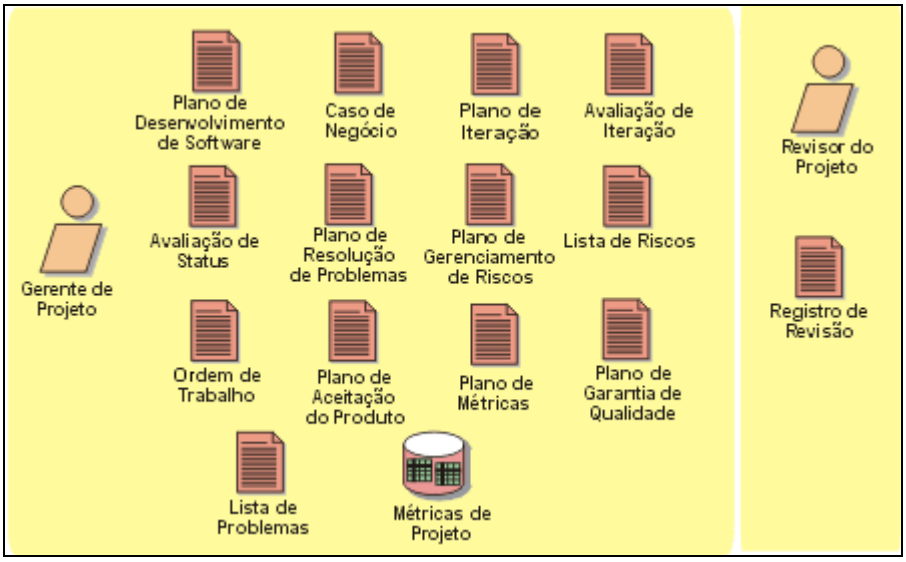

Fonte: Rational (2002)

Figura 23 – Fluxo de trabalho genérico para a disciplina Gerenciamento de Projeto




Fonte: Rational (2002)

Figura 24 – Atividades da disciplina Gerenciamento de Projeto




Fonte: Rational (2002)

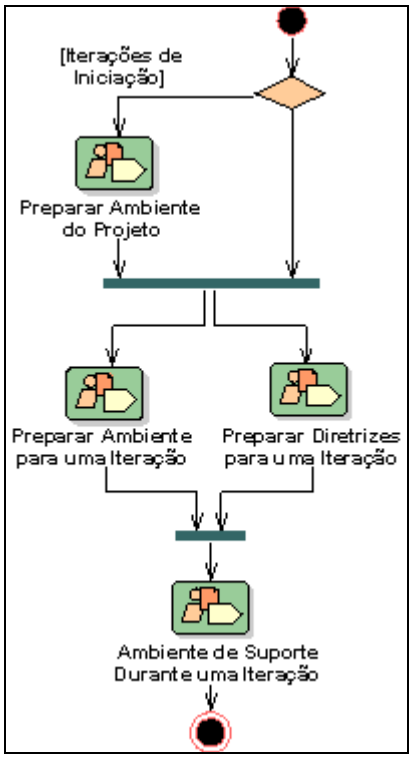
Figura 25 – Artefatos desenvolvidos na disciplina Gerenciamento de Projeto

### 2.11.9 Ambiente

A meta das atividades dessa disciplina é oferecer à organização o ambiente de desenvolvimento de software que dará suporte à equipe de desenvolvimento.

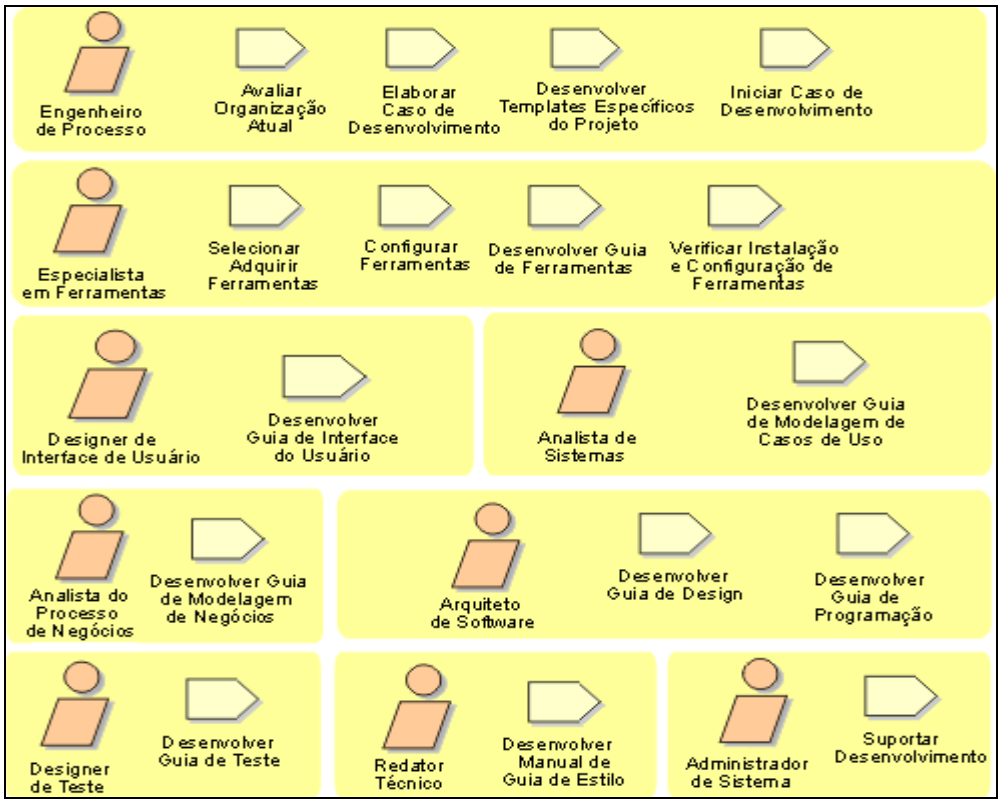
Na figura 26 é apresentado o fluxo de trabalho genérico do RUP para a disciplina, onde, nas iterações iniciais do projeto, deve ser iniciado o fluxo de trabalho com *Preparar Ambiente para Projeto*, cuja principal conseqüência é *Avaliação da Organização de Desenvolvimento*. Em seguida, para cada iteração, segue-se *Preparar Ambiente para uma Iteração* e *Preparar Diretrizes para uma Iteração* (RATIONAL, 2002).

A figura 27 mostra as atividades da disciplina e a figura 28, os artefatos desenvolvidos para a mesma.



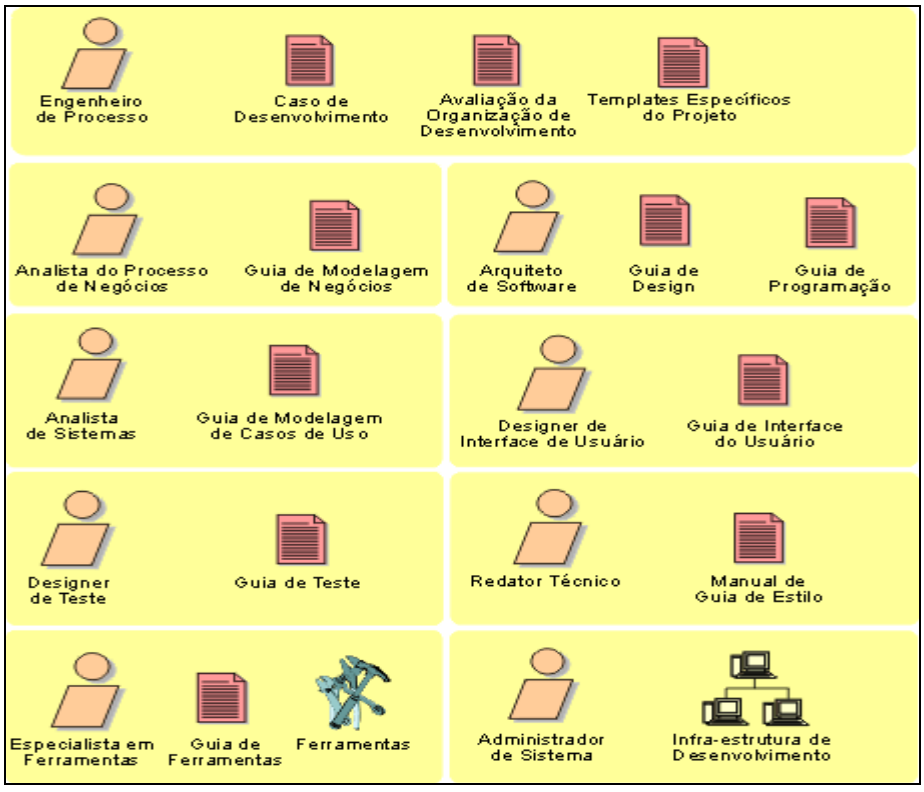

Fonte: Rational (2002)

Figura 26 – Fluxo de trabalho genérico para a disciplina Ambiente




Fonte: Rational (2002)

Figura 27 – Atividades da disciplina Ambiente




Fonte: Rational (2002)

Figura 28 – Artefatos desenvolvidos na disciplina Ambiente

## 2.12 O RUP É UM PROCESSO CONFIGURÁVEL

A finalidade de um projeto de software é gerar um produto. Um processo eficaz permite que o projeto gere um produto que atenda às necessidades de seus envolvidos, dentro do prazo e do orçamento. A chave para um processo eficaz consiste em adaptá-lo para que seja o mais simples possível (RATIONAL, 2002).

O desenvolvedor bem-intencionado pode ter uma extensa lista de itens desejáveis como, por exemplo, métricas, controles, relatórios, etc. Contudo, atividades e artefatos custam tempo e dinheiro. Alguns desses custos, como a interação diária com o conjunto de ferramentas do ambiente, podem estar visíveis ou simplesmente disfarçados na produtividade mais baixa em tarefas padrão. O RUP incentiva a adaptação. Contudo, ela não é uma licença para ignorar o processo como um todo (RATIONAL, 2002).

O RUP é um processo configurável e pode ser ajustado e redimensionado para atender às necessidades de projetos que variam desde pequenas equipes até grandes empresas de desenvolvimento de *software* (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 443).

Uma alternativa para a adaptação do RUP para projetos de pequeno e médio porte é o MyRup, proposto por REIS e BELCHIOR (2004, p. 597), oferecendo um guia simplificado, de fácil entendimento e passo a passo sobre as diversas atividades a serem executadas pela equipe de desenvolvimento durante o projeto.

O MyRup, ao invés de apresentar a seqüência das atividades por disciplina, como o RUP faz, apresenta esta seqüência por fase, mostrando claramente o encadeamento destas em cada momento do projeto. Além disso, foi retirada a disciplina de modelagem de negócio e várias atividades de outras disciplinas para tornar o processo mais leve para pequenos e médios projetos, visando ter o mínimo de atividades do RUP para um projeto garantir a qualidade de desenvolvimento (REIS; BELCHIOR, 2004, p. 599).

## 2.13 TRABALHOS CORRELATOS

Lenzi (1998) apresenta em seu trabalho algumas características de controle de processos, como micro-processamento, comunicação serial, entre outros e aborda conceitos sobre aquisição de dados. Foi desenvolvido um protótipo de controle alimentar suíno usando os conceitos de controle de processos.

No trabalho apresentado por Bork (2003), foi feita uma aplicação prática da metodologia RUP com o auxílio da UML na empresa *Dinamix Software*. É abordada uma visão geral de como funciona a metodologia RUP, sua divisão por fases e disciplinas, a iteração que pode ocorrer nos processos e os artefatos gerados.

# 3 DESENVOLVIMENTO DO TRABALHO

O RUP foi configurado para atender o desenvolvimento do *software* segundo suas características, definindo iterações e confeccionando modelos que realmente agregaram valor a especificação.

Como parâmetro para configuração do RUP, foi seguida a metodologia utilizada por Larman (2004), que enfatiza as disciplinas de Requisitos, Análise e Projeto e Implementação do RUP.

O desenvolvimento foi dividido em três ciclos iterativos, onde o primeiro ciclo corresponde às fases de concepção e elaboração, o segundo ciclo corresponde à fase de construção e o terceiro ciclo à fase de transição.

As fases de concepção e elaboração foram reunidas na mesma iteração pelo fato de ter sido definido, de antemão, que o software seria desenvolvido, não havendo a necessidade de avaliação de riscos, o que torna breve a fase de concepção.

A cada iteração, um fluxo de trabalho, com suas atividades e artefatos, foi definido para orientar o desenvolvimento do projeto. Os artefatos desenvolvidos no projeto, divididos

por disciplina, são apresentados no quadro 1.

| **Disciplina** | **Artefato** |
|---|---|
| Modelagem de Negócio | Regras de Negócio |
| | Modelo de Domínio |
| Requisitos | Modelo de Casos de Uso |
| | Visão |
| | Glossário |
| Análise e Projeto | Realização de Casos de Uso |
| | Modelo de Dados |
| Implementação | Diagrama de Implementação |
| Ambiente | Pasta de Desenvolvimento |

Quadro 1 – Pasta de desenvolvimento do projeto

Para a confecção e organização dos modelos necessários ao projeto, foi utilizada a ferramenta *Enterprise Architect* versão 4.51 da *Sparx Systems*, que suporta a versão 2.0 da UML. O *Enterprise Architect* possibilita também a geração de relatórios em formato HTML e *rich text file* (RTF), detalhados na pasta de desenvolvimento do projeto.

## 3.1 PRIMEIRA ITERAÇÃO

A primeira iteração, que abrangeu as fases de concepção e elaboração, seguiu o fluxo de trabalho apresentado na figura 29.

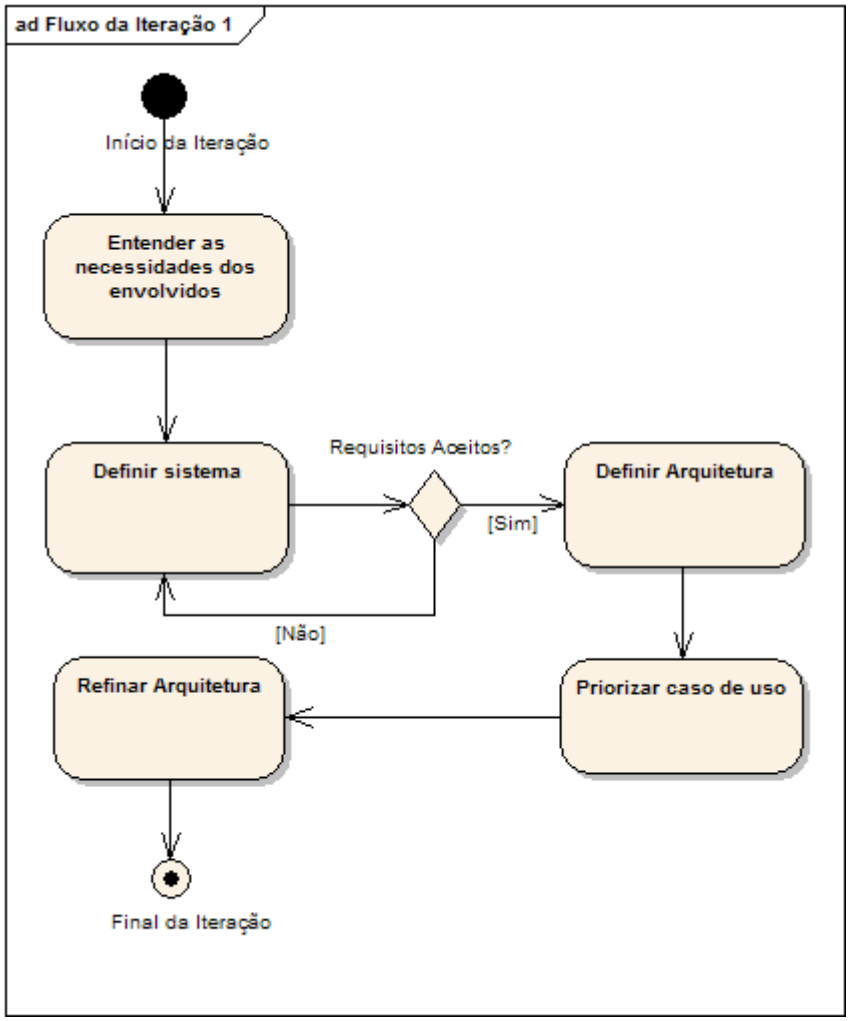



Figura 29 – Fluxo de trabalho da primeira iteração

O detalhamento *Entender necessidades dos envolvidos*, do fluxo de trabalho, trata do primeiro contato com os usuários para o entendimento do problema e investigação dos requisitos. Este contato gerou um documento de visão, mostrado no apêndice B, que descreve o que de mais relevante descobriu-se sobre como o protótipo deverá funcionar, a visão inicial do cliente, suas principais preocupações, e não contempla detalhes, que serão melhor estruturados em outros modelos, como requisitos ou casos de uso. Também foram definidas as regras de negócio, apresentadas no apêndice C e o glossário, que registra e esclarece termos usados na documentação. O glossário é apresentado no apêndice A.

No processo *Definir sistema* foram listados os requisitos do sistema, definidos os casos

de uso, seus cenários e seus diagramas de atividades, bem como, um diagrama de implementação, definindo a estrutura de persistência e negócios das classes do projeto.

### 3.1.1 Requisitos principais do problema a ser trabalhado

Os requisitos funcionais têm uma forte influência sobre a arquitetura do *software.* Por isso, requisitos definidos com qualidade originam uma melhor compreensão do problema e um produto final mais adequado às necessidades do cliente.

Com a análise do modelo de visão, os seguintes requisitos funcionais foram propostos:

a) configuração geral dos controladores;

b) configuração dos parâmetros de calibração de sensores e instrumentos;

c) configuração dos parâmetros de comunicação;

d) configuração e construção de interface dos controladores;

e) definição das estratégias de controle de processo industrial;

f) consulta do sinótico de forma visual;

g) armazenamento do histórico de erros de comunicação na porta serial.

### 3.1.2 Requisitos não funcionais

Alguns requisitos não funcionais foram propostos:

a) Armazenamento dos dados coletados em banco de dados;

b) Interface gráfica em *windows*.

É essencial que as leituras de sinais enviadas pelos sensores sejam armazenadas em memória secundária para que futuramente sejam consultados, fornecendo um histórico confiável de cada sensor.

As informações obtidas *on line* serão apresentadas através de um sinótico, que será ativado conforme solicitação do usuário.

Neste momento é importante que seja feito um *workshop* com o cliente, apresentando a visão inicial do problema e os requisitos propostos, para obter a aprovação dos mesmos e minimizar visões distorcidas ou errôneas do problema. Com esta aprovação, o problema estará definido e pronto para ser especificado. No *workshop* realizado com o cliente, os requisitos citados foram validados, dando início à execução do projeto.

### 3.1.3 Casos de uso

Os casos de uso captam o comportamento pretendido de um sistema ou parte de um sistema sem que seja necessário especificar como esse comportamento é implementado (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 217).

Considerando-se que o RUP é orientado por casos de uso e sendo o desenvolvimento baseado nos mesmos, torna-se fundamental que eles sejam bem-estruturados e que definam claramente o comportamento do sistema pretendido pelo usuário.

