Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

RODRIGO SILVEIRA MARQUES

SISTEMA DE CONTROLE ACADÊMICO

Assis
2012

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (18) 3302-1055 homepage: www.fema.edu.br

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

RODRIGO SILVEIRA MARQUES

SISTEMA DE CONTROLE ACADÊMICO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis - IMESA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando: Rodrigo Silveira Marques
Orientador: Prof. Ms. Douglas Sanches da Cunha

Assis
2012

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

FICHA CATALOGRÁFICA

MARQUES, Rodrigo Silveira
Sistema de Controle Acadêmico / Rodrigo Silveira Marques. Fundação Educacional do Município de Assis – FEMA – Assis, 2012.
72p.

Orientador: Douglas Sanches da Cunha
Trabalho de Conclusão de Curso – Instituto de Ensino Superior de Assis – IMESA

1. Controle. 2. Acadêmico. 3. Música

CDD:001.61
Biblioteca da FEMA

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

RODRIGO SILVEIRA MARQUES

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis - IMESA, como requisito parcial à obtenção do Certificado de Conclusão, analisado pela seguinte Comissão Examinadora.

Orientador: Prof. Ms. Douglas Sanches da Cunha

Analisador 01: _____

Analisador 02: _____

DEDICATÓRIA

Dedico este trabalho aos meus pais, Renato e Maria, meu irmão, Ricardo, que me incentivaram e me deram todo apoio necessário.

Aos amigos que me ajudaram durante esse curso com toda a alegria e amizade.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus por esta longa caminhada.

Ao meu orientador Prof. Douglas Sanches da Cunha que me instruiu neste trabalho, sempre me ajudando e incentivando em momentos decisivos.

Ao Prof. Fábio Éder Cardoso que com muita atenção e profissionalismo proporcionou um melhor direcionamento a este trabalho.

Aos outros professores que de alguma forma me ensinaram e me deram força para estar concluindo este curso.

Aos meus familiares e amigos que me aconselharam e ajudaram para que pudesse me dedicar a esta faculdade.

RESUMO

Este sistema foi desenvolvido para uma escola de música, onde seu plano de negócio são ensinamentos na área musical, desde a parte prática como também a parte teórica.

O motivo da escolha do sistema deve-se a todos os controles e registros serem feitos manualmente.

Palavras-chave: Controle; Acadêmico; Música.

ABSTRACT

This system was developed for a music school, where her business plan are lessons in music, from the practical as well as theoretical.

The reason for the choice of system should be to all controls and records are made manually.

Keywords: Control; Academic; Music.

SUMÁRIO

Índice Pág.

1 - INTRODUCÃO

O Sistema S.C.A. (Sistema de Controle Acadêmico) foi desenvolvido para a *Escola de Música "Belo Som",* onde seu plano de negócio são ensinamentos na área musical desde a parte prática como também a parte teórica.

Envolve aulas ministradas para ensinamento de vários instrumentos e também prática de repertório e demais ícones da teoria musical.

A empresa possui atualmente 04 (quatro funcionários).

Os motivos da escolha do sistema é que tudo controlado na escola é feito manualmente como os cadastros, controle de freqüência, emissão de certificados e recibos, cartas de cobranças e congratulações aos alunos, controle do caixa, controle dos professores, controle das notas, controle das mensalidades e controle das matérias ministradas em aula.

Os objetivos do sistema é informatizar toda a parte de controle dos alunos e professores como:

- ✓ Cadastro completo de alunos e professores;
- ✓ Controle da freqüência as aulas;
- ✓ Controle das Mensalidades;
- ✓ Emissão de Certificados;
- ✓ Emissão de Recibos;
- ✓ Controle das Notas;
- ✓ Controle das matérias ministradas em aula;
- ✓ Controle do Movimento de Caixa;
- ✓ Emissão de cobranças e congratulações aos alunos;
- ✓ Controle das matérias ministradas em aula;

2 - METODOLOGIA E ANÁLISE

2.1 - Resumo

UML significa "Unified Modeling Language" e é a padronização das metodologias de desenvolvimento de sistemas baseados na orientação a objetos. A UML foi criada por três grandes desenvolvedores de sistemas orientados a objetos: Grady Booch, James Rumbaugh, e Ivar Jacobson, que já haviam criado outras notações de desenvolvimento de software. A UML incorpora as noções do desenvolvimento de software totalmente visual. Ela se baseia em diagramas que são modelados e classificados em visões de abstração. O desenvolvimento de um sistema em UML divide-se em 5 fases: análise de requisitos, análise, design, implementação (programação) e testes. A UML se propõe a ser a linguagem definitiva para modelagem de sistemas orientado a objetos por ser unificada e facilitar que grupos de desenvolvimentos de software interpretem de uma maneira correta e sem ambigüidades modelos gerados por outros analistas ou grupos de desenvolvimento. [H.E.E.]

Quando a "Unified Modeling Language" (UML) foi lançada, muitos desenvolvedores da área da orientação a objetos ficaram entusiasmados já que essa padronização proposta pela UML era o tipo de força que eles sempre esperaram.

A UML é muito mais que a padronização de uma notação. É também o desenvolvimento de novos conceitos não normalmente usados. Por isso e muitas outras razões, o bom entendimento da UML não é apenas aprender a simbologia e o seu significado, mas também significa aprender a modelar orientado a objetos no estado da arte.

UML foi desenvolvida por Grady Booch, James Rumbaugh, e Ivar Jacobson que são conhecidos como "os três amigos". Eles possuem um extenso conhecimento na área de modelagem orientado a objetos já que as três mais conceituadas metodologias de modelagem orientado a objetos foram eles que desenvolveram e a UML é a junção do que havia de melhor nestas três

metodologias adicionado novos conceitos e visões da linguagem. Veremos características de cada uma destas metodologias no desenvolver deste trabalho.

A UML aborda o caráter estático e dinâmico do sistema a ser analisado levando em consideração, já no período de modelagem, todas as futuras características do sistema em relação a utilização de "packages" próprios da linguagem a ser utilizada, utilização do banco de dados bem como as diversas especificações do sistema a ser desenvolvido de acordo com as métricas finais do sistema.

2.2 - Desenvolvimento de Softwares orientado a objetos

Os conceitos da orientação a objetos já vêm sido discutidos há muito tempo, desde o lançamento da 1ª linguagem orientada a objetos, a SIMULA. Vários "papas" da engenharia de software mundial como Peter Coad, Edward Yourdon e Roger Pressman abordaram extensamente a análise orientada a objetos como realmente um grande avanço no desenvolvimento de sistemas. Mas mesmo assim, eles citam que não existe (ou que não existia no momento de suas publicações) uma linguagem que possibilitasse o desenvolvimento de qualquer software utilizando a análise orientada a objetos.

Os conceitos que Coad, Yourdon, Pressman e tantos outros abordaram, discutiram e definiram em suas publicações foram que:

- ✓ A orientação a objetos é uma tecnologia para a produção de modelos que especifiquem o domínio do problema de um sistema.
- ✓ Quando construídos corretamente, sistemas orientados a objetos são flexíveis a mudanças, possuem estruturas bem conhecidas e provem a oportunidade de criar e implementar componentes totalmente reutilizáveis.
- ✓ Modelos orientado a objetos são implementados convenientemente utilizando uma linguagem de programação orientada a objetos. A engenharia de software orientada a objetos é muito mais que utilizar mecanismos de sua linguagem de programação, é saber utilizar da melhor forma possível todas as técnicas da modelagem orientada a objetos.

- ✓ A orientação a objetos não é só teoria, mas uma tecnologia de eficiência e qualidade comprovadas usada em inúmeros projetos e para construção de diferentes tipo de sistemas.
- ✓ A orientação a objetos requer um método que integre o processo de desenvolvimento e a linguagem de modelagem com a construção de técnicas e ferramentas adequadas.(R.P.]

2.3 - UML - A unificação dos métodos para a criação de um novo padrão

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

Existem várias metodologias de modelagem orientada a objetos que até o surgimento da UML causavam uma guerra entre a comunidade de desenvolvedores orientado a objetos. A UML acabou com esta guerra trazendo as melhores idéias de cada uma destas metodologias, e mostrando como deveria ser a migração de cada uma para a UML.

Será descrito a seguir algumas das principais metodologias que se tornaram populares nos anos 90:

- ✓ Booch - O método de Grady Booch para desenvolvimento orientado a objetos está disponível em muitas versões. Booch definiu a noção de que um sistema é analisado a partir de um número de visões, onde cada visão é descrita por um número de modelos e diagramas. O Método de Booch trazia uma simbologia complexa de ser desenhada a mão, continha também o processo pelo qual sistemas são analisados por macro e micro visões.
- ✓ OMT - Técnica de Modelagem de Objetos (Object Modelling Technique) é um método desenvolvido pela GE (General Electric) onde James Rumbaugh trabalhava. O método é especialmente voltado para o teste dos modelos, baseado nas especificações da análise de requisitos do sistema. O modelo total do sistema baseado no método OMT é composto pela junção dos

modelos de objetos, funcional e use-cases.

- ✓ OOSE/Objectory - Os métodos OOSE e o Objectory foram desenvolvidos baseados no mesmo ponto de vista formado por Ivar Jacobson. O método OOSE é a visão de Jacobson de um método orientado a objetos, já o Objectory é usado para a construção de sistemas tão diversos quanto eles forem. Ambos os métodos são baseados na utilização de use-cases, que definem os requisitos iniciais do sistema, vistos por um ator externo. O método Objectory também foi adaptado para a engenharia de negócios, onde é usado para modelar e melhorar os processos envolvidos no funcionamento de empresas.

Cada um destes métodos possui sua própria notação (seus próprios símbolos para representar modelos orientado a objetos), processos (que atividades são desenvolvidas em diferentes partes do desenvolvimento), e ferramentas (as ferramentas CASE que suportam cada uma destas notações e processos).

Diante desta diversidade de conceitos, "os três amigos", Grady Booch, James Rumbaugh e Ivar Jacobson decidiram criar uma Linguagem de Modelagem Unificada. Eles disponibilizaram inúmeras versões preliminares da UML para a comunidade de desenvolvedores e a resposta incrementou muitas novas idéias que melhoraram ainda mais a linguagem.

Os objetivos da UML são:

- ✓ A modelagem de sistemas (não apenas de software) usando os conceitos da orientação a objetos;
- ✓ Estabelecer uma união fazendo com que métodos conceituais sejam também executáveis;
- ✓ Criar uma linguagem de modelagem usável tanto pelo homem quanto pela máquina.

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

semântica da linguagem representada em metamodelos.(H.E.E.]

2.4 - Uso da UML

A UML é usada no desenvolvimento dos mais diversos tipos de sistemas. Ela abrange sempre qualquer característica de um sistema em um de seus diagramas e é também aplicada em diferentes fases do desenvolvimento de um sistema, desde a especificação da análise de requisitos até a finalização com a fase de testes.

O objetivo da UML é descrever qualquer tipo de sistema, em termos de diagramas orientado a objetos. Naturalmente, o uso mais comum é para criar modelos de sistemas de software, mas a UML também é usada para representar sistemas mecânicos sem nenhum software. Aqui estão alguns tipos diferentes de sistemas com suas características mais comuns:

- ✓ Sistemas de Informação: Armazenar, pesquisar, editar e mostrar informações para os usuários. Manter grandes quantidades de dados com relacionamentos complexos, que são guardados em bancos de dados relacionais ou orientados a objetos.
- ✓ Sistemas Técnicos: Manter e controlar equipamentos técnicos como de telecomunicações, equipamentos militares ou processos industriais. Eles devem possuir interfaces especiais do equipamento e menos programação de software de que os sistemas de informação. Sistemas Técnicos são geralmente sistemas real-time.(R.P.]

2.5 - Fases do Desenvolvimento de um Sistema em UML

Existem cinco fases no desenvolvimento de sistemas de software: análise de requisitos, análise, design (projeto), programação e testes. Estas cinco fases não devem ser executadas na ordem descrita acima, mas concomitantemente de forma que problemas detectados numa certa fase modifiquem e melhorem as fases

desenvolvidas anteriormente de forma que o resultado global gere um produto de alta qualidade e performance. A seguir falaremos sobre cada fase do desenvolvimento de um sistema em UML: [R.P.]

2.5.1 - Análise de Requisitos

Esta fase captura as intenções e necessidades dos usuários do sistema a ser desenvolvido através do uso de funções chamadas "use-cases". Através do desenvolvimento de "use-case", as entidades externas ao sistema (em UML chamados de "atores externos") que interagem e possuem interesse no sistema são modelados entre as funções que eles requerem, funções estas chamadas de "use-cases". Os atores externos e os "use-cases" são modelados com relacionamentos que possuem comunicação associativa entre eles ou são desmembrados em hierarquia. Cada "use-case" modelado é descrito através de um texto, e este especifica os requerimentos do ator externo que utilizará este "use-case". O diagrama de "use-cases" mostrará o que os atores externos, ou seja, os usuários do futuro sistema deverão esperar do aplicativo, conhecendo toda sua funcionalidade sem importar como esta será implementada. A análise de requisitos também pode ser desenvolvida baseada em processos de negócios, e não apenas para sistemas de software.[R.P.]