Para o protótipo sugerido, foram especificados oito casos de uso:

a) realizar a configuração geral dos controladores (*Start Up*);

b) cadastrar equipamentos sensores e atuadores;

c) configurar parâmetros de comunicação;

d) realizar a configuração e construção de interface dos controladores;

e) definir estratégias de controle de processo industrial;

f) consultar sinótico;

g) buscar dados do controlador;

h) enviar dados para o controlador.

O papel desempenhado pelos usuários dos casos de uso quando interagem com esses casos é representado por atores. Os atores podem representar seres humanos, dispositivos de *hardware* ou outros sistemas.

Foram definidos três atores para interação com os casos de uso especificados:

a) operador;

b) administrador;

c) CP1000(CPU).

Os casos de uso e atores foram representados através de um modelo de casos de uso, onde também são mostrados os relacionamentos entre eles. O modelo de casos de uso é mostrado na figura 30.

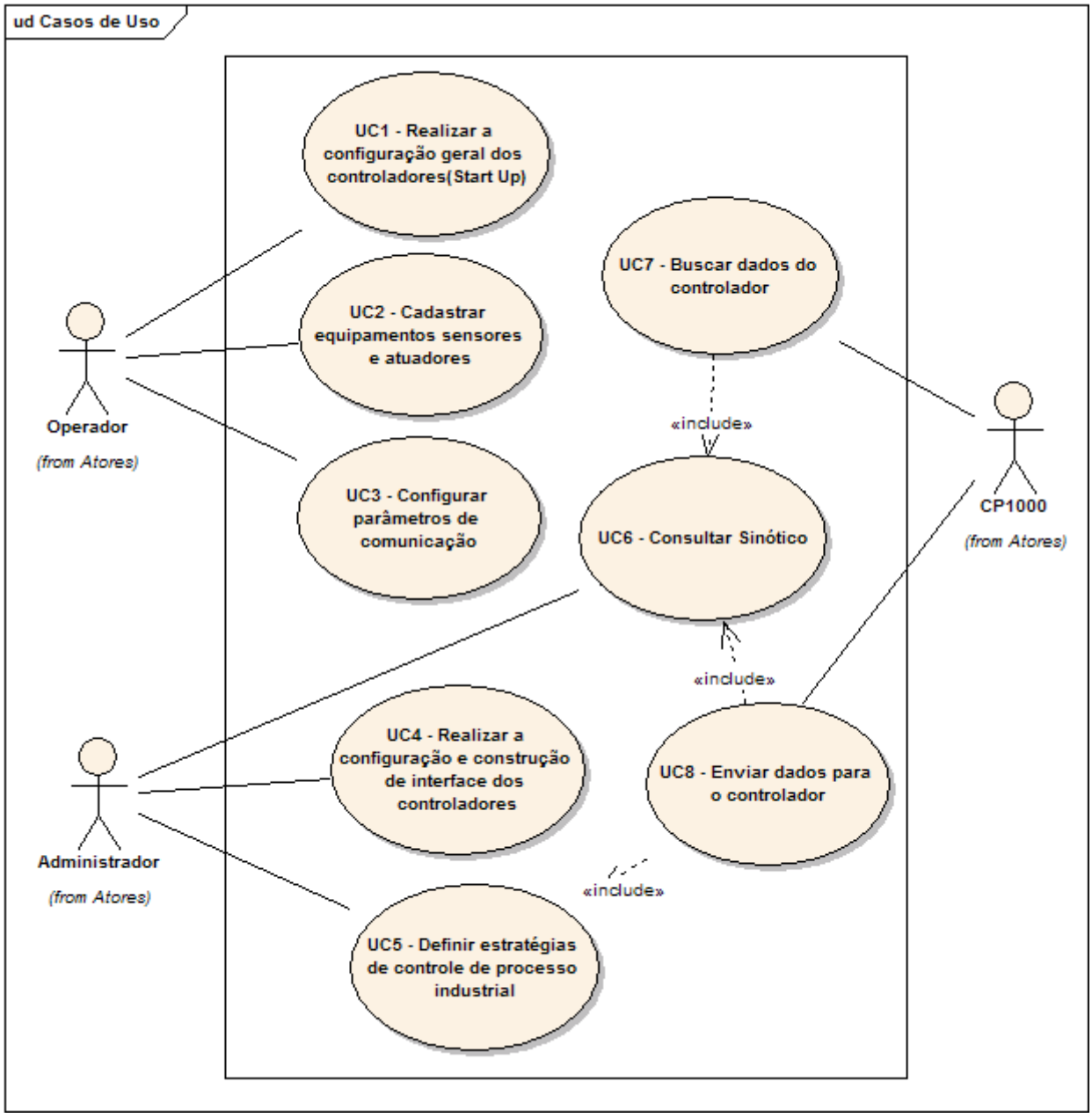



Figura 30 – Modelo de casos de uso

Cada caso de uso conta com um cenário próprio, representando o fluxo principal e alternativo, descrito passo a passo. Um cenário é uma seqüência específica de ações e interações entre atores e o sistema.

No detalhamento dos casos de uso foram desenhados diagramas de atividades para melhor visualização do fluxo principal definido em cada cenário. Nos casos de uso “buscar dados do controlador” e “enviar dados para o controlador”, os cenários e diagramas de atividades não foram desenhados por terem baixa complexidade.

No apêndice D é mostrado um diagrama de rastreabilidade entre os requisitos funcionais e os casos de uso do sistema.

### 3.1.3.1 UC1 – Realizar a configuração geral dos controladores (*Start Up*)

Devem ser especificados os endereços de cada controlador na rede, definidos os parâmetros dos canais de entrada e saída e o tempo em que o sistema atualizará o sinal enviado pelo canal.

No quadro 2 e na figura 31 são mostrados o cenário do caso de uso e o diagrama de atividades correspondente respectivamente.

| | |
|---|---|
| Fluxo Principal : | 1. O operador entra na tela de configurações(*Start Up*).<br>2. O operador cadastra o controlador a ser configurado.<br>3. O operador informa o modelo do controlador, sua descrição, tempo de *scan* e seu endereço na rede .<br>4. O operador informa a descrição de cada canal do controlador, tempo de *scan*, tipo de sinal e quais estão ativos.<br>5. O operador finaliza a configuração(*Start Up*). |
| Fluxo Alternativo : | 2a. O controlador já está cadastrado.<br>2a.1. O operador seleciona o controlador.<br>2a.2. Retorna ao fluxo principal no passo 2. |

Quadro 2 – Cenário do caso de uso “realizar a configuração geral dos controladores”

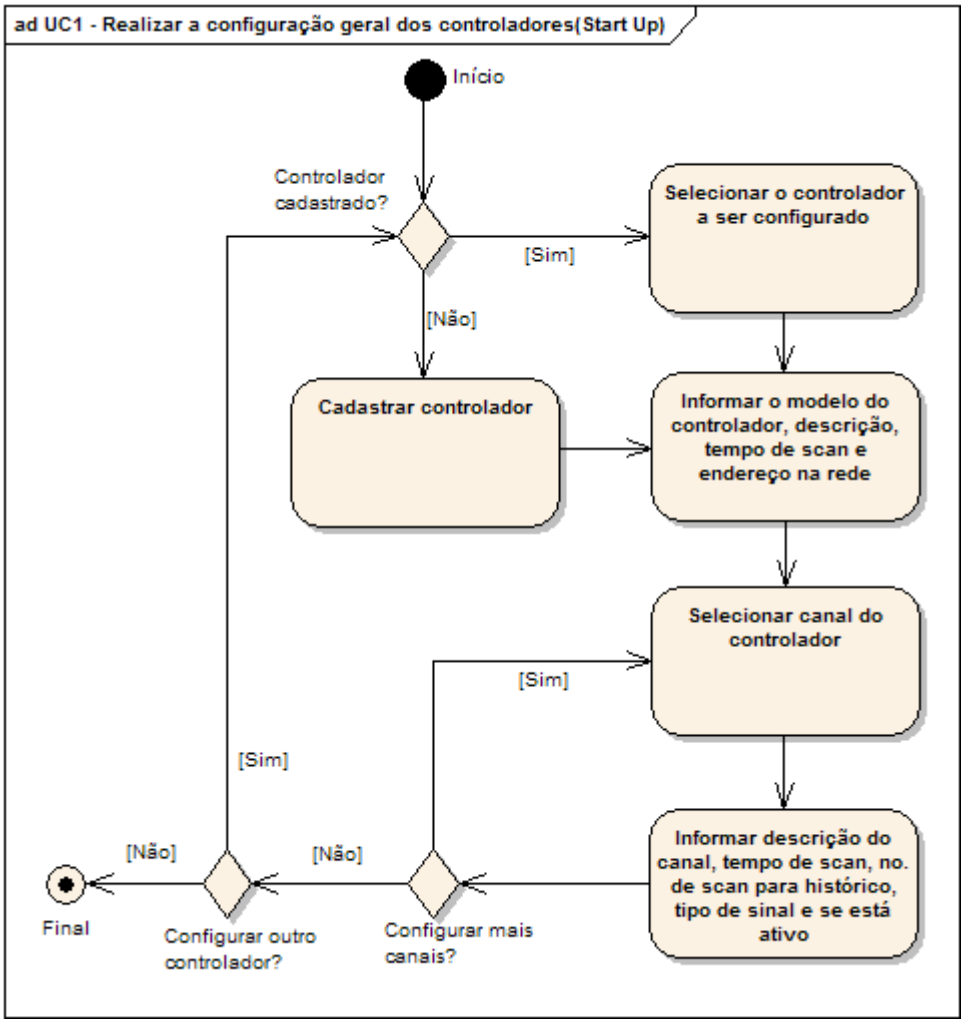



Figura 31 – diagrama de atividades para “realizar a configuração geral dos controladores”

### 3.1.3.2 UC2 – Cadastrar equipamentos sensores e atuadores

Devem ser cadastrados os equipamentos, sensores e atuadores, que farão parte do processo industrial, informando sua descrição, uma descrição reduzida, que aparecerá no visor da CPU, a relação entre o sinal máximo e mínimo recebidos da CPU, com os respectivos valores da unidade de medição utilizada. No caso de um sensor, também deve ser informado a cada quantas leituras da CPU o sinal do sensor deve ser armazenado no banco de dados.

No quadro 3 e na figura 32 são mostrados o cenário do caso de uso e o diagrama de atividades correspondente respectivamente.

| | |
|---|---|
| Fluxo Principal: | 1. O operador entra na tela de cadastro.<br>2. O operador cadastra um novo sensor.<br>3. O operador informa o canal de entrada e o controlador no qual o equipamento está ligado.<br>4. O operador informa a descrição, descrição reduzida, a unidade de medição, o valor mínimo e máximo para a relação entre o sinal de entrada do canal e a unidade de medição.<br>5. O operador finaliza o cadastro. |
| Fluxo Alternativo: | 2a. O sensor já está cadastrado.<br>2a.1. O operador seleciona o sensor.<br>2a.2. O operador altera as informações do sensor.<br>2a.3. Retorna ao passo 2 do fluxo principal.<br><br>2b. O cadastro é de um atuador.<br>2b.1. O operador cadastra um novo atuador.<br>2b.2. O operador informa o canal de saída e o controlador no qual o equipamento está ligado.<br>2b.3. Retorna ao passo 4 do fluxo principal.<br><br>2c. O cadastro é de um atuador e o mesmo já está cadastrado.<br>2c.1. O operador seleciona o atuador.<br>2c.2. O operador altera as informações do atuador.<br>2c.3. Retorna ao passo 2 do fluxo principal. |

Quadro 3 – Cenário de “cadastrar equipamentos sensores e atuadores”.

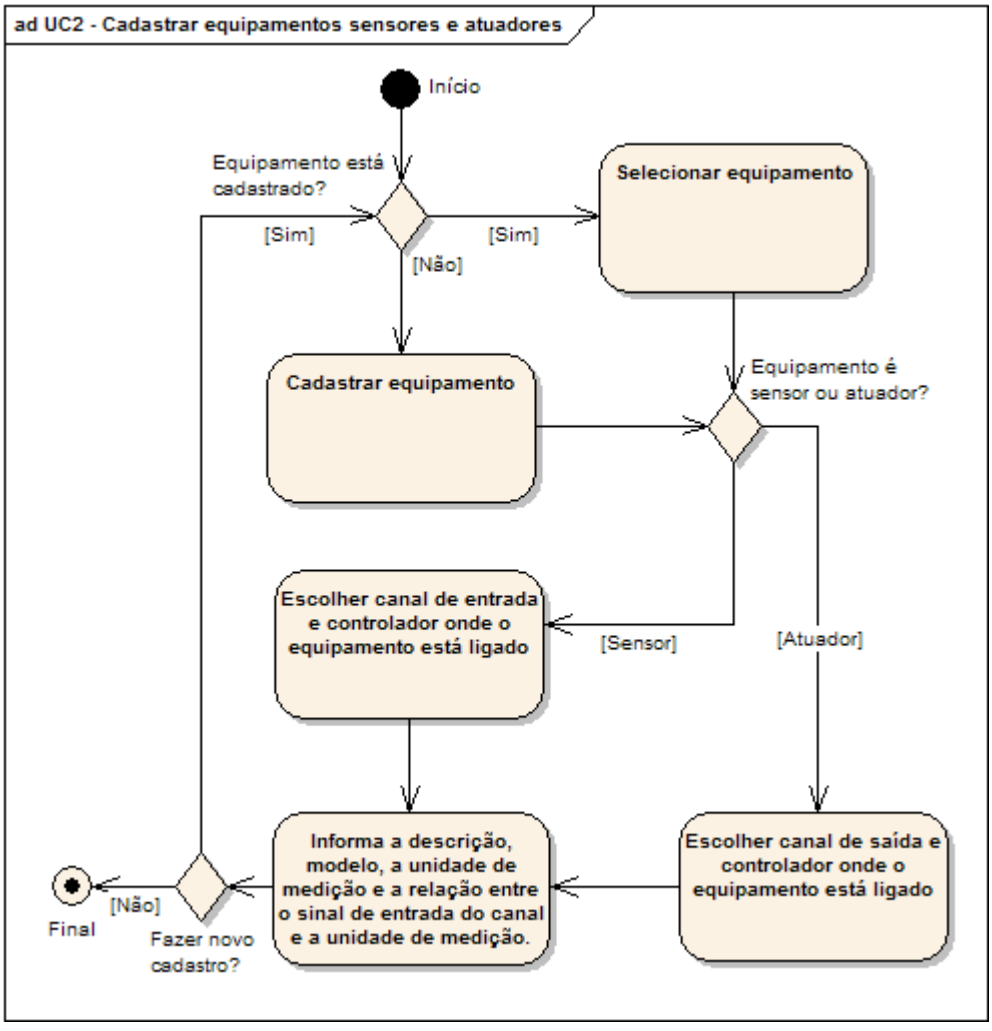



Figura 32 – Diagrama de atividades para “cadastrar equipamentos sensores e atuadores”.

### 3.1.3.3 UC3 – Configurar parâmetros de comunicação

Deve ser definida a porta serial conectada, a velocidade de comunicação, o tempo limite para "*time out*" e o número de "*time out*" para gerar uma falha de comunicação.

No quadro 4 mostra o cenário do caso de uso. Para o caso de uso mencionado não foi desenhado o diagrama de atividades por se tratar de um caso de uso com baixa complexidade.

| | |
|---|---|
| Fluxo Principal : | 1. O operador entra na tela de configurações.<br>2. O operador informa os parâmetros para comunicação com o CP1000: porta serial, velocidade, *time outs*/falha e tempo para *time out*.<br>3. O operador finaliza a configuração. |

Quadro 4 – Cenário de “configurar parâmetros de comunicação”.

### 3.1.3.4 UC4 – Realizar a configuração e construção de interface dos controladores

Através de um design visual, deve ser possível construir um sinótico da fábrica, distribuindo de forma organizada a representação visual de cada equipamento a ser consultado. Vários componentes visuais devem estar à disposição do usuário para representar os sinais lidos e mensagens apresentadas.

No quadro 5 e na figura 33 são mostrados o cenário do caso de uso e o diagrama de atividades correspondente respectivamente.

| | |
|---|---|
| Fluxo Principal : | 1. O Administrador entra na tela de *design*.<br>2. O Administrador cria um sinótico para uma determinada planta da fábrica.<br>3. O Administrador arrasta para o sinótico o componente visual desejado.<br>4. O Administrador liga o componente a um canal cadastrado.<br>5. O Administrador faz ligações visuais entre os componentes.<br>6. O Administrador salva o sinótico.<br>7. O Adminstrador finaliza o *design*. |
| Fluxo Alternativo : | 2a. Já existe sinótico para a planta da fábrica desejada.<br>2a.1. O Administrador seleciona o sinótico.<br>2a.2. Retorna ao fluxo principal no passo 2.<br><br>3a. O Administrador deseja alterar os componentes ligados ao sinótico.<br>3a.1. O Administrador altera os componentes ligados ao sinótico.<br>3a.2. Retorna ao fluxo principal no passo 6. |

Quadro 5 – Cenário de “realizar a configuração e construção de interface dos controladores”

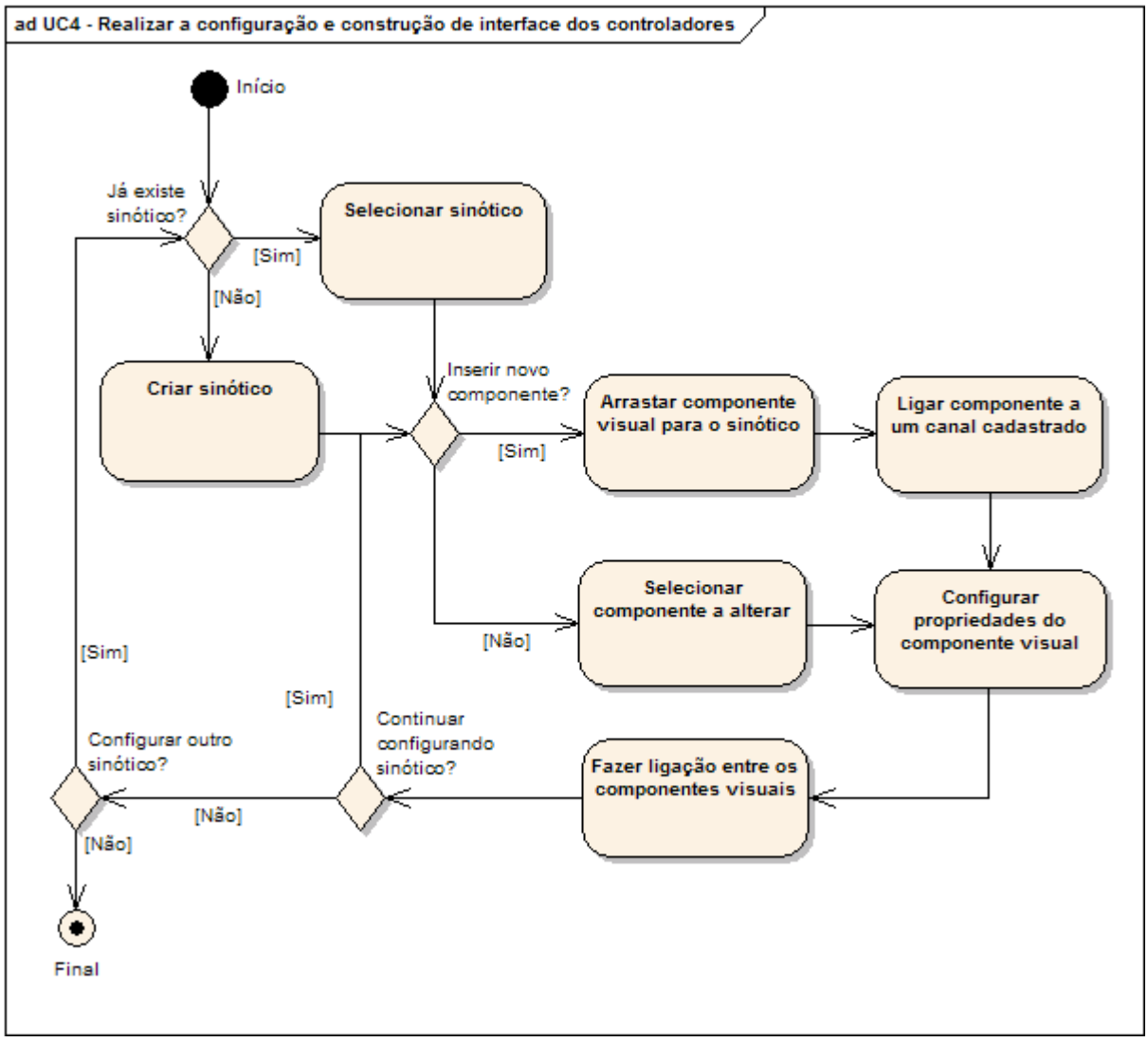



Figura 33 – Diagrama de atividades: “realizar a configuração e construção de interface dos controladores”

### 3.1.3.5 UC5 – Definir estratégias de controle de processo industrial

Definir quais atuadores serão manipulados de acordo com as respectivas leituras dos sensores. Tendo definido o valor em que o sinal do sensor deve ser mantido (*Set Point*), a própria CPU fará este controle.

No quadro 6 e na figura 34 são mostrados o cenário do caso de uso e o diagrama de atividades correspondente respectivamente.

| | | |
|---|---|---|
| Fluxo Principal : | Principal | 1. O Administrador entra na tela de estratégias de controle.<br>2. O Administrador seleciona um equipamento(sensor).<br>3. O Administrador seleciona um equipamento(atuador) para interagir com o equipamento(sensor).<br>4. O Administrador define o valor de set point que deve ser mantido no equipamento (sensor).<br>5. Os valores de set point são enviados para o CP1000.<br>6. O Administrador finaliza a configuração de estratégias. |
| Fluxo Alternativo : | Alternativo | 2a O Administrador seleciona um equipamento (sensor).<br>2a.1. O Administrador define a mensagem para o valor mínimo do sinal do equipamento.<br>2a.2. O Administrador define a mensagem para o valor máximo do sinal do equipamento.<br>2a.3. Retorna ao fluxo principal no passo 2. |
| Fluxo de Exceção | Alternativo | 5a Erro de comunicação na porta serial<br>5a.1 É gravado o momento e o erro gerado na porta serial<br>5a.2 Retorna ao fluxo principal no passo 5. |

Quadro 6 – Cenário de “definir estratégias de controle de processo industrial”

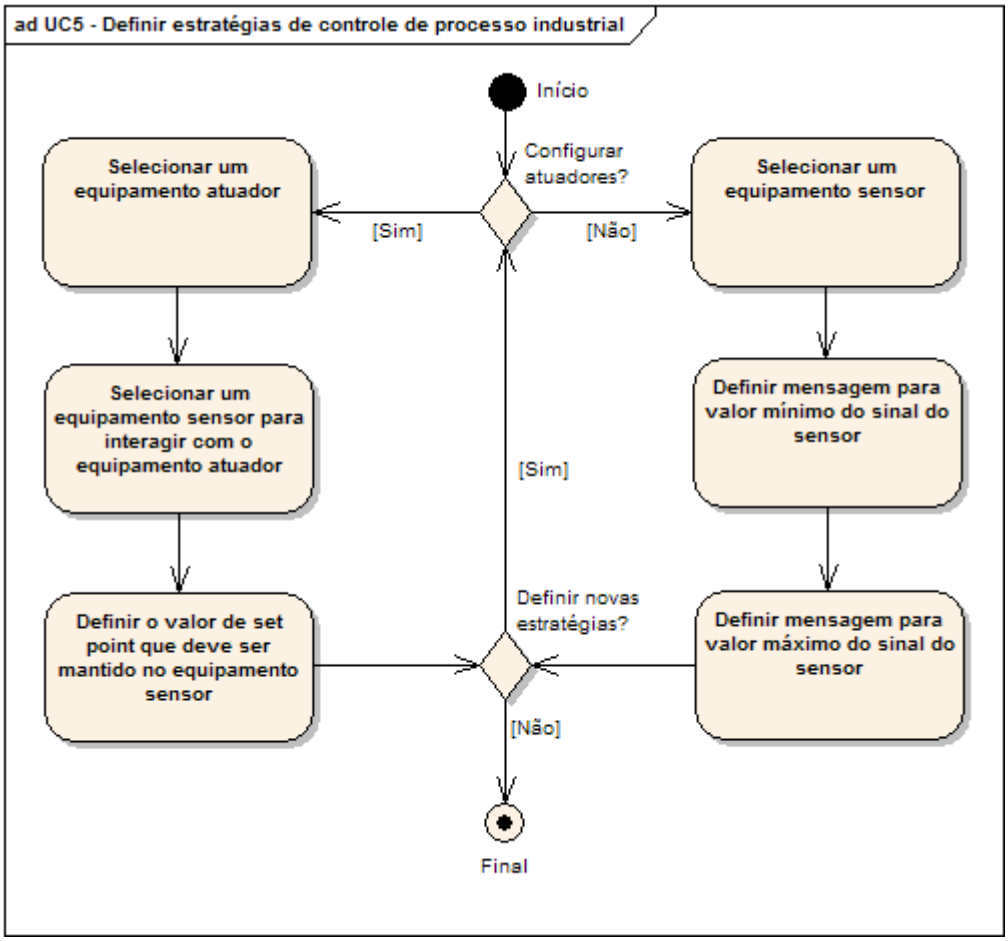



Figura 34 – Diagrama de atividades: “definir estratégias de controle de processo industrial”

### 3.1.3.6 UC6 – Consultar sinótico

De acordo com o sinótico montado pelo usuário os valores de sinal dos controladores serão apresentados na tela de forma visual, atualizados em um intervalo de tempo pré-definido.

No quadro 7 e na figura 35 são mostrados o cenário do caso de uso e o diagrama de atividades correspondente respectivamente.

| | | |
|---|---|---|
| Fluxo Principal : | Principal | 1. O Administrador entra na tela de consulta de sinóticos.<br>2. O Administrador ativa um sinótico.<br>3. O Administrador ativa estratégias de controle.<br>4. O Sistema busca, no tempo determinado, o sinal dos canais lidos pelo CP1000.<br>5. O Sistema atualiza os componentes visuais, apresentando os valores lidos no sinótico.<br>6. O Sistema grava no histórico do sensor os valores lidos.<br>6. O Administrador finaliza a consulta de sinóticos. |
| Fluxo de Exceção : | Alternativo | 4a Erro de comunicação na porta serial<br>4a.1 É gravado o momento e o erro gerado na porta serial<br>4a.2 Retorna ao fluxo principal no passo 4. |

Quadro 7 – Cenário de “consultar sinótico”

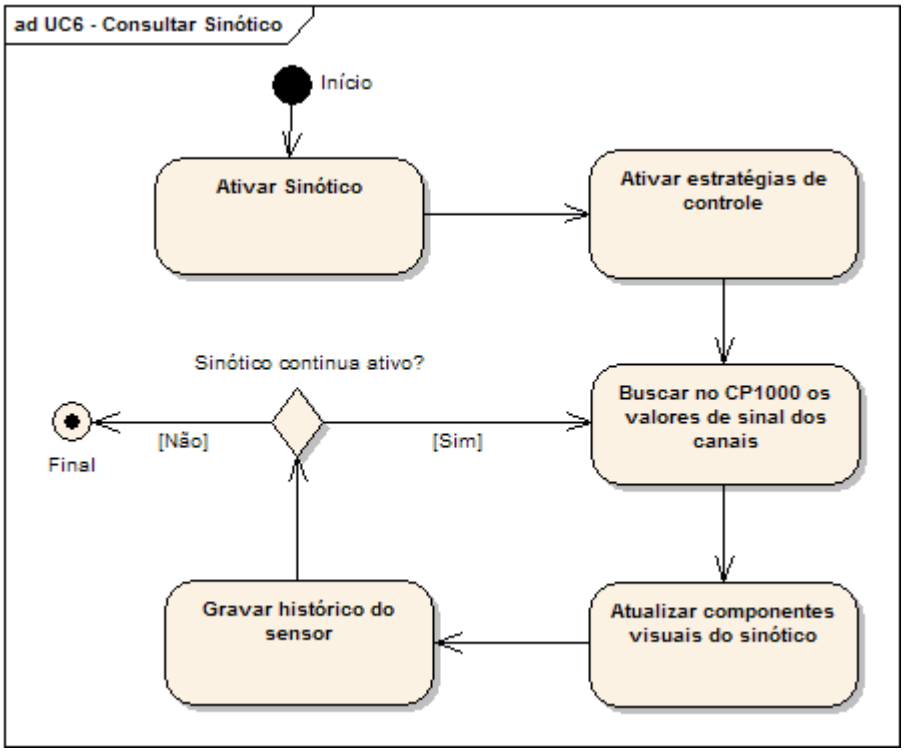



Figura 35 – Diagrama de atividades para “consultar sinótico”

#### 3.1.3.7 UC7 – Buscar dados do controlador

A comunicação com o *CP1000* será feita através de um componente fornecido pelo fabricante, que faz a leitura dos canais ativos em um intervalo de tempo definido para o controlador. Os parâmetros de comunicação servirão para que o componente tenha as configurações necessárias para leitura das informações do controlador. Deve ser gerado um histórico com erros de comunicação com a porta serial.

#### 3.1.3.8 UC8 – Enviar dados para o controlador

O envio de dados para o *CP1000* será feita através de um componente fornecido pelo fabricante, atualizando uma propriedade do componente com o valor de saída. Os parâmetros de comunicação servirão para que o componente tenha as configurações necessárias para comunicação com o controlador. Deve ser gerado um histórico com erros de comunicação com a porta serial.

### 3.1.4 Modelo de domínio conceitual

De acordo com os casos de uso, seus cenários e diagramas de atividades, algumas classes conceituais foram propostas. Estas classes descrevem objetos do mundo real em um domínio do problema, e não componentes de *software*.

O modelo de domínio é uma representação visual dessas classes conceituais significativas, com suas associações e atributos.

Em um processo de desenvolvimento com várias iterações na fase de elaboração, as classes conceituais são identificadas à medida que as iterações acontecem. No

desenvolvimento deste trabalho todas as classes conceituais foram identificadas na mesma iteração. O modelo de domínio geral é mostrado na figura 36.

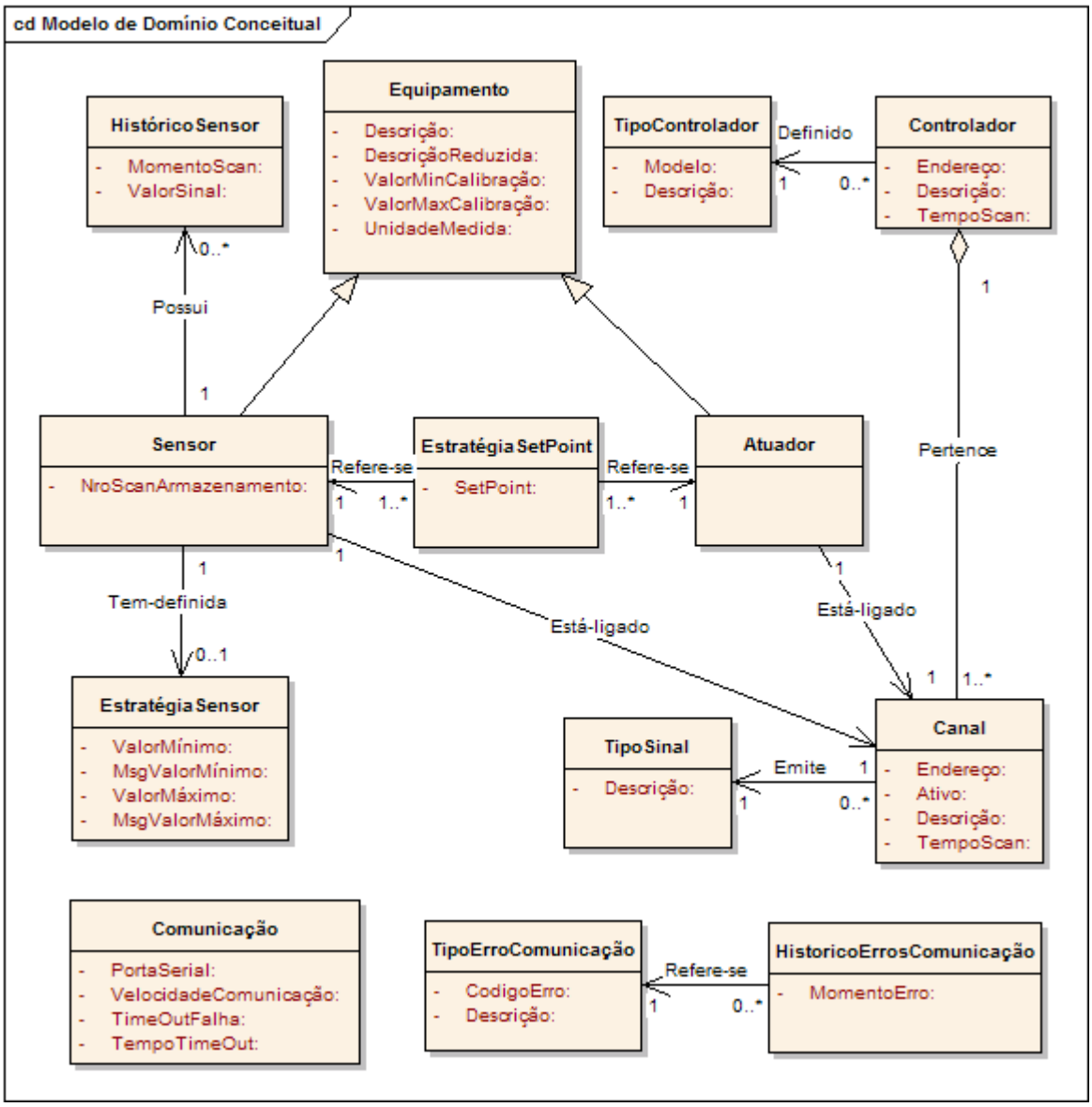



Figura 36 – Modelo de Domínio Conceitual Geral

Analisando o modelo, pode-se verificar que existe uma classe *Equipamento*, que armazena as informações relevantes de cada equipamento do processo industrial. Essa classe é especificada em duas sub-classes: *Sensor*, usada para armazenar informações específicas dos equipamentos de leitura e *Atuador*, onde são guardadas informações sobre equipamentos que atuam sobre o processo industrial.

Pelo fato do sensor possuir ou não estratégias definidas para ele, foi criada a classe

*EstratégiaSensor*, que armazena essas informações e está ligada à classe *Sensor*. A classe *HistoricoSensor* traz todas as leituras de sinal feitas no sensor.

As estratégias de *set point* definidas entre o equipamento sensor e o equipamento atuador, estão armazenadas na classe *EstrategiaSetPoint*.

Tanto a classe *Sensor* quanto *Atuador*, estão ligadas à classe *Canal*, que emite um tipo de sinal definido em *TipoSinal* . A classe *Canal* está fortemente ligada à classe *Controlador*, pois canais pertencem a um controlador. A classe *TipoControlador* define o tipo do controlador existente.

A classe *TipoErroComunicação* traz uma relação de possíveis erros de comunicação na conexão com a porta serial. O histórico de erros em relação ao tempo é armazenado na classe *HistoricoErrosComunicação*.

Para armazenar as informações pertinentes à comunicação serial, foi criada a classe *Comunicação*.

O Modelo de Domínio do gráfico do sinótico é mostrado na figura 37.

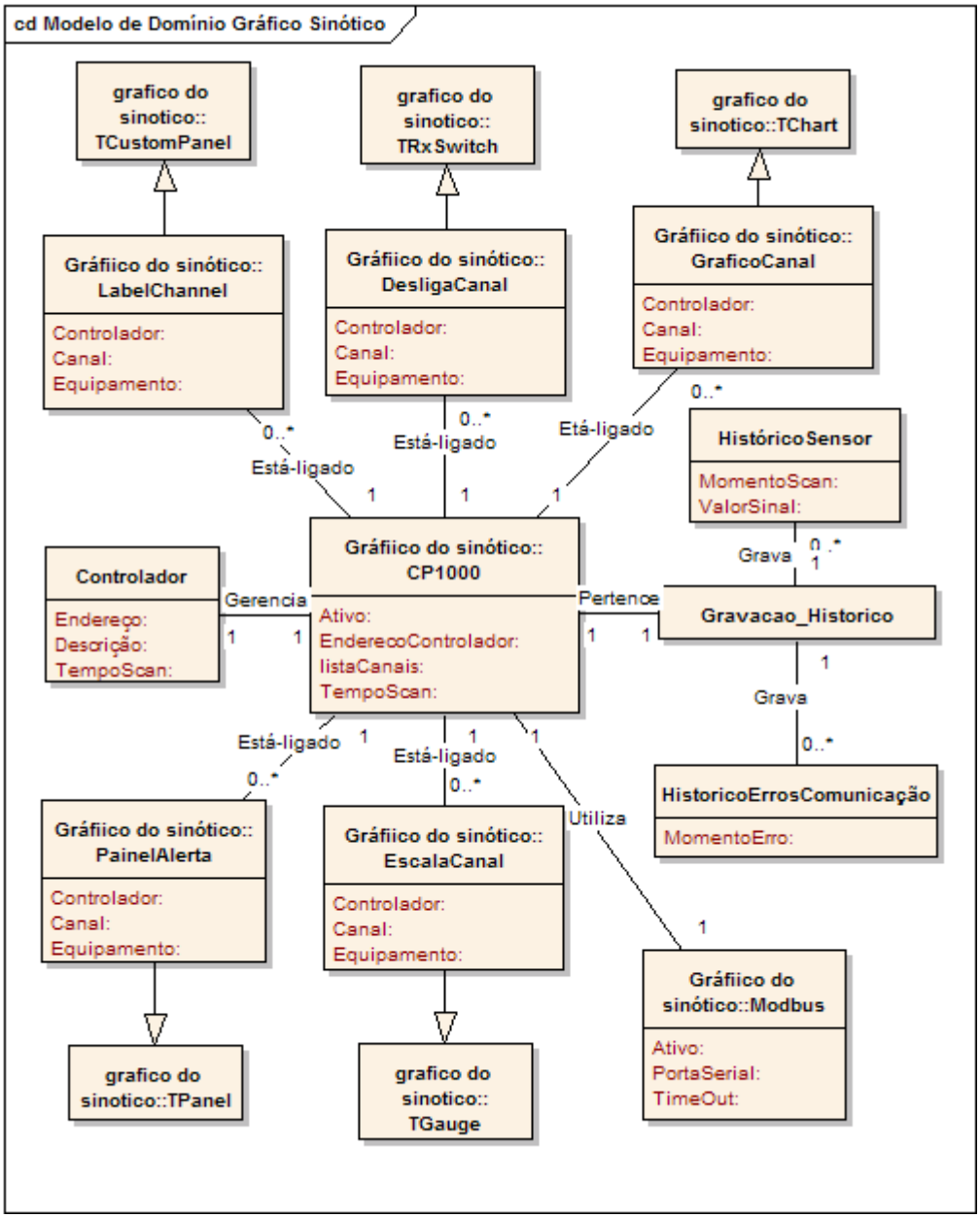



Figura 37 – Modelo de Domínio Conceitual do gráfico do sinótico

Para a construção do sinótico, foram modeladas classes de componentes que herdam as propriedades de componentes da *Visual Component Library* (VCL), do *Delphi*. O *CP1000* e o *Modbus* são componentes para *Delphi* 7, fornecidos pelo fabricante do *CP1000*. O *Modbus* é responsável pela comunicação serial com o módulo *CP1000*, através do protocolo *Modbus* e deve ser instanciado apenas uma vez para cada porta serial, para que não haja conflito na porta.