2.5.2 - Análise

A fase de análise contextualiza as primeiras abstrações (classes e objetos) e mecanismos que estarão presentes no domínio do problema. As classes são modeladas e ligadas através de relacionamentos com outras classes, e são descritas no Diagrama de Classe. As colaborações entre classes também são mostradas neste diagrama para desenvolver os "use-cases" modelados anteriormente, estas colaborações são criadas através de modelos dinâmicos em UML. Na análise, só serão modeladas classes que pertençam ao domínio principal do problema do software, ou seja, classes técnicas que gerenciem banco de dados, interface, comunicação, concorrência e outros não estarão presentes neste

diagrama. [R.P.]

2.5.3 - Design (Projeto)

Na fase de design, o resultado da análise é expandido em soluções técnicas.

Novas classes serão adicionadas para prover uma infra-estrutura técnica: a interface do usuário e de periféricos, gerenciamento de banco de dados, comunicação com outros sistemas, dentre outros. As classes do domínio do problema modeladas na fase de análise são mescladas nessa nova infra-estrutura técnica tornando possível alterar tanto o domínio do problema quanto a infra-estrutura. O design resulta no detalhamento das especificações para a fase de programação do sistema. [R.P.]

2.5.4 - Programação

Na fase de programação, as classes provenientes do design são convertidas para o código da linguagem orientada a objetos escolhida (a utilização de linguagens procedurais é extremamente não recomendada). Dependendo da capacidade da linguagem usada, essa conversão pode ser uma tarefa fácil ou muito complicada. No momento da criação de modelos de análise e design em UML, é melhor evitar traduzi-los mentalmente em código. Nas fases anteriores, os modelos criados são o significado do entendimento e da estrutura do sistema, então, no momento da geração do código onde o analista conclua antecipadamente sobre modificações em seu conteúdo, seus modelos não estarão mais demonstrando o real perfil do sistema. A programação é uma fase separada e distinta onde os modelos criados são convertidos em código. [R.P.]

2.5.5 - Testes

Um sistema normalmente é rodado em testes de' unidade, integração, e aceitação. Os testes de unidade são para classes individuais ou grupos de classes e são geralmente testados pelo programador. Os testes de integração são aplicados já usando as classes e componentes integrados para se confirmar se as classes estão cooperando uma com as outras como especificado nos modelos. Os testes de aceitação observam o sistema como uma" caixa preta" e verificam se o sistema está funcionando como o especificado nos primeiros diagramas de "use-cases".

O sistema será testado pelo usuário final e verificará se os resultados mostrados estão realmente de acordo com as intenções do usuário final. [R.P.]

2.6 - Modelos de Elementos

Os conceitos usados nos diagramas são chamados de modelos de elementos.

Um modelo de elemento é definido com a semântica, a definição formal do elemento com o exato significado do que ele representa sem definições duvidosas ou ambíguas e também define sua representação gráfica que é mostrada nos diagramas da UML. Um elemento pode existir em diversos tipos de diagramas, mas existem regras que definem que elementos podem ser mostrados em que tipos de diagramas. Alguns exemplos de modelos de elementos são as classes, objetos, estados, pacotes e componentes. Os relacionamentos também são modelos de elementos, e são usados para conectar outros modelos de elementos entre si. Todos os modelos de elementos serão definidos e exemplificados a seguir. [R.P.]

2.6.1 - Classes

Uma classe é a descrição de um tipo de objeto. Todos os objetos são instâncias de classes, onde a classe descreve as propriedades e comportamentos daquele objeto. Objetos só podem ser instanciados de classes. Usam-se classes

para classificar os objetos que identificamos no mundo real. Tomando como exemplo Charles Darwin, que usou classes para classificar os animais conhecidos, e combinou suas classes por herança para descrever a "Teoria da Evolução". A técnica de herança entre classes é também usada em orientação a objetos.

Uma classe pode ser a descrição de um objeto em qualquer tipo de sistema sistemas de informação, técnicos, integrados real-time, distribuídos, software etc. Num sistema de software, por exemplo, existem classes que representam entidades de software num sistema operacional como arquivos, programas executáveis, janelas, barras de rolagem, etc.

Identificar as classes de um sistema pode ser complicado, e deve ser feito por experts no domínio do problema a que o software modelado se baseia. As classes devem ser retiradas do domínio do problema e serem nomeadas pelo que elas representam no sistema. Quando procuramos definir as classes de um sistema, existem algumas questões que podem ajudar a identificá-las:

- ✓ Existem informações que devem ser armazenadas ou analisadas? Se existir alguma informação que tenha que ser guardada, transformada ou analisada de alguma forma, então é uma possível candidata para ser uma classe.
- ✓ Existem sistemas externos ao modelado? Se existir, eles deverão ser vistos como classes pelo sistema para que possa interagir com outros externos.
- ✓ Existem classes de bibliotecas, componentes ou modelos externos a serem utilizados pelo sistema modelado? Se sim, normalmente essas classes, componentes e modelos conterão classes candidatas ao nosso sistema.
- ✓ Qual o papel dos atores dentro do sistema? Talvez o papel deles possa ser visto como classes, por exemplo, usuário, operador, cliente e daí por diante.

Em UML as classes são representadas por um retângulo dividido em três compartimentos: o compartimento de nome, que conterá apenas o nome da classe modelada, o de atributos, que possuirá a relação de atributos que a classe possui em sua estrutura interna, e o compartimento de operações, que serão os métodos de manipulação de dados e de comunicação de uma classe com outras do sistema. A sintaxe usada em cada um destes compartimentos é independente de qualquer linguagem de programação, embora pode ser usadas outras sintaxes como a do

C++, Java, e etc. [R.P.]

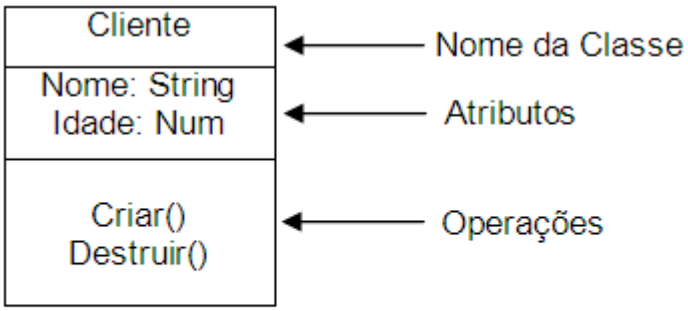


Figura 1

2.6.2 - Objetos

Um objeto é um elemento que podemos manipular, acompanhar seu comportamento, criar, destruir etc. Um objeto existe no mundo real. Pode ser uma parte de qualquer tipo de sistema, por exemplo, uma máquina, uma organização, ou negócio. Existem objetos que não encontramos no mundo real, mas que podem ser vistos de derivações de estudos da estrutura e comportamento de outros objetos do mundo real.

Em UML um objeto é mostrado como uma classe só que seu nome (do objeto) é sublinhado, e o nome do objeto pode ser mostrado opcionalmente precedido do nome da classe. [R.P.]