O *CP1000* faz a verificação dos canais de um controlador, atualizando os valores de

sinais com o módulo *CP1000*, em um intervalo de tempo definido na propriedade *TempoScan*, disparando o evento *OnMudaSinal*. É necessária uma instância da classe *CP1000* para cada controlador configurado no sistema. Cada componente que herda propriedades de componentes da VCL, tem um evento ligado ao evento *OnMudaSinal* da classe *CP1000*, que quando é disparado, atualiza a interface do componente, mostrando informações referentes ao canal que está ligado.

A classe *GravacaoHistorico* foi modelada para a gravação do histórico de erros de comunicação, gerado no evento *OnErrorModbus* do componente *Modbus*, e do histórico do sensor, gerado na leitura de sinal feita pela classe *CP1000*.

As classes conceituais deram origem às classes do sistema, que fazem parte do domínio da solução.

### 3.1.5 Estrutura da Implementação

Neste ponto do desenvolvimento discutiu-se a arquitetura macro do *software*, definindo-se por uma organização em camadas, utilizando a definição mais comum para arquiteturas deste tipo: aplicação (apresentação), negócio, persistência (dados). Através do diagrama de componentes mostrado na figura 38, são apresentadas as camadas de *software* a serem implementadas no código fonte.

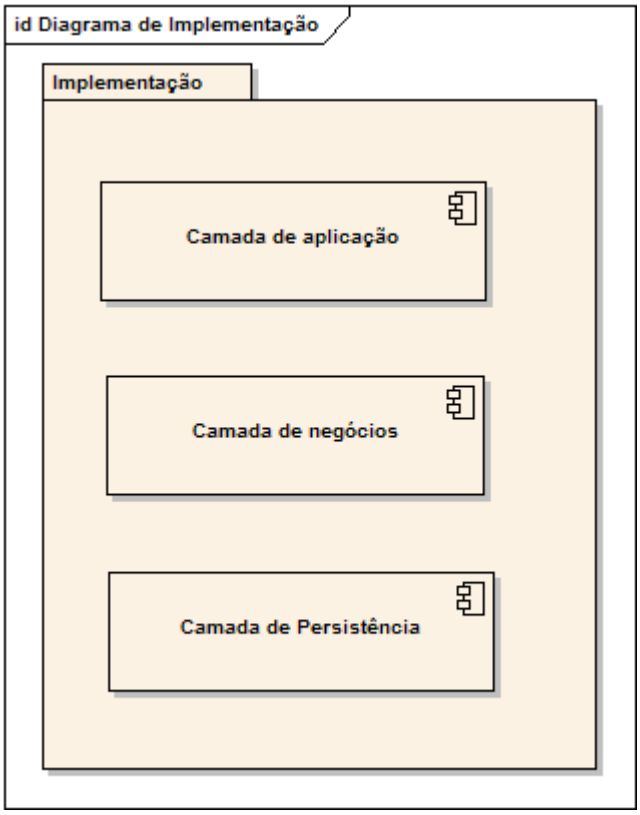



Figura 38 – Diagrama de Implementação

## 3.2 SEGUNDA ITERAÇÃO

A segunda iteração abrange a fase de construção do RUP. Como já se tem os requisitos compreendidos e a espinha dorsal do sistema construída, é hora de escrever o código e documentá-lo, montar interfaces e fazer os testes do projeto. A segunda iteração seguiu o fluxo de trabalho apresentado na figura 39.

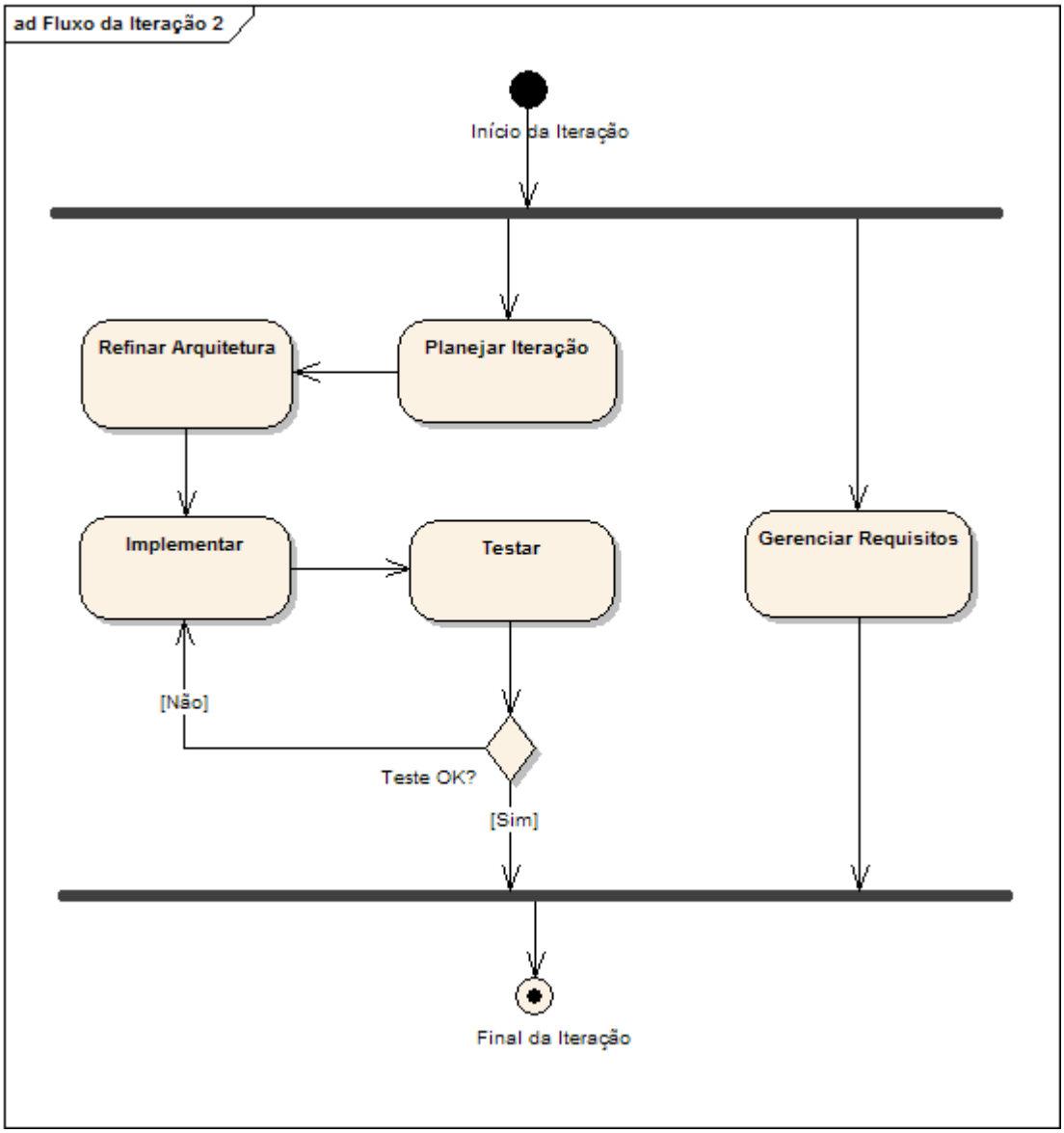



Figura 39 – Fluxo de trabalho para a segunda iteração

Em *Refinar Arquitetura* foram feitas as realizações dos casos de uso, através de diagramas de seqüência, definidas as classes e o modelo de dados do projeto. No processo *Implementar* foram gerados os códigos fontes para as classes e o banco de dados a partir do modelo definido. Em *Testar* foram aplicados testes empíricos para validar o sistema, sem seguir o rigor da disciplina de testes do RUP.

### 3.2.1 Diagramas de interação

Com a definição das classes do sistema, suas associações e seus atributos, através do modelo de domínio conceitual, o próximo passo é definir as operações executadas em cada classe. Para isso, foram montados diagramas de interações, que identificam as mensagens trocadas entre as classes.

Foi escolhido o diagrama de seqüência para esta modelagem por dar ênfase à ordem temporal das mensagens e por ser uma notação mais simples e de mais fácil entendimento.

Os diagramas de seqüência são baseados nos casos de uso e são também chamados de realizações dos casos de uso. Essas realizações fazem parte do modelo de projeto.

Nas figuras 40, 41 e 42, são mostrados os diagramas de seqüência dos principais casos de uso do projeto.

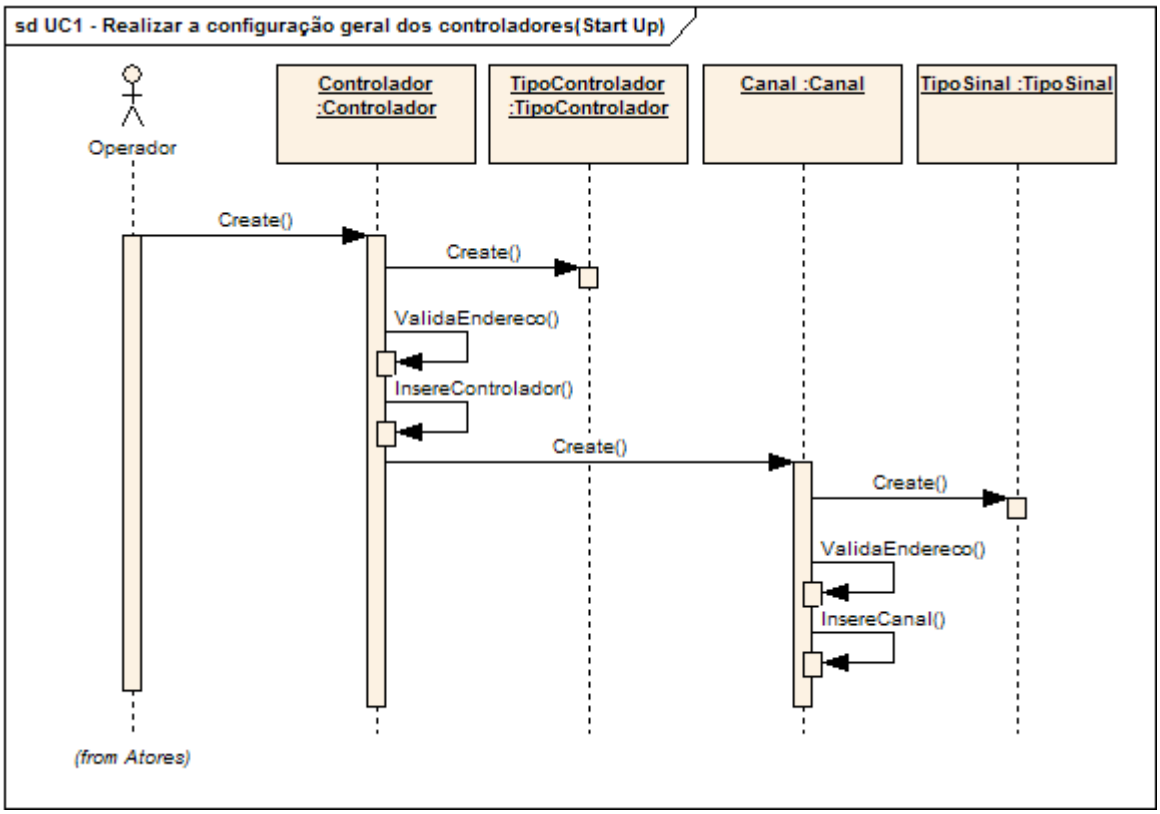



Figura 40 – Diagrama de sequência: “realizar a configuração geral dos controladores(*Start Up*)”

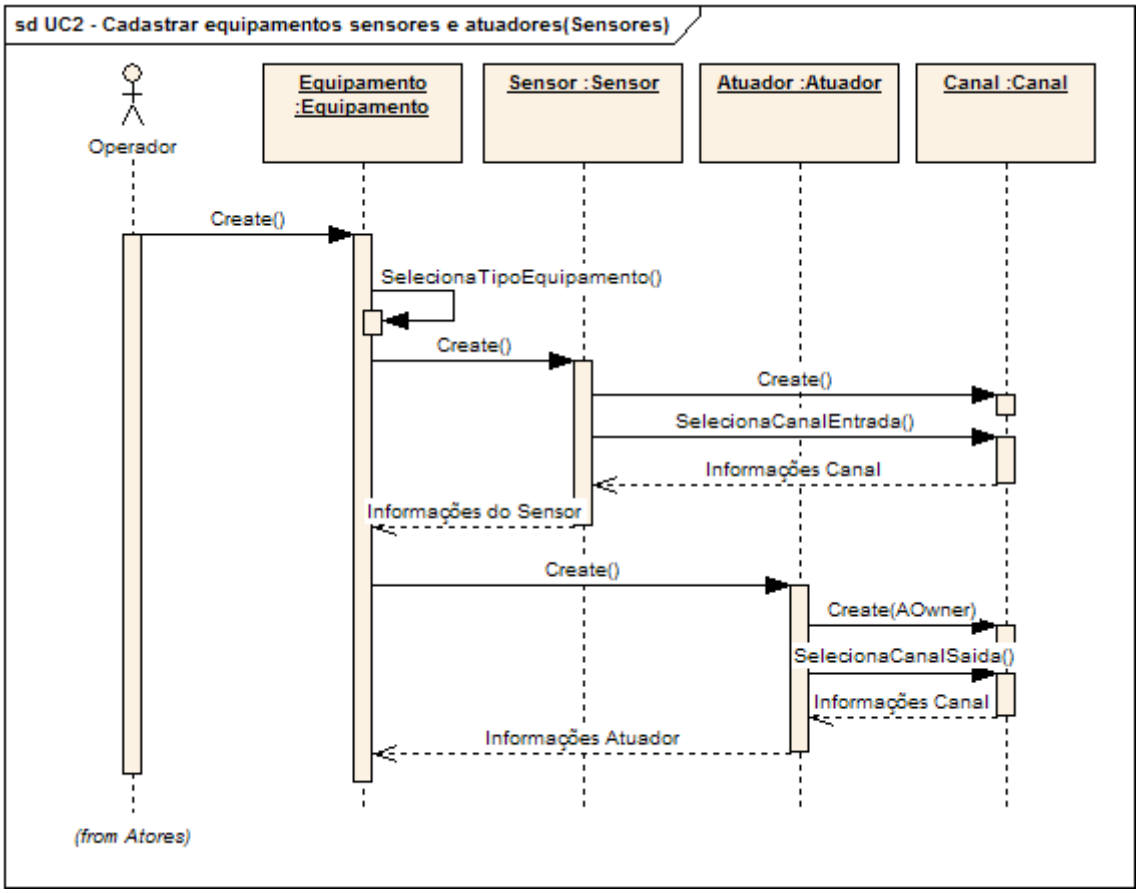



Figura 41 – Diagrama de seqüência: “cadastrar equipamentos sensores e atuadores”

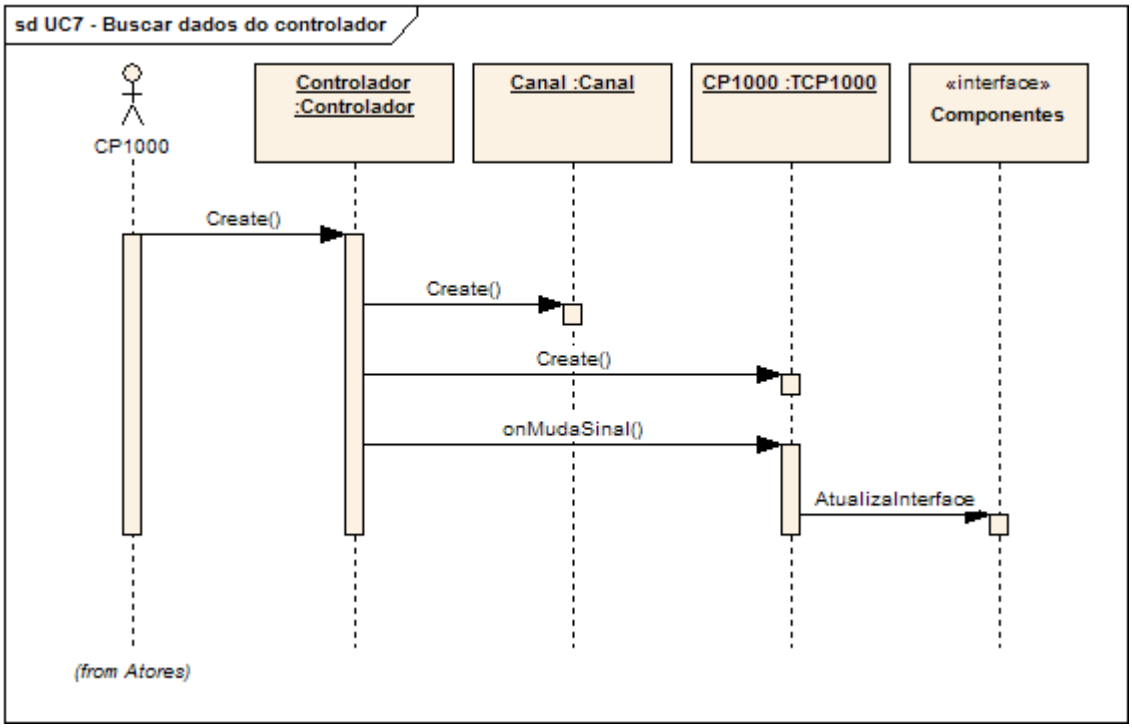



Figura 42 – Diagrama de seqüência: “buscar dados do controlador”

### 3.2.2 Diagrama de classes

Os diagramas de classes mostram um conjunto de classes e seus relacionamentos, e são essenciais na modelagem de sistemas orientados a objetos. Eles são uma apresentação gráfica de um único aspecto da visão estática do projeto de um sistema e contém informações essenciais à compreensão desse aspecto.