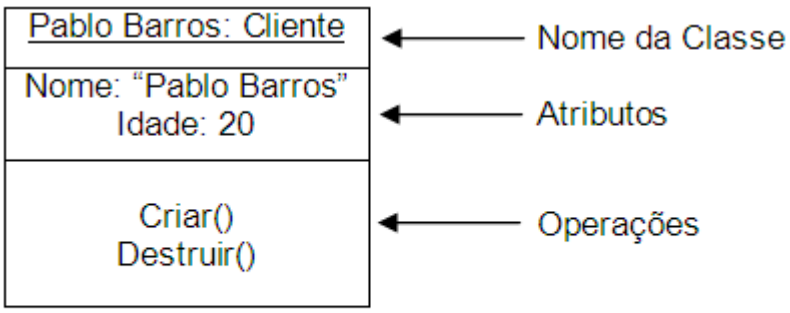


Figura 2

2.6.3 - Estados

Todos os objetos possuem um estado que significa o resultado de atividades executadas pelo objeto, e é normalmente determinada pelos valores de seus atributos e ligações com outros objetos.

Um objeto muda de estado quando acontece algo, o fato de acontecer alguma coisa com o objeto é chamado de evento. Através da análise da mudança de estados dos tipos de objetos de um sistema, podemos prever todos os possíveis comportamentos de um objetos de acordo com os eventos que o mesmo possa sofrer. [R.P.]

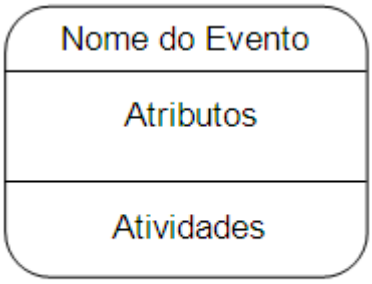


Figura 3

2.6.4 - Pacotes

Pacote é um mecanismo de agrupamento, onde todos os modelos de elementos podem ser agrupados. Em UML, um pacote é definido como: "Um mecanismo de propósito geral para organizar elementos semanticamente relacionados em grupos." Todos os modelos de elementos que são ligados ou referenciados por um pacote são chamados de "Conteúdo do pacote". Um pacote possui vários modelos de elementos, e isto significa que estes não podem ser incluídos em outros pacotes. [H.E.E.]

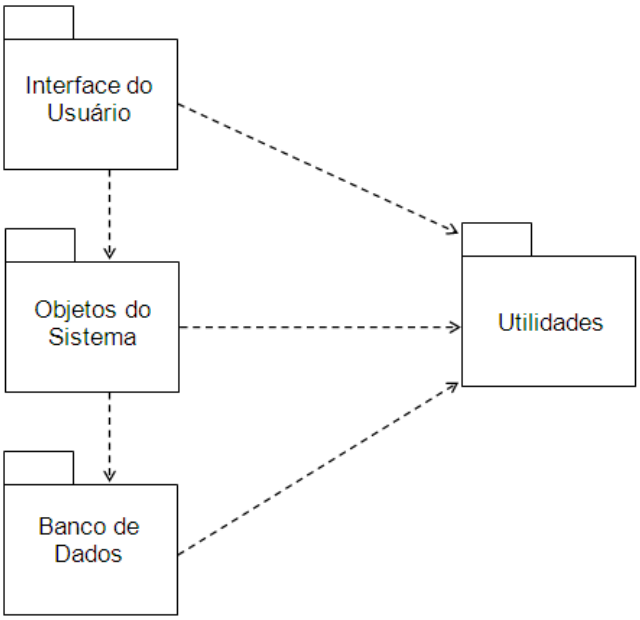


Figura 4

Pacotes podem importar modelos de elementos de outros pacotes. Quando um modelo de elemento é importado, refere-se apenas ao pacote que possui o elemento. Na grande maioria dos casos, os pacotes possuem relacionamentos com outros pacotes. Embora estes não possuam semânticas definidas para suas instâncias. Os relacionamentos permitidos entre pacotes são de dependência, refinamento e generalização (herança). [H.E.E.]

2.7 - Relacionamentos

Os relacionamentos ligam as classes/objetos entre si criando relações lógicas entre estas entidades. Os relacionamentos podem ser dos seguintes tipos:

- ✓ Associação: É uma conexão entre classes, e também significa que é uma conexão entre objetos daquelas classes. Em UML, uma associação é definida com um relacionamento que descreve uma série de ligações, onde a ligação é definida como a semântica entre as duplas de objetos ligados.
- ✓ Generalização: É um relacionamento de um elemento mais geral e outro

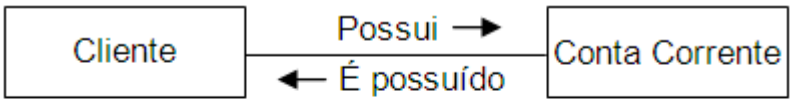
maIS específico. O elemento mais específico pode conter apenas informações adicionais. Uma instância (um objeto é uma instância de uma classe) do elemento mais específico pode ser usada onde o elemento mais geral seja permitido.

- ✓ Dependência e Refinamentos: Dependência é um relacionamento entre elementos, um independente e outro dependente. Uma modificação é um elemento independente afetará diretamente elementos dependentes do anterior. Refinamento é um relacionamento entre duas descrições de uma mesma entidade, mas em níveis diferentes de abstração.

Abordaremos agora cada tipo de relacionamento e suas respectivas sub-divisões: [R.P.]

2.7.1 - Associações

Uma associação representa que duas classes possuem uma ligação (link) entre elas, significando por exemplo que elas "conhecem uma a outra", "estão conectadas com", "para cada X existe um Y" e assim por diante. Classes e associações são muito poderosas quando modeladas em sistemas complexos.



Duas Classes se relacionando por associação normal

Figura 5

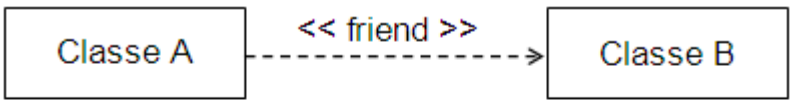
Obs.: Existem outros tipos de ASSOCIAÇÕES [R.P.]

2.7.2 - Dependência e Refinamentos

Além das associações, existem ainda dois tipos de relacionamentos em

UML. O relacionamento de dependência é uma conexão semântica entre dois modelos de elementos, um independente e outro dependente. Uma mudança no elemento independente irá afetar o modelo dependente. Como no caso anterior com generalizações, os modelos de elementos podem ser uma classe, um pacote, um use-case e assim por diante. Quando uma classe recebe um objeto de outra classe como parâmetro, uma classe acessa o objeto global da outra. Nesse caso existe uma dependência entre estas duas classes, apesar de não ser explícita.

Uma relação de dependência é simbolizada por uma linha tracejada com uma seta no final de um dos lados do relacionamento. E sobre essa linha o tipo de dependência que existe entre as duas classes. As classes "Amigas" provenientes do C++ são um exemplo de um relacionamento de dependência.