Três diagramas de classe desenhados para este projeto mostram aspectos importantes para seu desenvolvimento. O primeiro diagrama mostra a classe de comunicação e as classes responsáveis pela organização dos erros de comunicação, mostrado na figura 43. O segundo diagrama mostra as classes envolvidas na organização geral do sistema, originadas do modelo de domínio conceitual, e é mostrado na figura 44. O terceiro diagrama mostra as classes envolvidas na construção do sinótico referente à planta de uma fábrica, mostrado na figura 45.

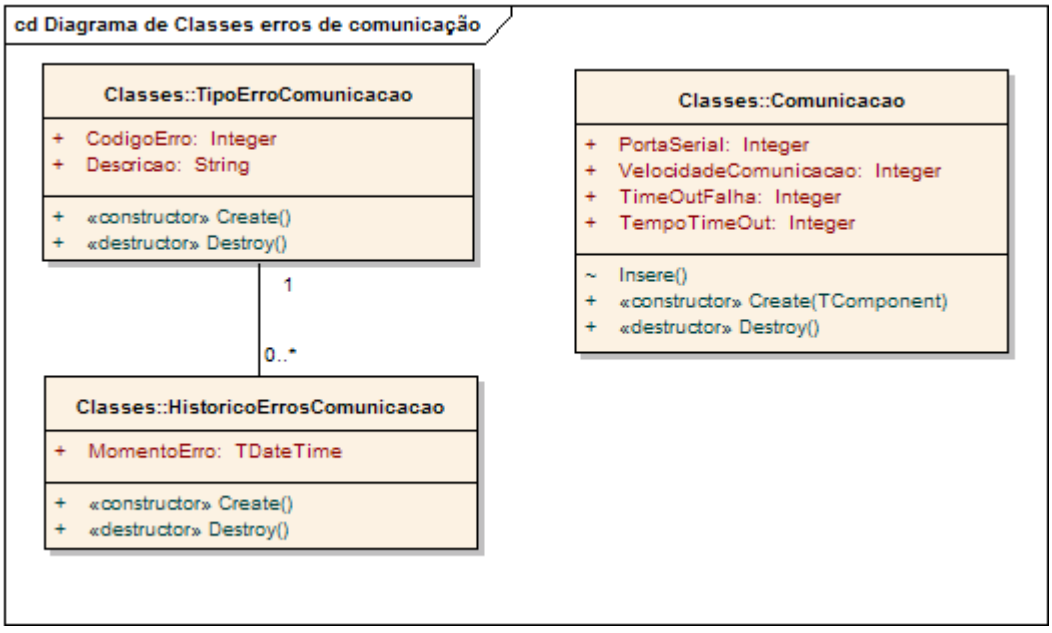



Figura 43 – Diagrama de classes de erros de comunicação

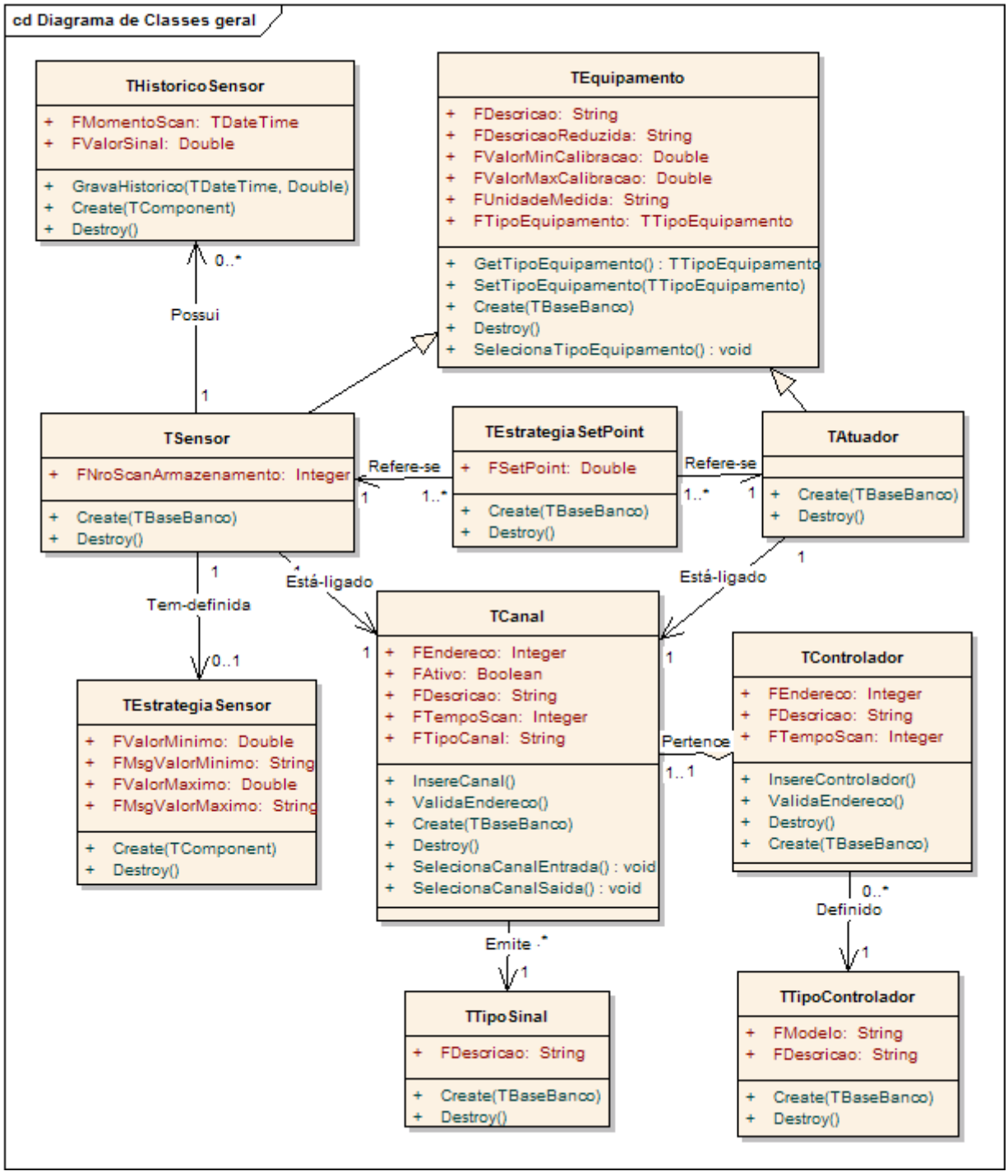



Figura 44 – Diagrama de classes geral do sistema

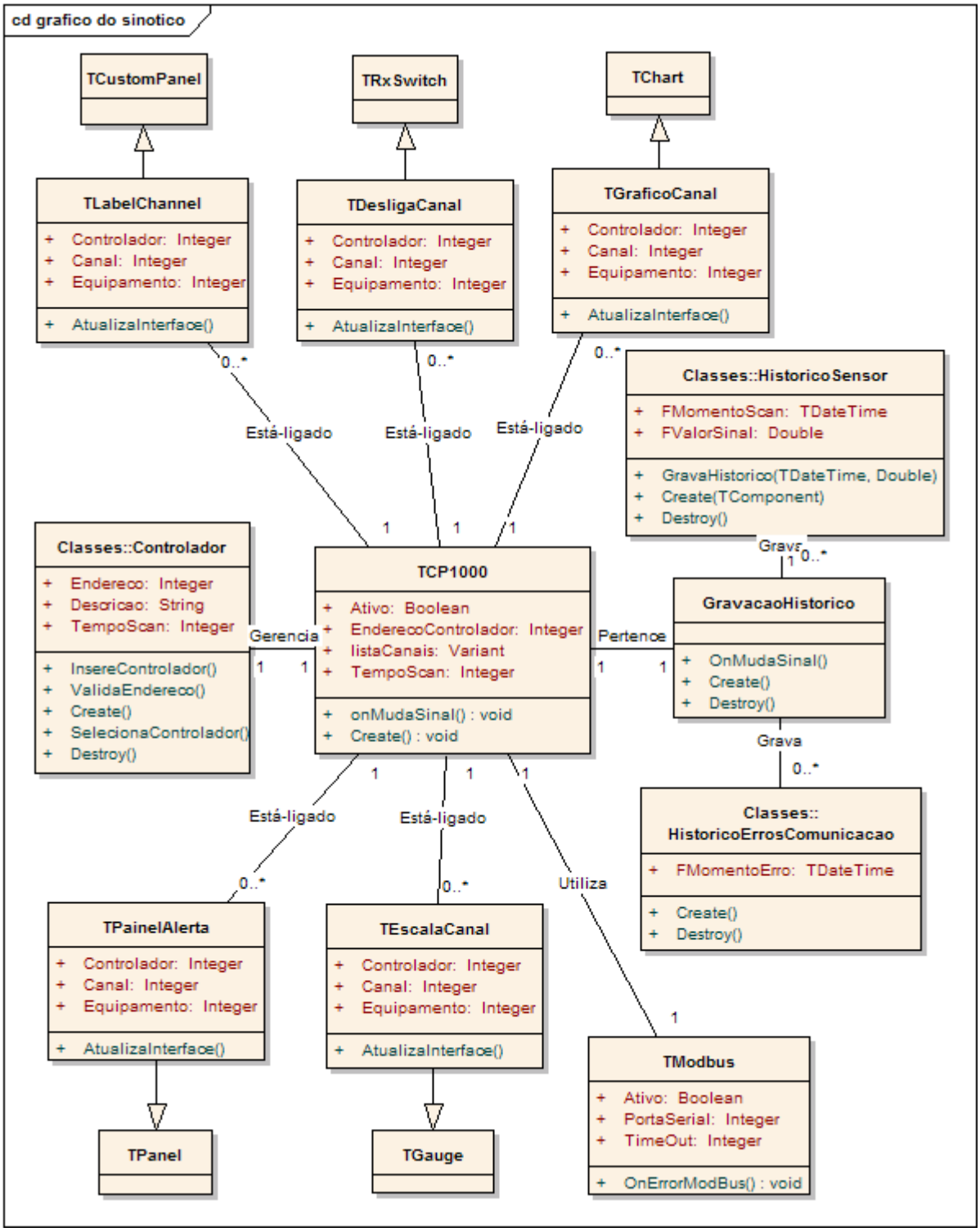



Figura 45 – Diagrama de classes do gráfico do sinótico.

### 3.2.3 Modelo de dados

Com o modelo de classes solidificado, é o momento de definir o modelo entidade/relacionamento, objetivando a criação do banco de dados relacional, que irá armazenar os objetos persistentes. Para cada classe persistente foi criada uma entidade correspondente, e definidos quais atributos das classes correspondem aos atributos da entidade. O diagrama entidade/relacionamento do projeto é mostrado na figura 46.

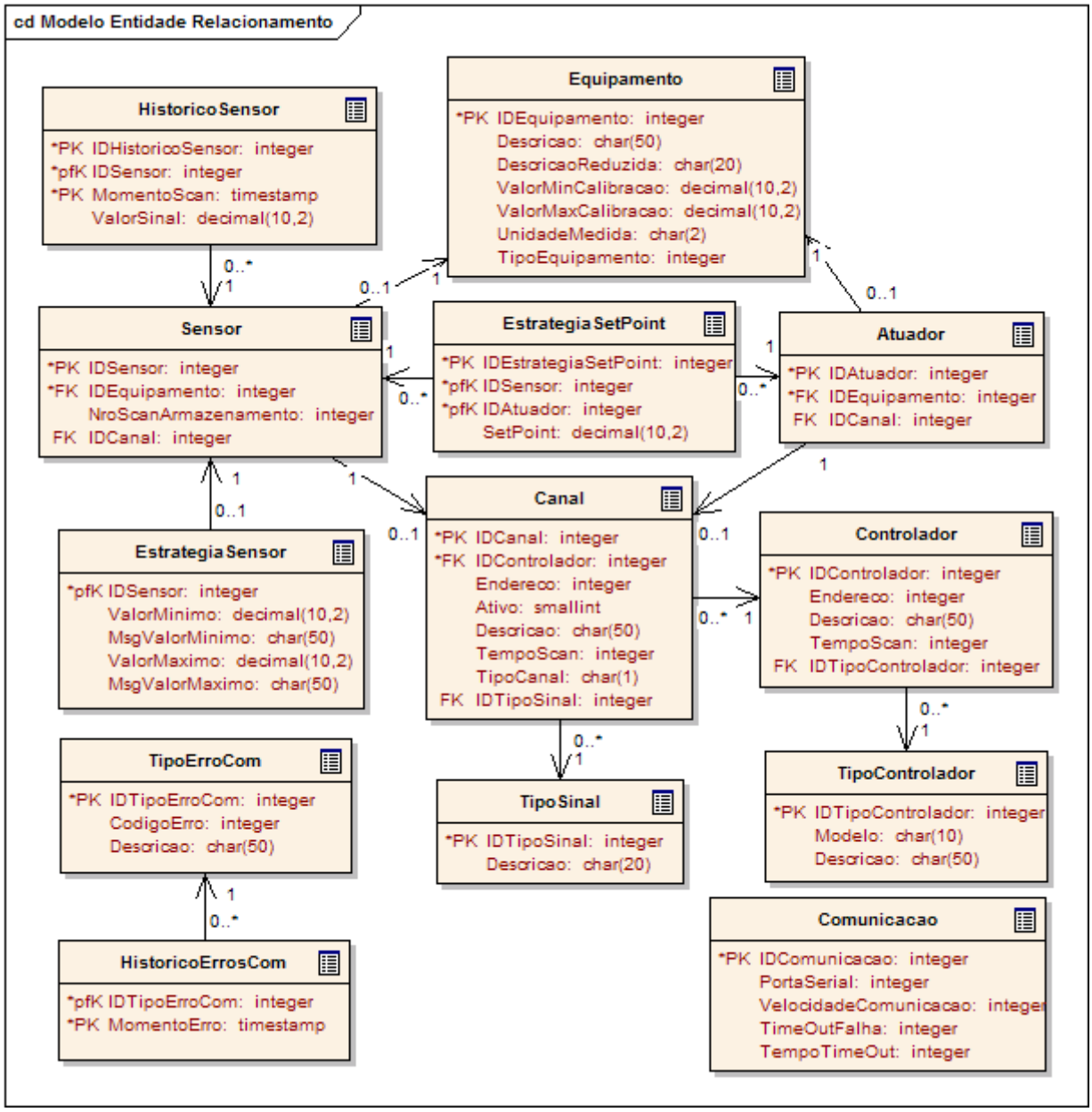



Figura 46 – Diagrama Entidade/Relacionamento.

### 3.2.4 Técnicas e ferramentas utilizadas

Foi utilizado o *Delphi* 7.0 para a implementação do código das classes e interfaces. As classes foram inicialmente geradas pelo *Enterprise Architect* e refinadas diretamente com o *Delphi*.

Para a construção da interface de configuração do sinótico foi utilizado um componente pronto para *Delphi* 7.0 chamando *Form Designer*. O *Form Designer* permite a construção de uma interface onde se pode adicionar componentes visuais a um formulário, alterar suas propriedades e salvar o formulário com extensão DFM, que é interpretado pelo *Delphi*.

O trabalho foi desenvolvido em três camadas para aumentar a coesão e reduzir o acoplamento:

a) camada de aplicação: correspondente à interface do sistema com o usuário;
b) camada de negócios: onde estão as classes que fazem parte do domínio do problema e suas operações;
c) camada de persistência: responsável pela materialização dos dados nas classes que fazem parte da camada de negócios.

A divisão em três camadas propicia que as operações essenciais do sistema não dependam da interface, nem do banco de dados escolhido para serem executadas. Isso aumenta o potencial de reutilização e a clareza do código. Mais detalhes da implementação em três camadas são mostrados no tópico 3.2.5.

O banco de dados escolhido foi o *Interbase*, versão 6.0. Um banco de dados de código aberto, fornecido pela *Inprise Corporation*.

### 3.2.5 Persistência dos dados

Um fator importante é a persistência dos dados armazenados no banco de dados, ou seja, a materialização das entidades em classes do sistema. Para isso foram codificadas duas classes básicas: *TPerBase*, responsável pela materialização de cada classe persistente, e *TPerLista*, que materializa uma coleção de uma determinada classe através de uma lista de objetos.

Todas as classes persistentes se tornaram generalizações de *TPerBase*, inclusive a classe *TPerLista*, herdando os métodos responsáveis pela materialização dos dados armazenados.

Através de engenharia reversa do *Enterprise Architect*, foi criado um diagrama de classes da persistência do sistema, com as classe *TPerBase* e suas generalizações. A figura 46 mostra um exemplo de persistência para as classes *TCanal* e *TControlador*, inclusive com as listas de coleção de objetos para as mesmas.

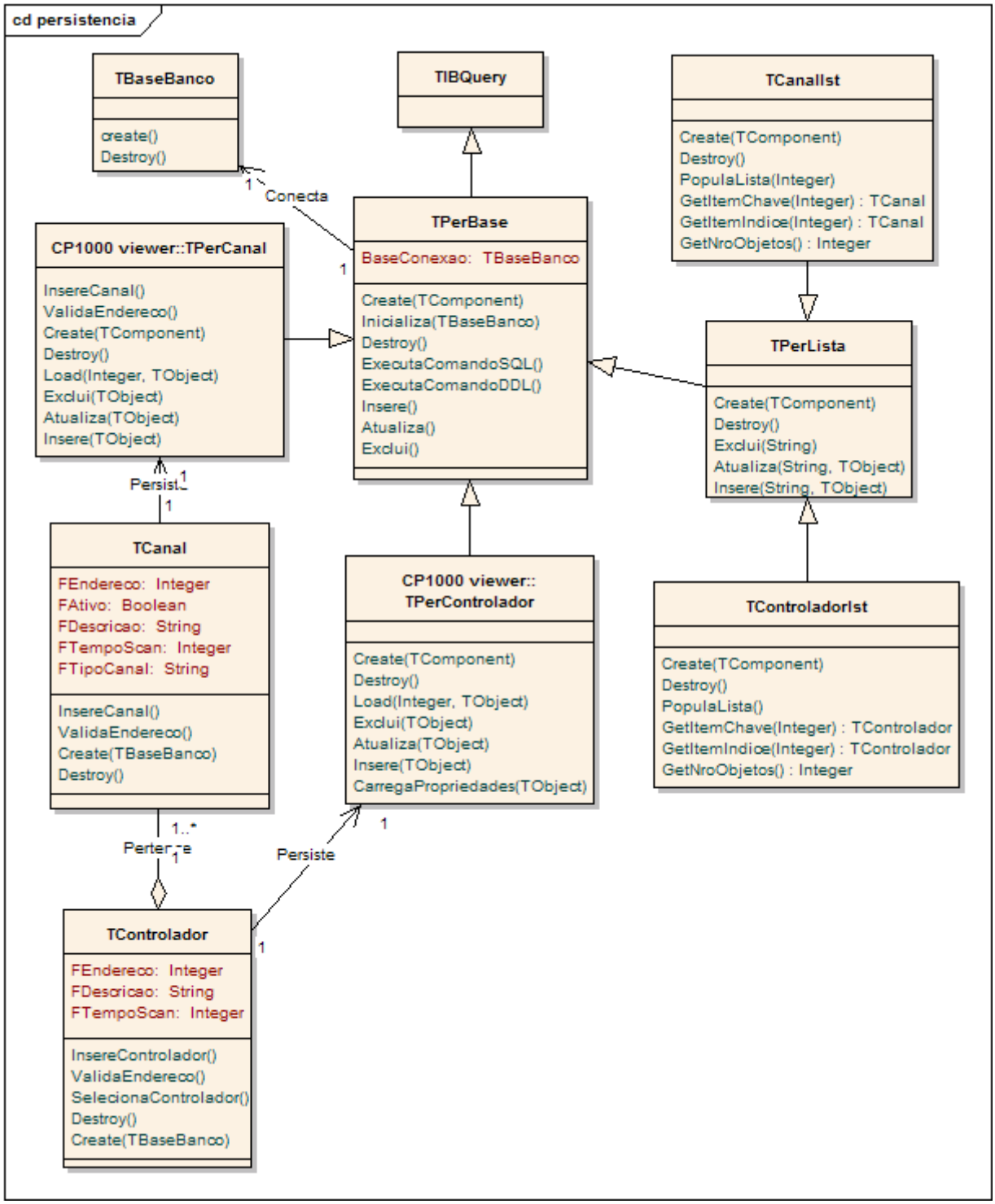



Figura 47 – Exemplo de persistência para as classes *TCanal* e *TControlador*.

O código fonte das classes de persistência é mostrado nas figuras 48 e 49. Nas figuras 50 e 51, são mostrados trechos do código das classes *TPerControlador* e *TControladorlst*, fornecendo uma visão do funcionamento do software em três camadas.

```
Procedure TPerBase.Inicializa(pBaseConexao : TBaseBanco);
begin
  BaseConexao              := pBaseConexao;
  Self.Database            := BaseConexao.IBDataBase;
  Self.Transaction         := BaseConexao.IBTransaction;
  Self.Transaction.Active := true;
end;

constructor TPerBase.Create(AOwner: TComponent);
begin
  inherited;
  FComandoSQL := TStringList.Create;
  FComandoDDL := TStringList.Create;
end;

destructor TPerBase.Destroy;
begin
  FComandoSQL.Free;
  FComandoDDL.Free;
  inherited;
end;

procedure TPerBase.ExecutaComandoSQL;
begin
  Self.SQL := FComandoSQL;
  Self.Open;
  Self.First;
end;

procedure TPerBase.ExecutaComandoDDL;
begin
  BaseConexao.IBSQL.SQL := FComandoDDL;
  BaseConexao.IBSQL.ExecQuery;
end;

procedure TPerBase.Insere;
begin
  ExecutaComandoDDL;
  BaseConexao.IBTransaction.CommitRetaining;
end;

procedure TPerBase.Atualiza;
begin
  ExecutaComandoDDL;
  BaseConexao.IBTransaction.CommitRetaining;
end;

procedure TPerBase.Exclui;
begin
  ExecutaComandoDDL;
  BaseConexao.IBTransaction.CommitRetaining;
end;
```

Figura 48 – Código fonte da classe *TPerBase*.

```
procedure TPerLista.Insere(Chave : String;lObjeto : TObject);
begin
  FListadeObjetos.AddObject(Chave, lObjeto);
end;

Function TPerLista.GetItemListaChave(Chave : String) : TObject;
var
  lIndex : Integer;
begin
  lIndex := FListadeObjetos.IndexOf(Chave);
  if  lIndex <> -1 then
      Result := FListadeObjetos.Objects[lIndex]
  Else
      Result := nil;
end;

Function TPerLista.GetItemListaIndice(Index : Integer) : TObject;
begin
  Result := FListadeObjetos.Objects[Index];
end;

Function TPerLista.GetCount : Integer;
begin
  Result := FListadeObjetos.Count;
end;

procedure TPerLista.Exclui(Chave : String);
var
  lIndex : Integer;
begin
  lIndex := FListadeObjetos.IndexOf(Chave);
  if  lIndex <> -1 then
      FListadeObjetos.Delete(lIndex);
end;

procedure TPerLista.Atualiza(Chave : String;pObj : TObject);
begin
  FListadeObjetos.AddObject(Chave, pObj);
end;
```

Figura 49 – Código fonte da classe *TPerLista*

```
procedure TPerControlador.Load(IDValue:Integer;objControlador:TObject)
begin
  ComandoSQL.Clear;
  ComandoSQL.Add('SELECT *');
  ComandoSQL.Add('FROM Controlador WHERE IDControlador=' +
  IntToStr(IDValue));
  ExecutaComandoSQL;
  CarregaPropriedades(objControlador);
end;

procedure TPerControlador.Exclui(objControlador : TObject);
begin
  ComandoDDL.Clear;
  ComandoDDL.Add('DELETE FROM Controlador WHERE IDControlador=' +
            IntToStr(TControlador(objControlador).IDControlador));
  inherited Exclui;
end;

procedure TPerControlador.Atualiza(objControlador : TObject);
begin
  ComandoDDL.Clear;
  ComandoDDL.Add('UPDATE Controlador SET Endereco = ' +
   inttostr(TControlador(objControlador).Endereco) +
   ', Descricao = ''' +
   TControlador(objControlador).Descricao + '''' +
   ', TempoScan = ' +
   inttostr(TControlador(objControlador).TempoScan) +
   ', IDTipoControlador = ' +
   TControlador(objControlador).m_TipoControlador.IDTipoControlador));
  ComandoDDL.Add('WHERE IDControlador=' +
          IntToStr(TControlador(objControlador).IDControlador));
  inherited Atualiza;
end;

procedure TPerControlador.Insere(objControlador : TObject);
begin
  ComandoDDL.Clear;
  ComandoDDL.Add('INSERT INTO Controlador ');
  ComandoDDL.Add('(IDControlador, Endereco, Descricao, ');
  ComandoDDL.Add(' TempoScan, IDTipoControlador)');
  ComandoDDL.Add(' VALUES');
  ComandoDDL.Add('(' +
    inttostr(TControlador(objControlador).IDControlador) + ','
    + inttostr(TControlador(objControlador).Endereco) +
    ',''' + TControlador(objControlador).Descricao + ''','
    + inttostr(TControlador(objControlador).TempoScan) +
    ',' +
    TControlador(objControlador).m_TipoControlador.IDTipoControlador)
    + ')');
  inherited Insere;
end;
```

Figura 50 – Código fonte da classe *TPerControlador*

```
procedure TControladorlst.CarregaObjeto;
begin
  FControlador := TControlador.Create(BaseConexao);
  FControlador.IDControlador:=Fields.FieldByName('IDControlador').AsIntege
  FControlador.Endereco  := Fields.FieldByName('Endereco').AsInteger;
  FControlador.Descricao := Fields.FieldByName('Descricao').AsString;
  FControlador.TempoScan:= Fields.FieldByName('TempoScan').AsInteger;
  FControlador.lstCanal.PopulaLista(Fields.FieldByName('IDControlador').As
  FControlador.m_TipoControlador.Load(Fields.FieldByName('IDTipoControlado
  Insere(Fields.FieldByName('IDControlador').AsString, FControlador);
end;

procedure TControladorlst.PopulaLista;
begin
  ComandoSQL.Clear;
  ComandoSQL.Add('SELECT *');
  ComandoSQL.Add('FROM Controlador ');
  ExecutaComandoSQL;
  LimpaLista;
  While  TemRegistros do
      begin
        CarregaObjeto;
        ProximoRegistro;
      end;
end;

Function TControladorlst.GetItemChave(IDChave:Integer):TControlador;
begin
  Result := GetItemListaChave(inttostr(IDChave)) as TControlador;
end;

Function TControladorlst.GetItemIndice(pIndice:Integer):TControlador;
begin
  Result := GetItemListaIndice(pIndice) as TControlador;
end;

Function TControladorlst.GetNroObjetos : Integer;
begin
  Result := GetCount;
end;
```

Figura 51 – Código fonte da classe *TControladorlst*

### 3.2.6 A organização do *Enterprise Architect*

A ferramenta *Enterprise Architect* disponibiliza a organização de suas pastas em uma árvore de visualização. As pastas foram organizadas de acordo com a tabela de amostra de artefatos do RUP, apresentado por LARMAN (2004, p. 98). A árvore é mostrada na figura 52.

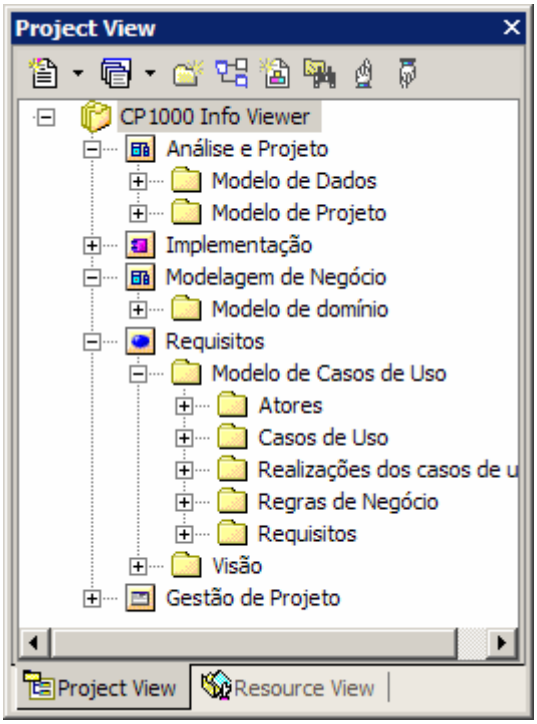



Figura 52 – Organização do *Enterprise Architect*

### 3.2.7 Operacionalidade da implementação

Para demonstrar a operacionalidade da implementação, serão apresentados os passos básicos para a configuração e uso do sistema, mostrando as principais telas envolvidas em cada passo.

O primeiro passo na execução do sistema é a configuração dos controladores e canais, que espelharão o *hardware* do *CP1000*.

A tela de configuração geral dos controladores traz, inicialmente, a lista dos controladores cadastrados. Podem ser incluídos novos controladores, alterados ou excluídos os existentes. Ao selecionar um controlador, ou incluir um novo, tem-se a tela de configurações do controlador e seus canais.

Na figura 53, é mostrada a lista de controladores já cadastrados. A figura 54 mostra as configurações de canais para o controlador 1, onde se pode escolher o canal a ser configurado.

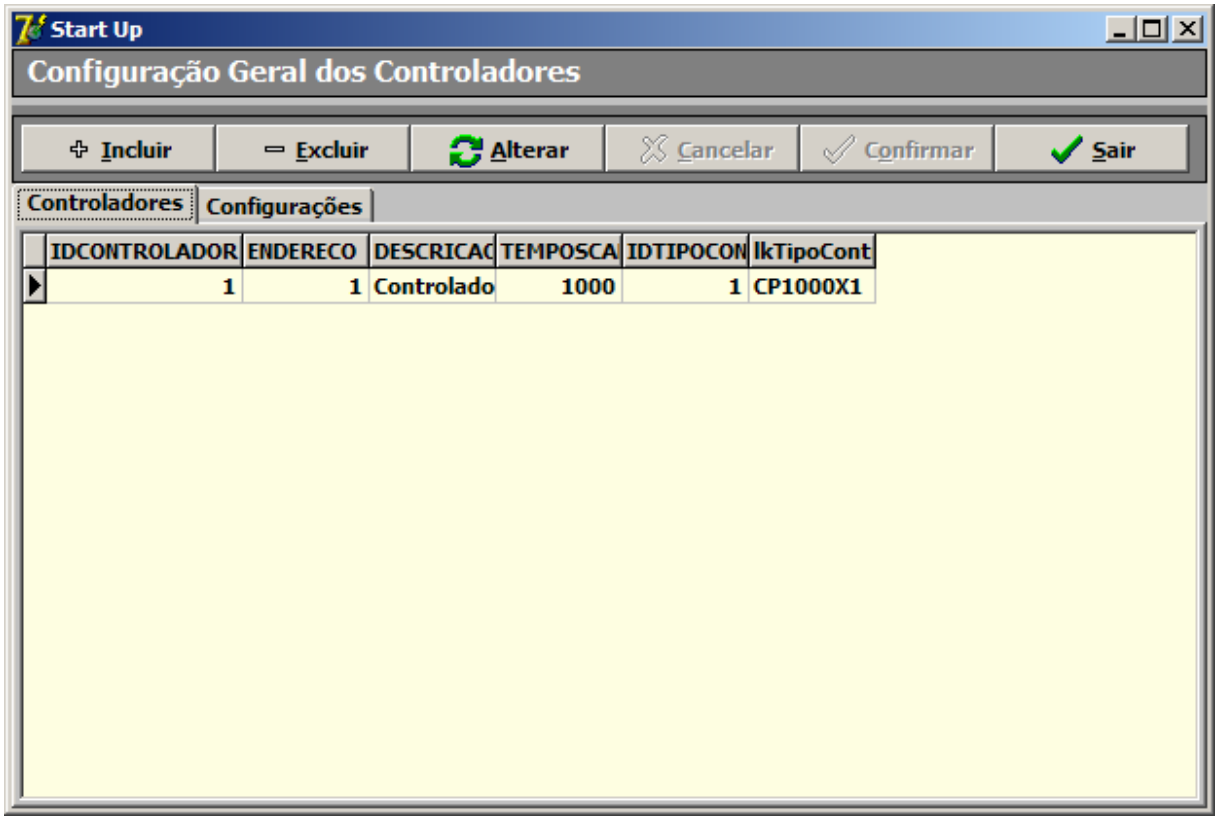



Figura 53 – Interface de configuração dos controladores(lista de controladores)


Figura 54 – Interface de configuração dos controladores(configuração dos canais)

A próxima etapa é o cadastro dos equipamentos de controle disponíveis na fábrica. Ao entrar na tela de cadastro de equipamentos, é mostrada uma listagem de todos os equipamentos já cadastrados, podendo alterá-los ou excluí-los. Escolhendo uma das duas opções, ou incluindo um novo componente, é apresentada a tela de configurações dos equipamentos, onde pode ser escolhido se o equipamento é um sensor ou um atuador, e preencher as demais características.

A figura 55 mostra a listagem de equipamentos disponíveis e a figura 56, as configurações disponíveis para os mesmos.

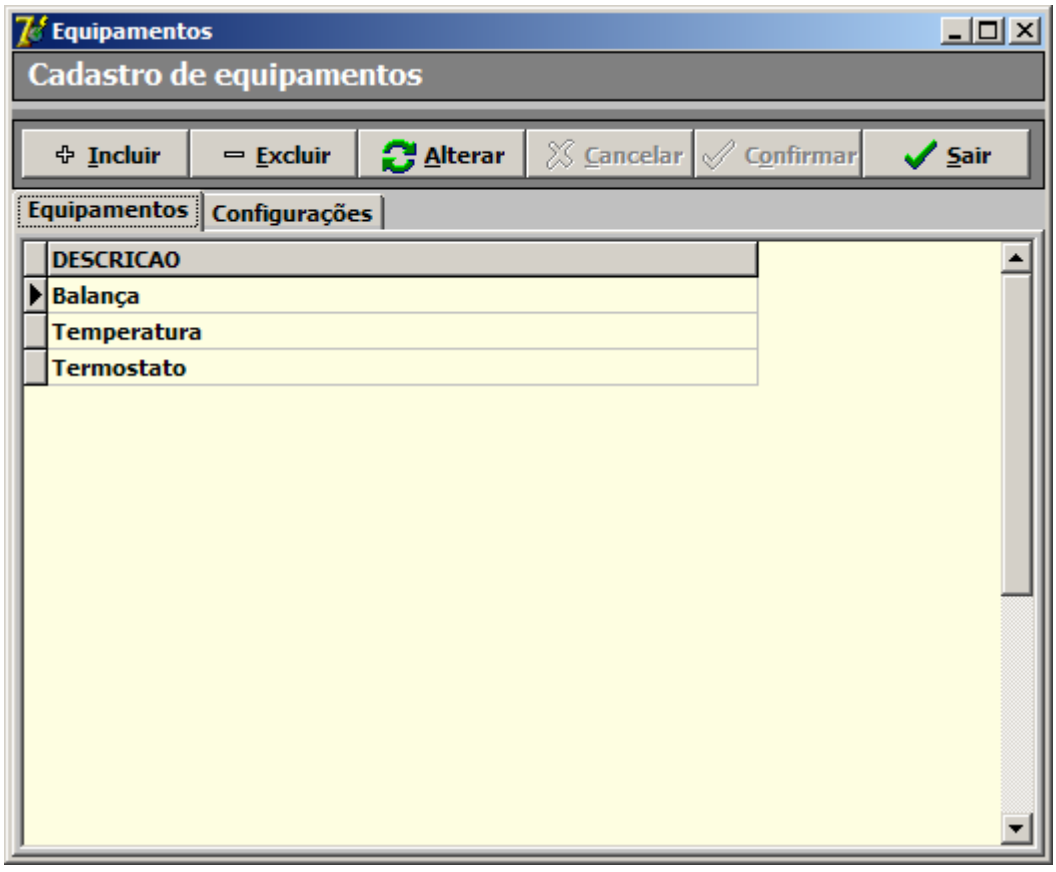



Figura 55 – Interface de cadastro de equipamentos(lista de equipamentos)

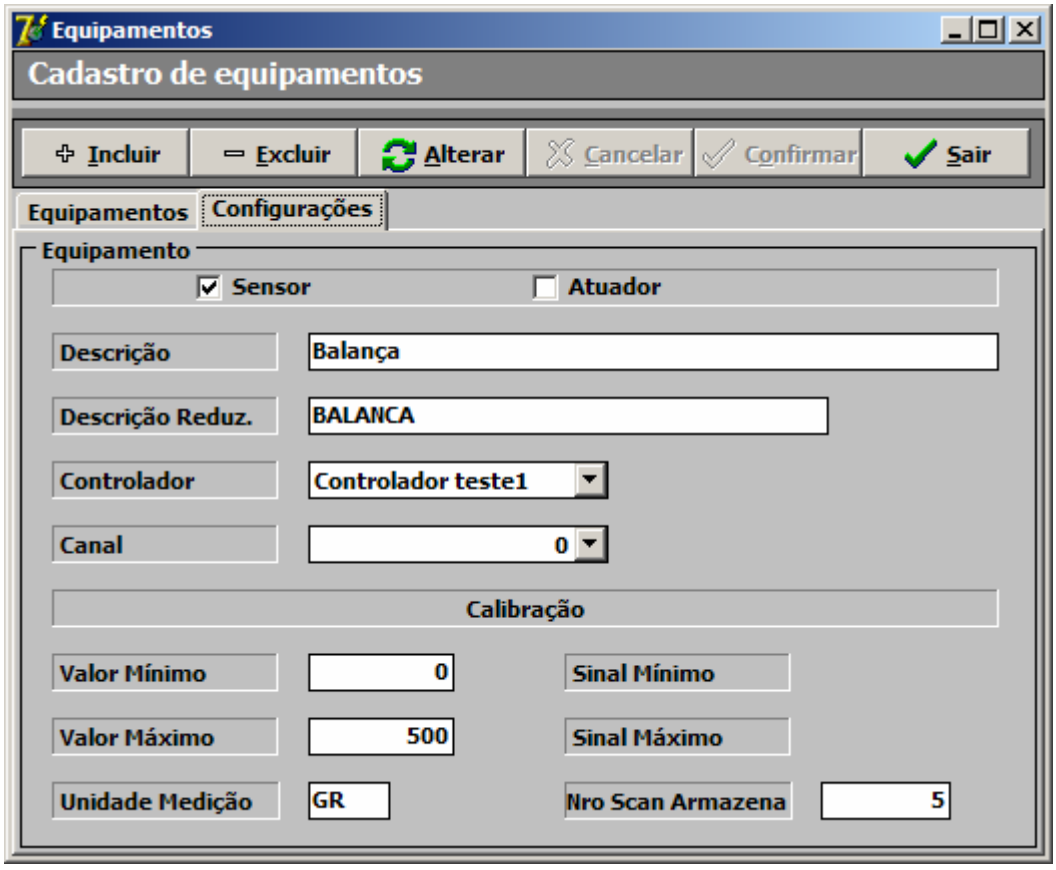



Figura 56 – Interface de cadastro de equipamentos(configurações)

Para configurar as estratégias de *Set Point* e sensores, é disponibilizada uma tela com duas divisões. Na primeira divisão é possível escolher um atuador e um sensor que funcionaram juntos para manter um valor de *Set Point* informado, como mostrado na figura 57. Na segunda divisão são configuradas as mensagens de alerta para os sensores quando atingem valores mínimos e máximos, mostrada na figura 58.