Exemplo de uma dependência entre classes

Figura 6

Os refinamentos são um tipo de relacionamento entre duas descrições de uma mesma coisa, mas em níveis de abstração diferentes e podem ser usados para modelar diferentes implementações de uma mesma coisa (uma implementação simples e outra mais complexa, mas também mais eficiente). [R.P.]

2.8 - Diagramas

Os diagramas utilizados pela UML são compostos de nove tipos: diagrama de use case, de classes, de objeto, de estado, de seqüência, de colaboração, de atividade, de componente e o de execução, mas aqui sós estaremos falando dos dois primeiws citados.

Todos os sistemas possuem uma estrutura estática e um comportamento dinâmico. A UML suporta modelos estáticos (estrutura estática), dinâmicos (comportamento dinâmico) e funcional. A Modelagem estática é suportada pelo diagrama de classes e de objetos, que consiste nas classes e seus relacionamentos. Os relacionamentos podem ser de associações, herança (generalização), dependência ou refinamentos. Os modelamentos dinâmicos são suportados pelos diagramas de estado, seqüência, colaboração e atividade. E o modelamento funcional é suportado pelos diagramas de componente e execução. Abordaremos agora cada um destes tipos de diagrama: [R.P.]

2.9 - Diagrama Use-Case

A modelagem de um diagrama use-case é uma técnica usada para descrever e definir os requisitos funcionais de um sistema. Eles são escritos em termos de atores externos, use-cases e o sistema modelado. Os atores representam o papel de uma entidade externa ao sistema como um usuário, um hardware, ou outro sistema que interage com o sistema modelado. Os atores iniciam a comunicação com o sistema através dos use-cases, onde o use-case representa uma seqüência de ações executadas pelo sistema e recebe do ator que lhe utiliza dados tangíveis de um tipo ou formato já conhecido, e o valor de resposta da execução de um use-case (conteúdo) também já é de um tipo conhecido, tudo isso é definido juntamente com o use-case através de texto de documentação.

Atores e use-cases são classes. Um ator é conectado a um ou mais use-cases através de associações, e tanto atores quanto use-cases podem possuir relacionamentos de generalização que definem um comportamento comum de herança em superclasses especializadas em subclasses.

O uso de use-cases em colaborações é muito importante, onde estas são a descrição de um contexto mostrando classes/objetos, seus relacionamentos e sua interação exemplificando como as classes/objetos interagem para executar uma atividade específica no sistema. Uma colaboração é descrita por diagramas de atividades e um diagrama de colaboração.

Quando um use-case é implementado, a responsabilidade de cada passo da execução deve ser associada às classes que participam da colaboração, tipicamente especificando as operações necessárias dentro destas classes juntamente com a definição de como elas irão interagir. Um cenário é uma instância de um use-case, ou de uma colaboração, mostrando o caminho específico de cada ação. Por isso, o cenário é um importante exemplo de um use-case ou de uma colaboração. Quando visto a nível de um use-case, apenas a interação entre o ator externo e o use-case é vista, mas já observando a nível de uma colaboração, toda as interações e passos da execução que implementam o sistema serão descritos e especificados.

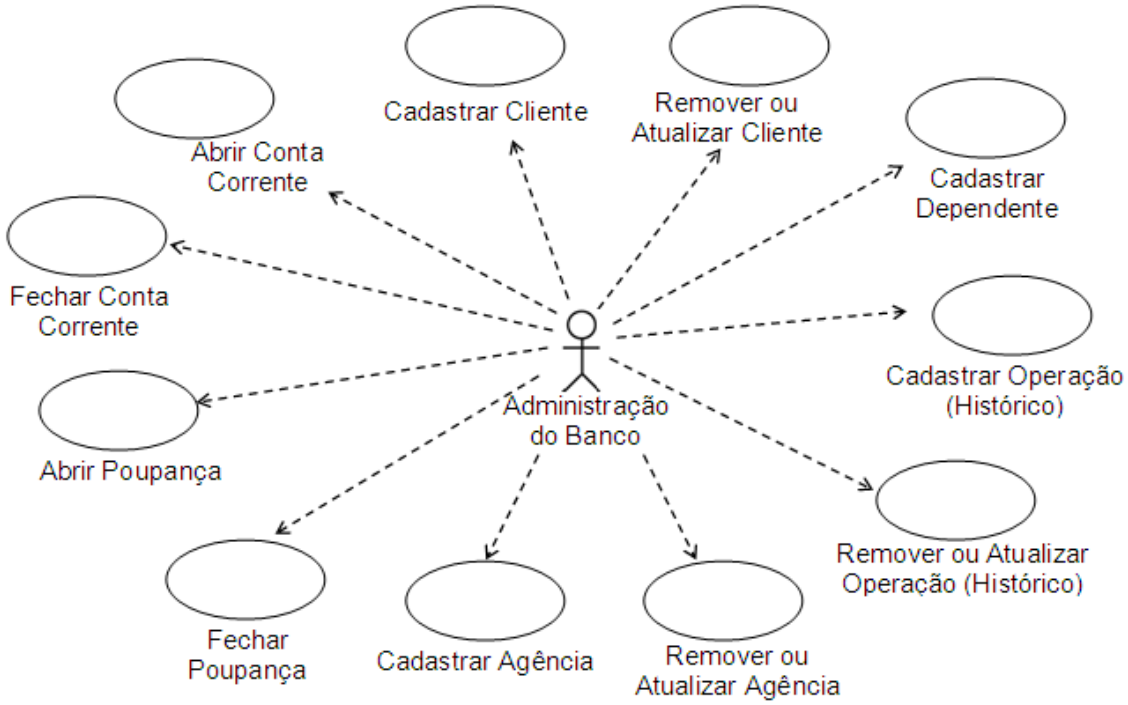


Figura 7

O diagrama de use-cases acima demonstra as funções de um ator externo de um sistema de controle bancário de um banco fictício que foi modelado no estudo de caso no final deste trabalho. O diagrama especifica que funções o administrador do banco poderá desempenhar. Pode-se perceber que não existe

nenhuma preocupação com a implementação de cada uma destas funções, já que este diagrama apenas se resume a determinar que funções deverão ser suportadas pelo sistema modelado. [R.P.]

2.10 - *Diagrama de Classes*

O diagrama de classes demonstra a estrutura estática das classes de um sistema onde estas representam as "coisas" que são gerenciadas pela aplicação modelada. Classes podem se relacionar com outras através de diversas maneiras: associação (conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe), ou em pacotes (classes agrupadas por características similares). Todos estes relacionamentos são mostrados no diagrama de classes juntamente com as suas estruturas internas, que são os atributos e operações. O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. Um sistema normalmente possui alguns diagramas de classes, já que não são todas as classes que estão inseridas em um único diagrama e uma certa classes pode participar de vários diagramas de classes.