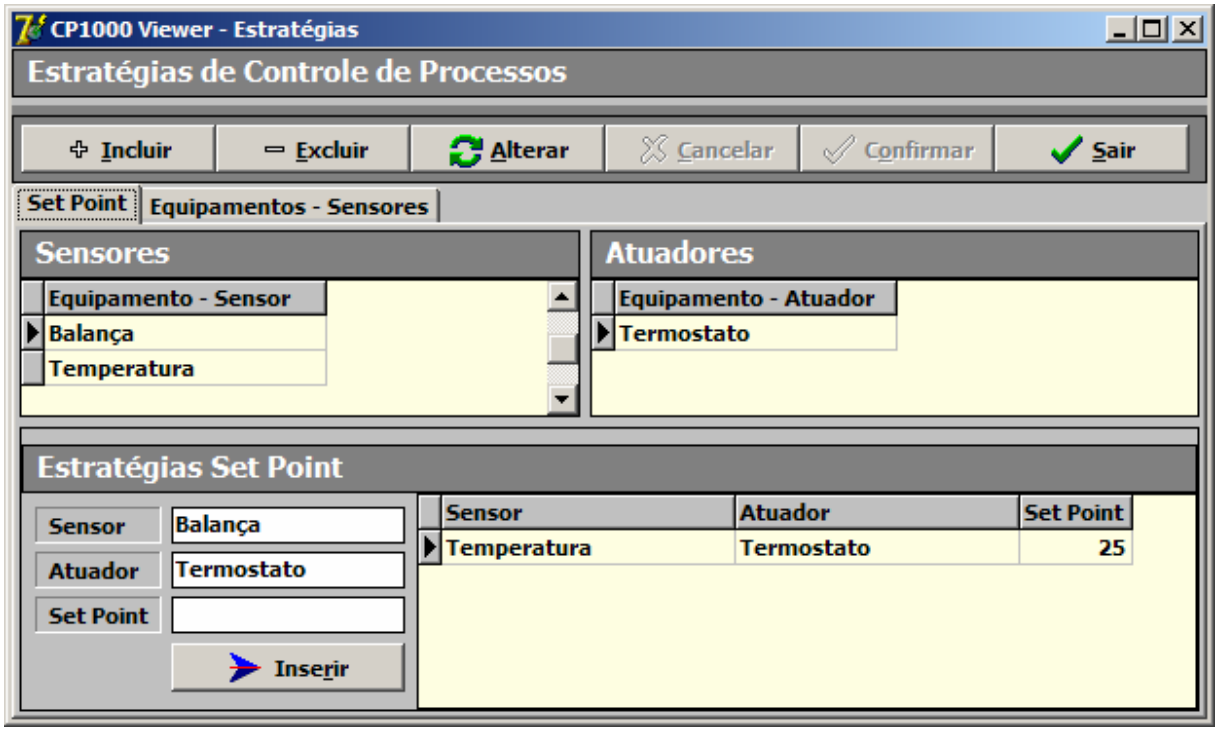



Figura 57 – Interface de estratégias de *Set Point*

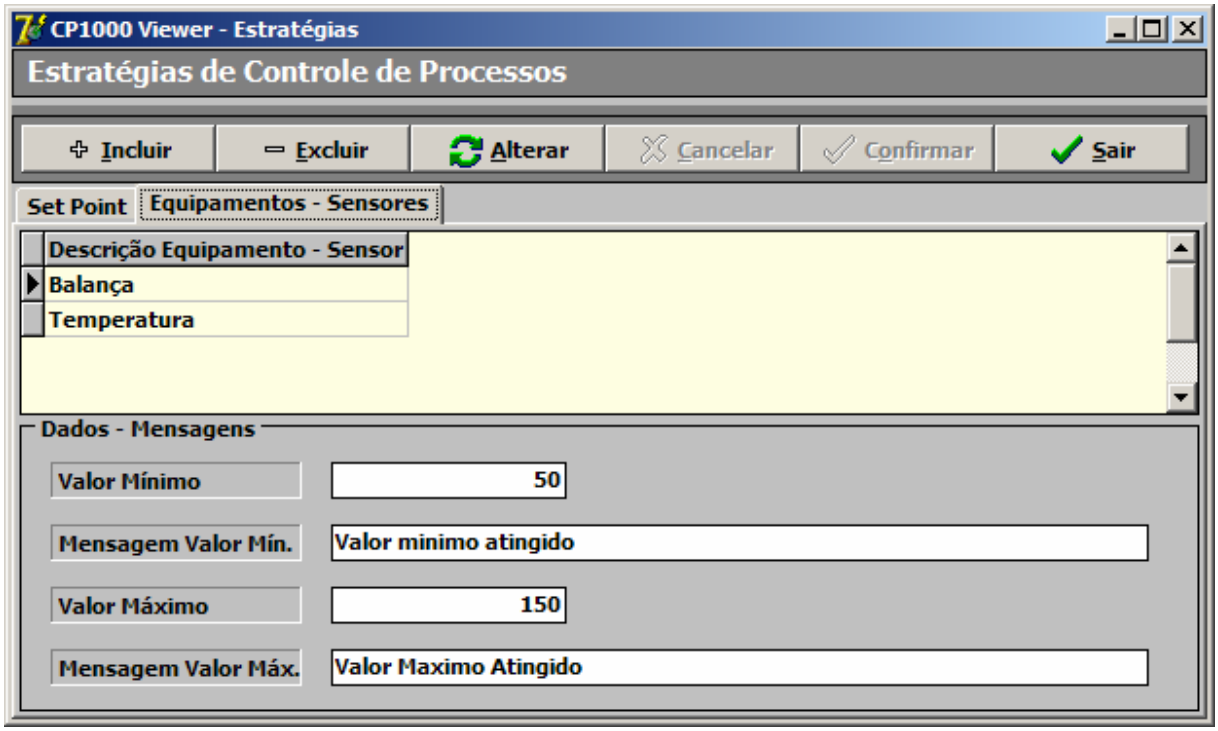



Figura 58 – Interface de estratégias de sensores

A tela com maior complexidade é a responsável pela construção do sinótico. Nesta tela, o usuário tem acesso a uma ferramenta que disponibiliza vários componentes visuais que

podem ser adicionados a um formulário, de acordo com o sinótico desejado. Este formulário pode ser salvo e alterado futuramente.

Para cada componente adicionado ao formulário, o usuário escolhe o canal ligado ao componente, configura as cores e tamanhos. Existem, nesta tela, funções auxiliares para a configuração dos componentes, como alinhamento, copiar e colar, entre outras. A construção de um sinótico para balança digital é mostrada na figura 59.

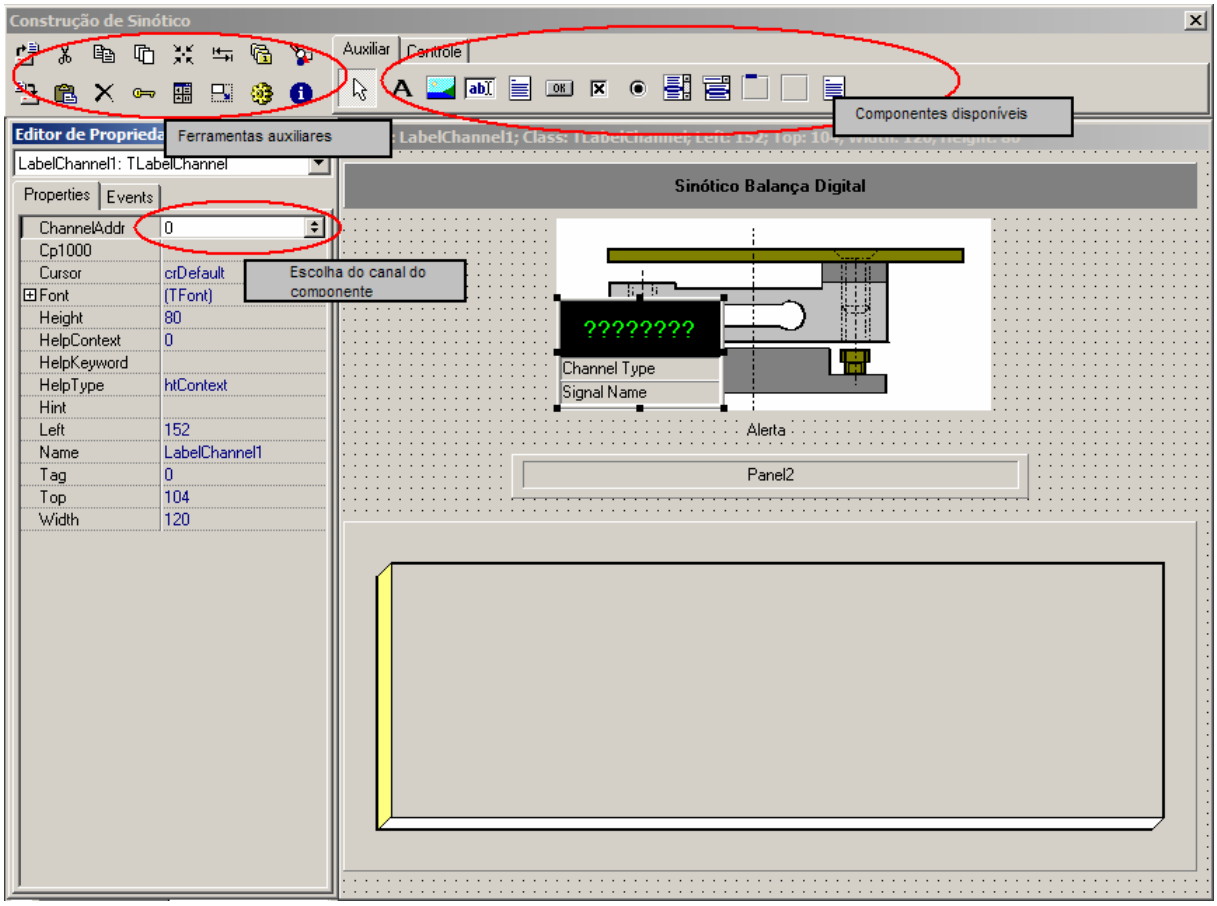



Figura 59 – Interface de construção do sinótico

Após a construção do sinótico, o usuário, quando necessário, ativa um determinado sinótico, que passa a ter comunicação *on line* com o *CP1000* e fornecer informações de leitura de cada equipamento presente no sinótico. A figura 60 mostra o sinótico da balança digital em funcionamento.

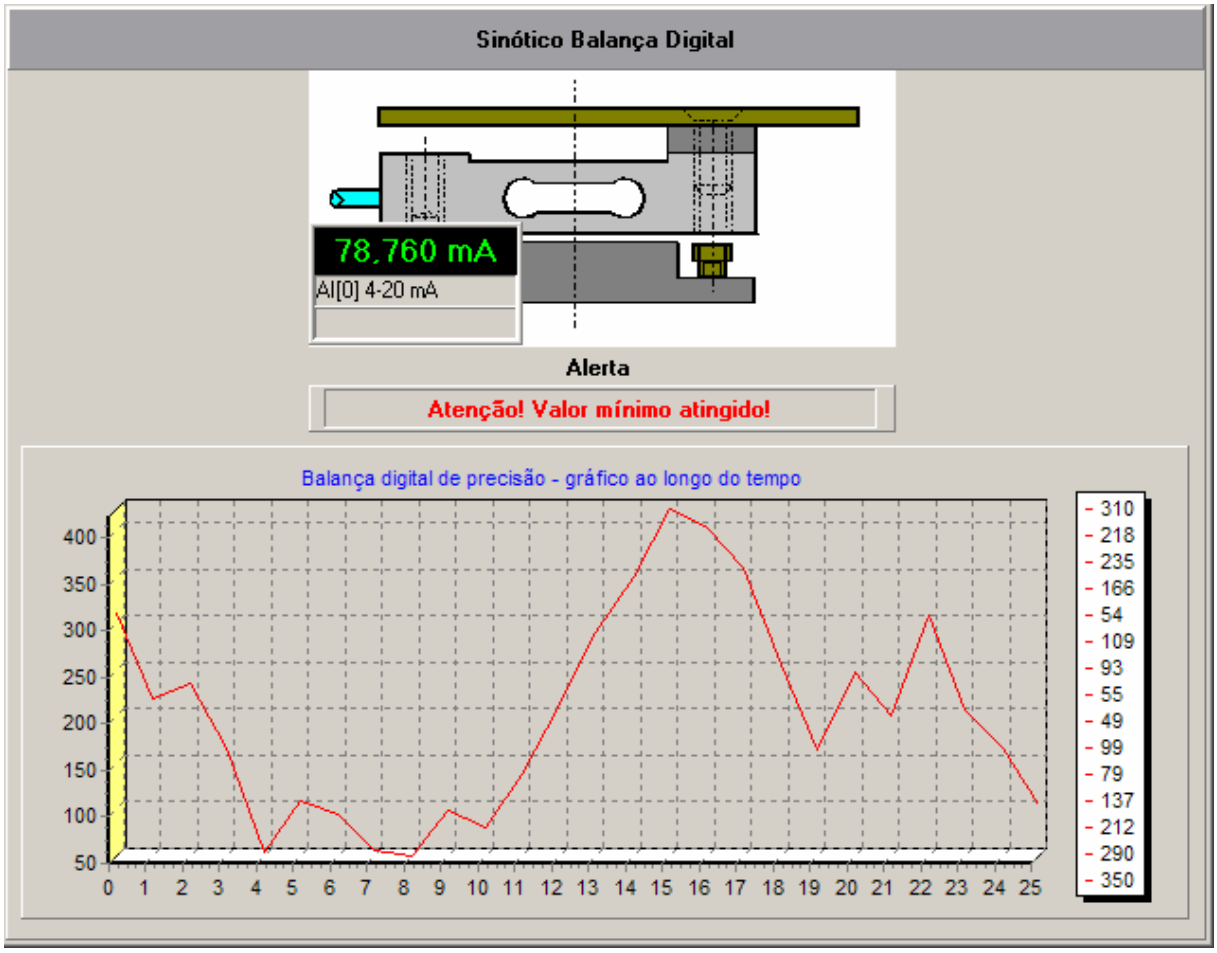



Figura 60 – Sinótico de uma balança digital ativo

## 3.3 TERCEIRA ITERAÇÃO

A terceira iteração, definida para a fase de transição do RUP, seguiu o fluxo de trabalho apresentado na figura 61. No fluxo observa-se os detalhamentos *Testar o Produto*, onde é feita a validação do sistema através de testes empíricos, *Preparar implantação*, onde é gerado o modelo de implantação, *Implantar o Sistema*, que é a disponibilização do sistema para o usuário final, e *Realizar Ajustes*, que trata das falhas e pequenas modificações solicitados pelo usuário final.

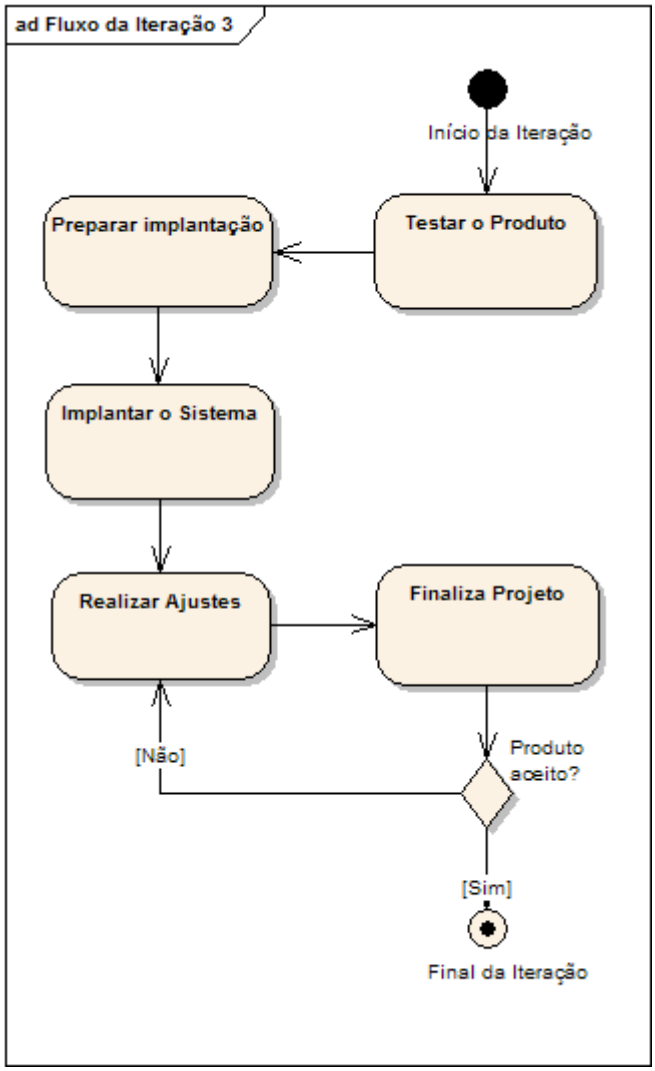



Figura 61 – Fluxo de trabalho para a terceira iteração

O fluxo de trabalho apresentado serviu apenas como referência para uma futura implantação do *software*, já que a implantação não chegou a ser feita efetivamente no escopo deste trabalho.

## 3.4 RESULTADOS E DISCUSSÃO

O *software* desenvolvido neste trabalho atendeu as expectativas da *Softlution*, empresa fabricante do *CP1000*. É possível, através do *software*, ter um controle visual dos processos, apresentando ao usuário informações estratégicas em tempo real e armazenando todas estas informações para consultas futuras.

Os artefatos apresentados no trabalho dizem respeito ao resultado final encontrado em cada iteração, foram abstraídos alguns refinamentos feitos nos artefatos no decorrer de cada iteração. Um exemplo desse refinamento é que no primeiro modelo de casos de uso encontrado, não seriam cadastrados sensores e atuadores. O controle seria feito diretamente no canal de entrada ou saída, que teriam valores para calibração, estratégias e histórico. Os cenários, digramas de atividades e modelo conceitual estavam todos voltados para este controle. Na figura 62 é mostrado o modelo de domínio conceitual com o controle por canal.