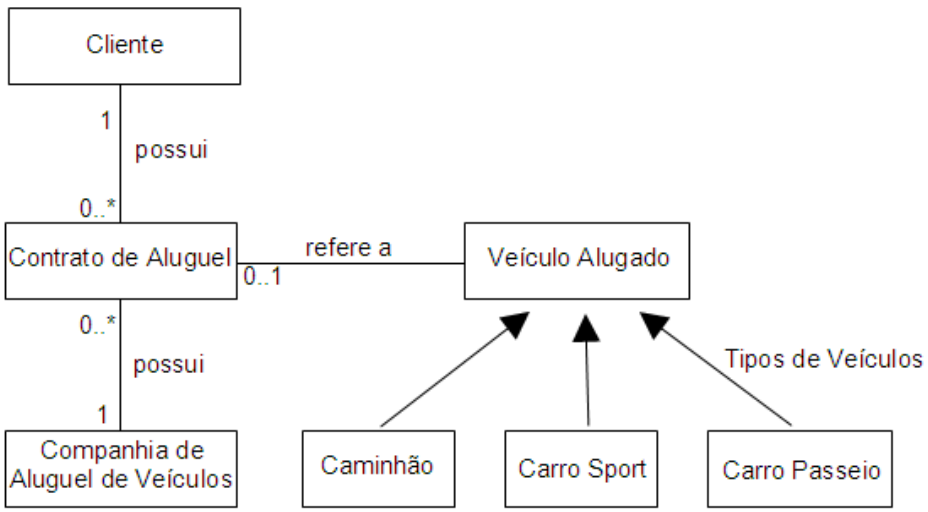


Figura 8

Uma classe num diagrama pode ser diretamente implementada utilizando-se uma linguagem de programação orientada a objetos que tenha suporte direto

para construção de classes. Para criar um diagrama de classes, as classes têm que estar identificadas, descritas e relacionadas entre si. [R.P.]

3- LINGUAGEM DE IMPLEMENTAÇÃO

3.1 - Visão Geral do Delphi

O Delphi é agora a mais poderosa forma de criar aplicações genéricas para o ambiente Windows. Muito embora seja visto como a melhor forma de criara aplicações Cliente/Servidor, ele também é excelente para criar aplicações mais genéricas, sem acesso a banco de dados. Você pode criar um programa de planilha eletrônica ou processador de texto no Delphi. Não existem limitações intrínsecas para suas capacidades. Você pode até mesmo criar um ambiente de desenvolvimento de aplicações dentro do Delphi, mas a Borland já o venceu nisso. Como você já deve Ter ouvido, o Delphi foi, ele próprio, escrito na linguagem do Delphi (Object Pascal), e o ambiente inteiro é uma aplicação Delphi. Isto mostra o quanto esta ferramenta é para propósitos genéricos!

A lista das vantagens que tornam o Delphi um produto tão aclamado é extensa.

Ele incorpora tudo o que os programadores aprenderam na procura de melhores abordagens para o desenvolvimento de aplicações Windows. Como disse J.D. Hildbrand no Windows Tech Journal, "I1's going to change our lives, you know." (Ele irá mudar nossas vidas). Eis uma lista parcial do conjunto de recursos exclusivos do Delphi:

- ✓ O Compilador otimizador mais veloz do mundo;
- ✓ Executáveis puros, sem a necessidade de bibliotecas de run-time para a distribuição de aplicações;
- ✓ Totalmente orientado a objeto, com uma linguagem bastante respeitada e sólida como base (Object Pascal);
- ✓ A habilidade de criar componentes nativos dentro do Delphi;
- ✓ Ferramentas visuais, two - way tools;
- ✓ Suporte a manipulação de exceção na linguagem, o que permite a criação de aplicações mais robustas e com mais qualidade;
- ✓ Incomparável conectividade a banco de dados através do Borland Database Engine.

Upsizing com menor esforço, de banco de dados locais para bancos de dados cliente/servidor.

Existem três tipos distintos de Delphi disponíveis: Standard, Professional

a versão Cliente/Servidor. [B.I.]

3.2 - Versão Standard

A versão Standard do Delphi inclui a maior parte dos recursos. Na realidade, o produto básico é exatamente o mesmo, não importa a versão que se utilize. A diferença está nas ferramentas adicionais que você recebe em cada pacote.

A Versão Standard inclui os seguinte softwares:

- ✓ O IDE e compilador do Delphi (incluindo um compilador de linha de comando);
- ✓ O Borland Database Engine, que inclui o dBASE e Paradox;
- ✓ A Visual Component Library (VCL) com mais de 90 componentes;
- ✓ O Database Desktop;
- ✓ O SQL Database Explorer. [B.I.]

3.3 - Versão Professional

A Borland CrIOU a edição Developer do Delphi para programadores que precisam de mais funcionalidade do que acompanha a edição Standard, mas não precisam de todos os recursos extra de banco de dados do pacote Client/Server. Ele possui tudo o que acompanha a edição Standard, mais:

- ✓ Código fonte da VCL;
- ✓ InstallShield Express;

- ✓ Servidor Local InterBAse;
- ✓ Conectividade com ODBC;
- ✓ Abertura para desenvolvimento em grupo com InterSolv PVCS;
- ✓ OCXs de exemplo para criação de gráficos;
- ✓ Active X, etc; [B.I.]

3.4 - Versão Client/Server

A versão Client/Server do Delphi inclui todos os softwares acima, além de:

- ✓ Drivers SQL Links nativos para Orade, Sybase, InterBase, SQL Server, DB2 e Informix;
- ✓ SQL Database Explorer para gerenciamento de dados do servidor;
- ✓ Visual Query Builder;
- ✓ Data Migration Tool;
- ✓ Servidor de banco de dados InterBase com licença para 2 usuários;
- ✓ InterSolv PVCS para desenvolvimento em grupo;

O ambiente de desenvolvimento do Delphi é exatamente o mesmo, não importando qual versão você tenha. A única diferença está nas ferramentas que acompanham o produto, de fato, você pode comprar a versão Professional, e posteriormente atualizar para a versão Client/Server Suite simplesmente pagando o custo da diferença. [B.I.]

3.5 - Alguns Objetos Básicos do Delphi

3.5.1- Formulário

O Coração de todo aplicativo Windows é o formulário. Talvez você conheça o formulário como "janela". A janela é aquele objeto sobre o qual o resto do

ambiente Windows é construído. Se os padrões não forem mudados, o Delphi assumirá que você tem um formulário em cada projeto e exibirá um em branco, sempre que for inicializado. Entretanto, você pode escolher uma das opções da área de trabalho em Tools | Options. Você pode salvar a posição de suas janelas, assim como as janelas de código abertas na última vez. Isso é feito de acordo com o projeto.