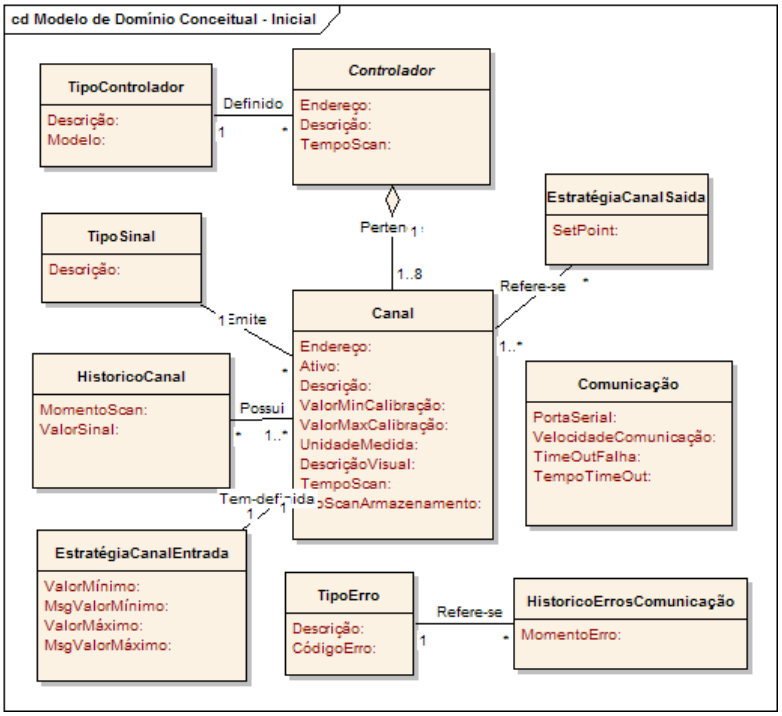



Figura 62 – Modelo de domínio conceitual inicial

Na segunda iteração foi observado, após alguma codificação, que seria mais aplicável se o controle fosse feito através de equipamentos, divididos em sensores e atuadores, possibilitando a troca dos canais configurados para cada equipamento, sem afetar históricos e informações dos mesmos.

Com a utilização do RUP, o processo deve ser feito de forma incremental, possibilitando atacar casos de uso complexos ou conjuntos de casos de uso, distribuindo o desenvolvimento para a equipe, e percorrendo todas as disciplinas do RUP, até encerrar o

caso de uso escolhido.

No desenvolvimento deste trabalho, os casos de uso não eram complexos e não havia uma equipe disponível para a execução dos passos de maneira distribuída. Por isso, na apresentação do desenvolvimento do trabalho pode-se ter a impressão da utilização de um método de desenvolvimento em cascata, apesar de terem sido feitos alguns refinamentos, já que todos os casos de uso foram atacados juntos.

### 3.4.1 Problemas encontrados durante o desenvolvimento

O principal problema encontrado foi referente à persistência dos dados. Isso fez com que a primeira iteração tivesse uma longa duração, até que fosse definida a tecnologia a ser utilizada no problema. Após essa definição, já discutida anteriormente, foi possível construir facilmente as classes de persistência na segunda iteração.

Também foram encontradas dificuldades relativas à utilização do RUP. O paradigma de desenvolvimento em cascata muitas vezes afetou um bom entendimento do RUP, que utiliza iterações e incrementos para o processo de desenvolvimento.

## 4 CONCLUSÕES

O principal objetivo do trabalho foi atingido. O protótipo desenvolvido atendeu os requisitos levantados e as expectativas do fabricante do CP1000. O controle de processos industriais pode ser feito de forma visual através da ativação de um sinótico montado pelo usuário. Todo o processo de configuração de equipamentos, sinóticos e estratégias é de fácil manutenção, o que agiliza as alterações por parte do usuário.

A *Softlution*, empresa fabricante do CP1000, demonstrou grande interesse na comercialização do *software* juntamente com a CPU, tornando-o o aplicativo padrão para gerenciamento da mesma.

O *Enterprise Architect* se mostrou uma ótima ferramenta no controle e confecção dos artefatos, além de proporcionar grande flexibilidade para gerar relatórios da documentação.

A UML teve grande importância no entendimento do problema e na construção de referências entre as fases, proporcionando uma documentação clara de todo o desenvolvimento do trabalho.

Por se tratar o RUP de um processo de desenvolvimento configurável, segui-se a metodologia utilizada por Larman (2004) para o desenvolvimento do trabalho, que enfatiza as disciplinas de Requisitos, Análise e Projeto e Implementação. Também foram gerados os artefatos: modelo de domínio e regras de negócio, da disciplina de Modelagem de Negócio. Os requisitos levantados inicialmente foram refinados a cada iteração, chegando a um produto final de qualidade e com uma documentação clara de todas as etapas do desenvolvimento.

Um ponto importante a se destacar é o conhecimento adquirido, tanto quanto à UML, como ao RUP, durante o desenvolvimento do trabalho. Foi possível aprender a desenhar alguns diagramas da UML, como o diagrama de casos de uso, o diagrama de atividades, o diagrama de seqüência, entre outros.

Uma limitação do protótipo é ter uma quantidade limitada de componentes visuais, o que pode dificultar a representação visual de determinados equipamentos, apesar de representar os principais processos propostos.

## 4.1 EXTENSÕES

Três extensões, que não faziam parte dos objetivos iniciais deste trabalho, podem ser sugeridas para trabalhos futuros:

a) a elaboração de uma biblioteca de componentes visuais feita especialmente para a o protótipo, contemplando a representação de um maior número de equipamentos;
b) o desenvolvimento das interfaces voltadas para *Web*, o que tornaria possível o gerenciamento dos processos industrias a distância em tempo real;
c) a confecção de um maior número de artefatos, contemplando disciplinas do RUP não exploradas neste desenvolvimento.

# REFERÊNCIAS BIBLIOGRÁFICAS


BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** guia do usuário. Rio de Janeiro: Campus, 2000.

BORK, Cláudio Juliano. **Customização e implantação de um processo de desenvolvimento de *software* baseado no Rational Unified Process (RUP)**. 2003. 90 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DEBONI, José Eduardo Zindel. **Modelagem orientada a objetos com a UML**. São Paulo: Futura, 2003.

DELMAR Ltda. **Modern control technology:** components and systems. Kilian, 2004.

LACERDA, Antônio Corrêa de et al. **Tecnologia:** estratégia para a competitividade. São Paulo: Nobel, 2001.

LARMAN, Craig. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos e ao processo unificado. 2. ed. São Paulo: Bookman, 2004.

LENZI, Gabriel. **Desenvolvimento de um protótipo de controle alimentar suíno, utilizando aquisição de dados e monitoramento de processo**. 1998. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LOPEZ, Rui. **Sistemas de redes para controle e automação**. São Paulo: Book Express, 2002.

MARTIN, Fowler; SCOTT, Kendall. **UML essencial**: um breve guia para a linguagem padrão de modelagem de objetos. Tradução Vera Pezerico e Christian Thomas Price. 2. ed. Porto Alegre: Bookman, 2000.

RATIONAL, Software Corporation. **Rational Unified Process**. Versão 2002.05.00. 2002. Disponível em: http://www.wthreex.com/rup. Acesso em 11 jul. 2005.

SCOTT, Kendall. **O processo unificado explicado**. Tradução Ana M. de Alencar Price. Porto Alegre: Bookman, 2003.

REIS, Jocelene Oliveira; BELCHIOR, Arnaldo Dias. MyRup: uma adaptação do RUP para projetos de pequeno e médio porte. In: XXX CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA, Arequipa, 2004. p. 597-608.

# APÊNDICE A – Glossário do projeto

| Item | Tipo | Significado |
|---|---|---|
| Atuador | Technical | Controle de processo industrial que recebe sinal do canal do controlador. Exemplo: aquecedor. |
| Calibração | Technical | Faixa de valores que define a apresentação do sinal adquirido do canal em volts para a unidade de medição escolhida (MT, KG, CM, etc.). |
| Canal | Technical | Entradas disponíveis no controlador, onde podem ser conectados sensores ou atuadores. Um canal pode ser de entrada ou de saída, analógico ou digital. |
| Controlador | Technical | Placa controladora contendo canais de entrada e saída. |
| CP1000 | Technical | Unidade central de processamento (CPU) micro-processada que pode ser modularizada, com canais de entradas e saídas digitais, analógicas, saídas de potência, comunicação serial, que controla não só sensores como atuadores, mandando comandos para os processos, como por exemplo, aumentar a temperatura. Esta CPU gerencia uma rede de equipamentos micro-processados que poderão estar ligados a qualquer processo industrial, tais como, controle e aquisição de dados de temperatura, pH, vazão, nível, umidade relativa, peso, entre outros. |
| Sensor | Technical | Controle de processo industrial que envia sinal para o canal do controlador. Exemplo: balança, termostato. |
| Sinótico | Technical | Interface visual interativa que retrata os processos industriais configurados em uma fábrica, com o mesmo formato da planta da fábrica. |
| Start Up | Technical | Configuração inicial de controladores e canais |
| Tempo de Scan | Technical | Intervalo de tempo em que a CPU faz a leitura dos canais configurados. |
| Time Out | Technical | Tempo máximo para resposta de um determinado sinal. |

**APÊNDICE B – Documento de Visão**

# 1 INTRODUÇÃO

A finalidade deste documento é coletar, analisar e definir necessidades e recursos de nível superior do *CP1000 Data Viewer*. Ele se concentra nos recursos necessários aos envolvidos e aos usuários-alvo e nas razões que levam a essas necessidades. Os detalhes de como o *CP1000 Data Viewer* satisfaz essas necessidades são descritos nos caso de uso, seus cenários e diagramas de atividades.

# 2 POSICIONAMENTO

## 2.1 DESCRIÇÃO DO PROBLEMA

| | |
|---|---|
| O problema de | Gerenciar uma CPU micro-processada desenvolvida para adquirir dados de processos industriais e armazenar em memória para consultas futuras. |
| Afeta | A empresa *Softlution,* fabricante da CPU denominada *CP1000.* |
| Cujo impacto é | Dificuldade de visualização dos dados na própria CPU. |
| Uma boa solução seria | Desenvolver um sistema, com interface gráfica que proporcione um gerenciamento visual dos processos industriais disponíveis e acessados pela CPU. |

## 2.2 SENTENÇA DE POSIÇÃO DO PRODUTO

| | |
|---|---|
| Para | Fábricas em geral |
| Que | Necessitem de um controle rigoroso de processos industriais distrubuídos pela planta da fábrica. |
| O CP1000 Data Viewer | É um sistema visual de controle de processos |
| Que | Permite ao usuário gerenciar processos industriais de forma visual, com interfaces amigáveis e de fácil manipulação. |
| Ao contrário de | Qualquer produto existente no mercado, que é específico para um determinado processo. |
| Nosso Produto | Pode gerenciar qualquer tipo de processo industrial, pois |

| | |
|---|---|
| | a CPU adquire o sinal enviado pelos mesmos independente do processo. |

# 3 DESCRIÇÃO DOS ENVOLVIDOS E DOS USUÁRIOS

Neste capítulo são identificados os envolvidos no projeto e os usuários do sistema, fornecendo um perfil dos envolvidos e dos usuários que integram o projeto, e dos principais problemas que, de acordo com o ponto de vista deles, poderão ser abordados pela solução proposta.

## 3.1 RESUMO DOS ENVOLVIDOS

| Nome | Descrição | Responsabilidades |
|---|---|---|
| Desenvolvedor | Já que trata-se de um projeto individual, apenas o desenvolvedor cuidará das responsabilidades do sistema. | - Levantar requisitos;<br>- Definir a arquitetura do *software*;<br>- Implementar a arquitetura;<br>- Aplicar os testes;<br>- Gerenciar o projeto. |

## 3.2 RESUMO DOS USUÁRIOS

| Nome | Descrição | Responsabilidades | Envolvido |
|---|---|---|---|
| Fabricante | Empresa fabricante do CP1000. | - Acompanhar os testes de sistema;<br>- Aprovar testes de sistema. | |
| Usuário Final | Usuário da fábrica | - Aprovar o sistema visual. | Fabricante |

## 3.3 AMBIENTE DO USUÁRIO

O usuário final tem uma mesa de controle não configurável, visualizando os processos

industriais em andamento. Ou um computador com vários sistemas, uma para cada processo industrial ou grupos de processos de um determinado fornecedor.

Com o *CP100 Data Viewer*, o usuário poderá ter apenas um computador, localizado em qualquer lugar na fábrica, de onde os processos podem ser gerenciados.

## 3.4 PRINCIPAIS NECESSIDADES DOS ENVOLVIDOS E DOS USUÁRIOS

O sistema deve adquirir dados de uma unidade central de processamento (CPU) micro-processada, que conta com vários canais previamente definidos, nos quais estão ligados processos industriais. O sistema deverá, em um intervalo de tempo definido como parâmetro, capturar os dados de cada canal, através da porta serial do computador e mostrá-los na tela com uma interface gráfica amigável para que o usuário possa gerenciar os processos de forma visual. Os dados também devem ser armazenados para futuras consultas e impressão de relatórios e gráficos.

O usuário terá acesso a uma ferramenta de *design* na qual ele poderá desenhar a planta da fábrica através de um sinótico, incluindo, através de componentes visuais (rótulos, gráficos, alertas), as informações de cada canal disponível. Quando o sinótico configurado estiver ativado, as informações de cada canal, atualizadas no intervalo de tempo estabelecido nos parâmetros, serão mostradas no sinótico.

O sistema deve contar com um cadastro de parâmetros a serem configurados pelo usuário, como o endereço de cada equipamento de controle na rede, calibração de sensores e instrumentos, especificando a relação entre o sinal de entrada de um sensor com a respectiva unidade de medição, a porta serial conectada, velocidade de comunicação, o tempo para o *time out* e o número de *time outs* para gerar uma falha de comunicação.

Deverá também contemplar a configuração das estratégias de controle de processo industrial, configurando quais atuadores serão manipulados de acordo com as respectivas leituras dos sensores.

# 4 VISÃO GERAL DO PRODUTO

## 4.1 PERSPECTIVA DO PRODUTO

O *CP100 Data Viewer* é dependente da CPU micro-processada, a CP1000, fabricada pela *Softlution*, e adquire as informações necessárias ao gerenciamento dos processos da mesma, que está ligada, através de canais de entrada e saída, aos equipamentos de processos a serem gerenciados.

# 5 RECURSOS DO PRODUTO

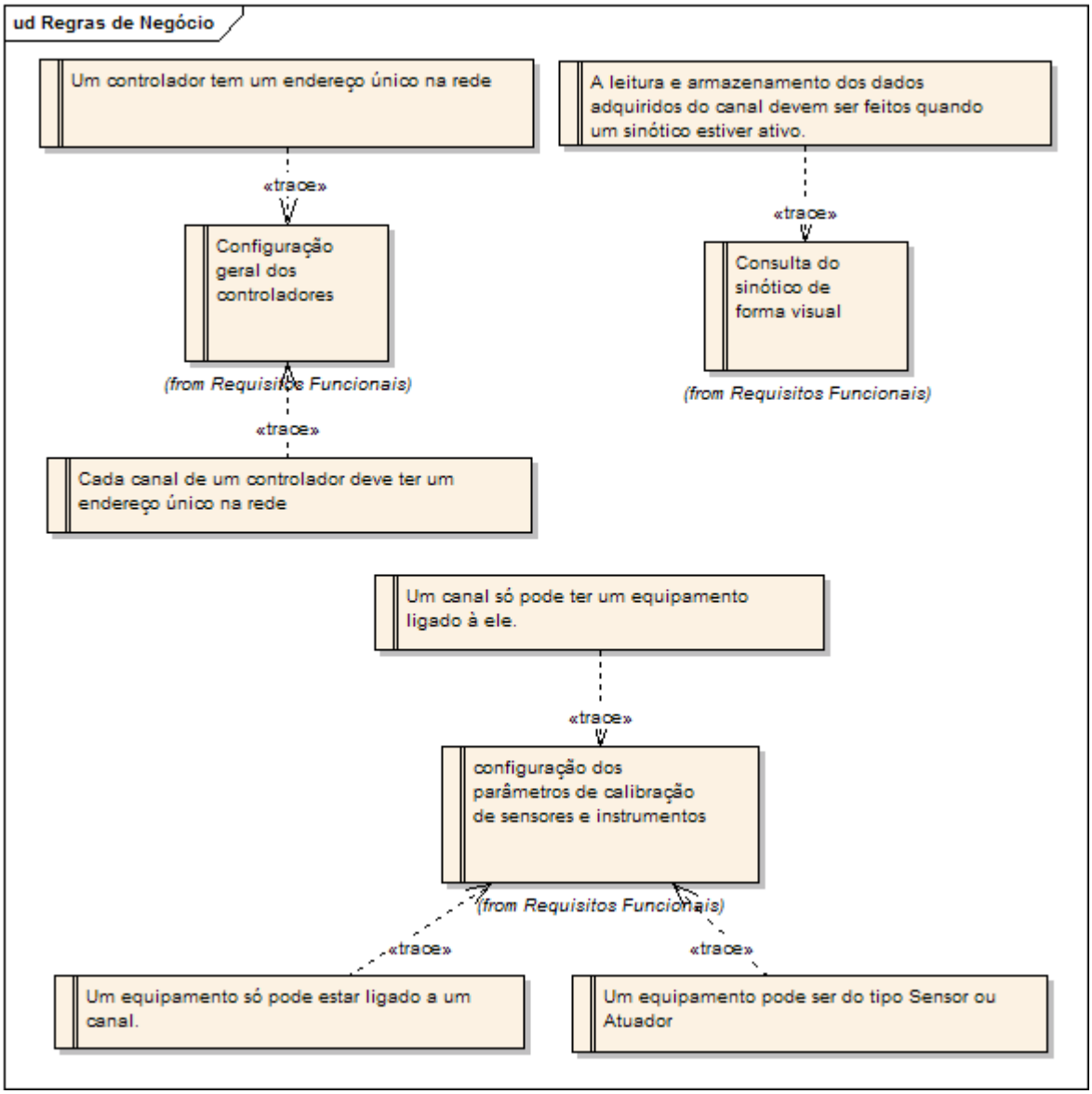
O sistema pode ser configurado de acordo com as necessidades do usuário, sendo possível cadastrar novos equipamentos, novos controladores e configurar vários tipos de visualizações dos processos através de sinóticos, que refletem a planta de uma fábrica, representados por painéis de sinal, de alerta, de percentual e gráficos.

É possível gerar relatórios de erros de comunicação durante as leituras dos canais e consultar históricos do sinal de cada equipamento com relação ao tempo.

# 6 OUTROS REQUISITOS DO PRODUTO

O *software* requer plataforma *Windows* e a instalação do banco de dados *Interbase* versão 6.0.

## APÊNDICE C – Regras de Negócio relacionadas com os requisitos funcionais



## APÊNDICE D – Rastreabilidade dos casos de uso