Você pode Ter visto o formulário em muitos de seus diferente papéis. Eles podem existir no estado modal e sem modo. Estou certo de que você já os viu. A janela modal é aquela que fica sobre todas as outras e deve ser fechada antes que as outras janelas sejam acessadas. Uma janela sem modo pode ser movida para uma lado e tratada a qualquer momento. [D.O.]

3.5.2 - Guia de Propriedades

A parte de propriedades do Object Inspector (ver figura abaixo) permite que você veja e modifique as propriedades de um objeto. Dê um clique na janela de formulário e depois observe os atributos presentes na guia Properties do Object Inspector. [D.0.]

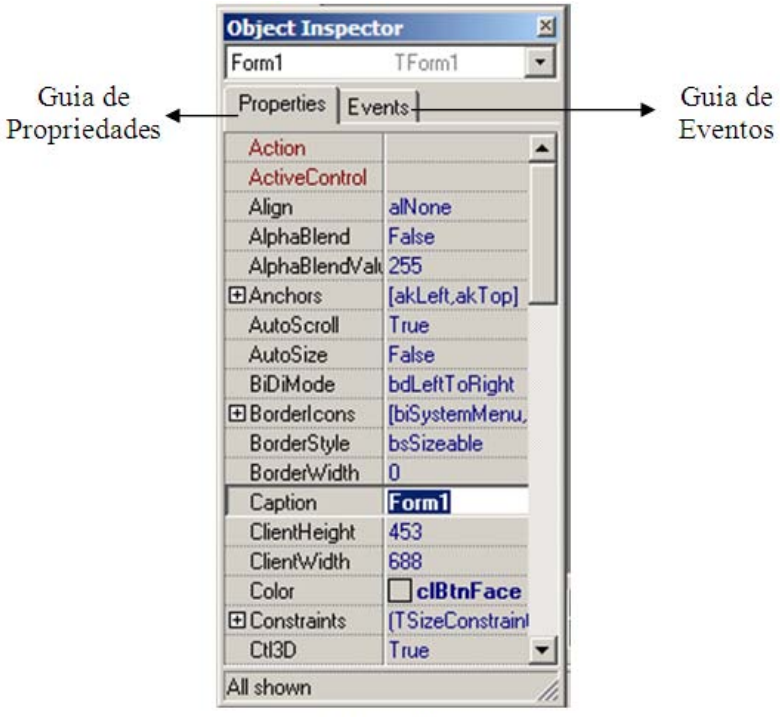


Figura 9

Além da guia de propriedades, o Object Inspector possui um *Guia de Eventos.* A Guia de Eventos é a outra metade da vida do Object Inspector. Ela está relacionada ao programador e aos eventos a que esse objeto responde. Por exemplo, se você precisa que um aplicativo faça algo especial quando a janela se fecha, pode usar o evento *OnClose* do formulário, para isso. [D.O.]

3.5.3 - Estrutura de Menus do Delphi

A estrutura de menus do Delphi dá acesso a um rico conjunto de ferramentas que podem ajudá-lo muito no ambiente de desenvolvimento RAO. Cada menu (figura abaixo) contém opções específicas para a o trabalho e configuração do Ambiente. [D.O.]

Figura 10

3.5.4 - SpeedBar e Paleta de Componentes

A *SpeedBar,* apresentada na figura abaixo, é projetada para ajudá-lo a acessar as funções que você mais utiliza fácil e rapidamente. A configuração padrão fornece 16 itens do que a Borland considera os mais usados. São itens que também estão disponíveis no menu do Oelphi, e são colocados na SpeedBar apenas para acelerar o acesso a elas.

SpeedBar Figura 11

A paleta de componentes (figura abaixo) é o "inventário visual" de sua Biblioteca de Componentes Visuais (VCL). Ela permite que você classifique seus

componentes visuais em determinados grupos. Por padrão, os componentes são agrupados em linhas funcionais; isto é, com todos os componentes de acesso aos dados juntos etc. Esses agrupamentos ou *páginas* são identificadas como guias rotuladas. [D.O.]

Palheta de Componentes

Figura 12

Além do Delphi padrão para o desenvolvimento do Estágio, está sendo usado uma biblioteca adicional chama da RXLIB versão 2.60 para a implementação do programa. Está biblioteca foi incorporada através de um arquivo pego em site oficial da Borland, ou seja, é um objeto original da Borland. [D.O.]

4 - ANÁLISE

4.1 - Lista de Eventos

Nº	DESCRIÇÃO	MENSAGEM	USE CASE
1	Secretária cadastra Aluno	MSG01	Cadastrar Aluno
2	Secretária transfere Aluno para Arquivo Morto	MSG02	Transfere Aluno
3	Secretária cadastra Curso	MSG03	Cadastrar Curso
4	Secretária cadastra Turma	MSG04	Cadastrar Turma
5	Secretária cadastra Professor	MSG05	Cadastrar Professor
6	Secretária faz a Movimentação do Caixa	MSG06	Controlar Caixa
7	Secretária controla a freqüência do Aluno	MSG07	Controlar Freqüência
8	Secretária controla as Notas do Aluno	MSG08	Controlar Notas
9	Secretária controla as Mensalidades do Aluno	MSG09	Controlar Mensalidades
10	Secretária controla a matéria dada pelo Professor	MSG10	Controlar Matéria
11	Secretária emite certificado	MSG11	Emitir Certificado
12	Secretária emite o recibo	MSG12	Emitir Recibo
13	Secretária emite a Ficha de Inscrição	MSG13	Emitir Ficha-Inscrição
14	Secretária emite a Lista de Chamada	MSG14	Emitir Lista Chamada
15	Secretária emite o Livro Ponto para o Professor	MSG15	Emitir Livro Ponto
16	Secretária emite o Fechamento do Caixa	MSG16	Emitir Fechamento Caixa
17	Secretária emite o Relatório dos Alunos Cadastrados por Curso	MSG17	Emitir Alunos do Curso
18	Secretária emite o Relatório dos Alunos Cadastrados por Turma	MSG18	Emitir Alunos da Turma
19	Secretária emite a Lista de Aniversariantes do Mês	MSG19	Emitir Aniversariantes
20	Secretária emite a Lista de Alunos com mensalidades atrasadas	MSG20	Emitir Mensalidades Atrasadas

21	Secretária emite o Relatório sobre o Professor	MSG21	Emitir Dados do Professor
22	Secretária emite o número de aulas dadas pelo Professor por Turma	MSG22	Emitir Número de Aulas
23	Secretária emite Relatório descriminando a matéria dada pelo Professor em aula	MSG23	Emitir Matéria por Aula
24	Secretária emite Relatório descriminando a matéria dada pelo Professor em aula por Turma	MSG24	Emitir Matéria por Turma

4.2 - Descrição das USE-CASES.

Use Case Cadastrar Aluno

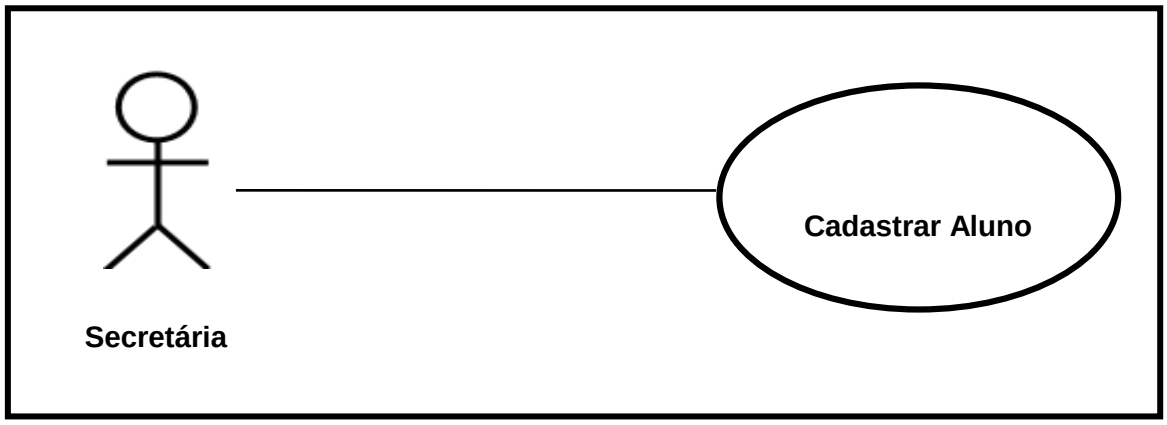


Figura 13 - Caso de Uso: Cadastrar Aluno

Passos

1 - Ler todas as informações do Aluno (Nome, Endereço, Cidade, Estado, Fone, RG, CPF, Nome do Pai, RG do Pai, Nome da Mãe, RG da Mãe)

Se Aluno Cadastrado => MSG01 "Aluno Cadastrado com Sucesso" Exceção

Se Aluno já Cadastrado => MSG01 "Aluno já Cadastrado"

Se algum campo não informado => MSG01 "Campo não Informado"

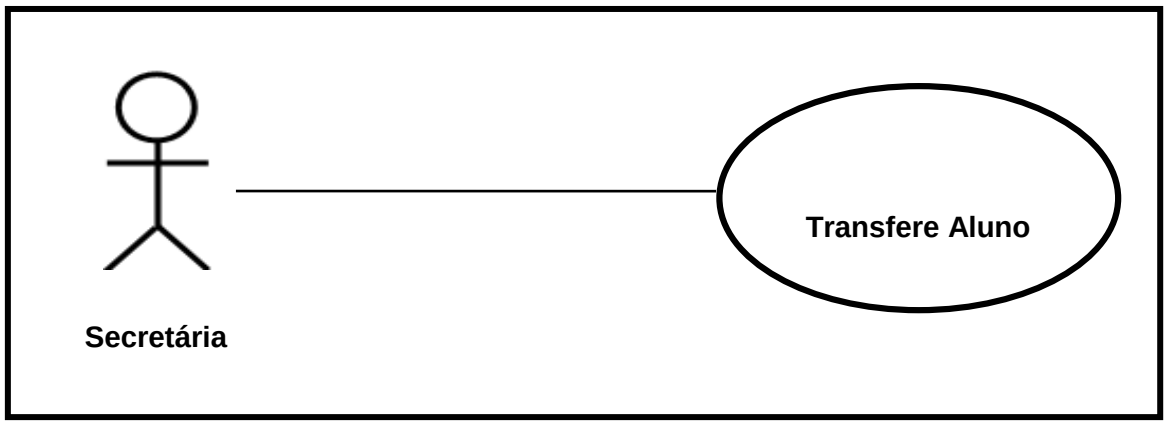


Figura 14 - Caso de Uso: Transfere Aluno

Use Case Transfere Aluno

Passos

1- Identificar os Alunos que concluiram o curso 2- Calcular as notas e freqüência

3- Transferir os Alunos com as notas e freqüências do Cadastro de Aluno para o Arquivo Morto

Se Aluno transferido => MSG02 "Aluno transferido com sucesso" Exceção

Se notas não encontrada => MSG02 "Notas não encontradas. Falar com o Professor"

Se freqüência não encontrada => MSG02 "Freqüência não encontrada. Falar com o Professor"

Se Aluno não concluiu o Curso => MSG02 "O Aluno não concluiu o Curso" Se Aluno em débito com a Escola => MSG02 "Aluno transferido com débito. Emitir Certificado só com quitação da divida"

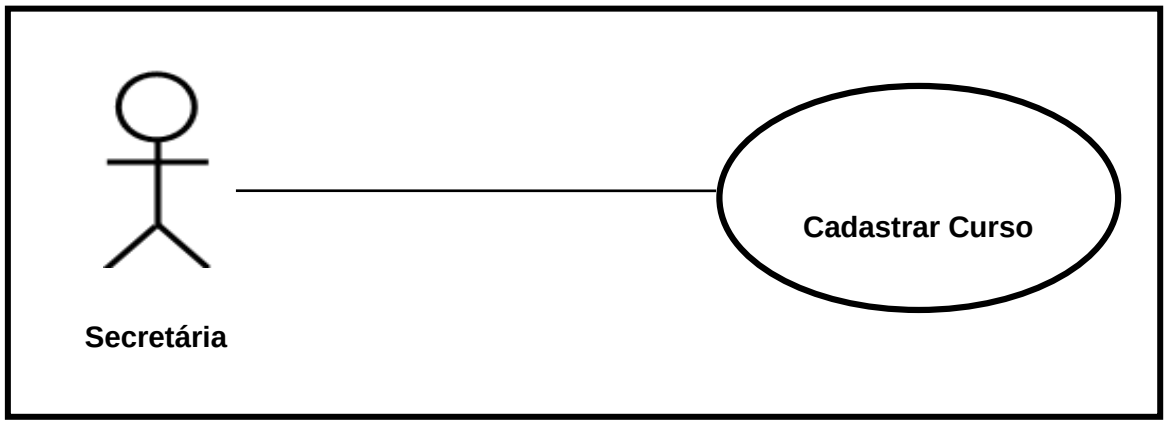


Figura 15 - Caso de Uso: Cadastrar Curso

Use Case Cadastrar Curso

Passos

1 - Ler o Código e a Descrição do Curso

Se Curso cadastrado => MSG03 "Curso Cadastrado com Sucesso"

Exceção

Se código já cadastrado ~> MSG03 "Já existe um Curso cadastrado com esse código"

Se descrição já cadastrada => MSG03 "Curso já está cadastrado com outro código"

Se algum campo não informado => MSG03 "Campo não informado"

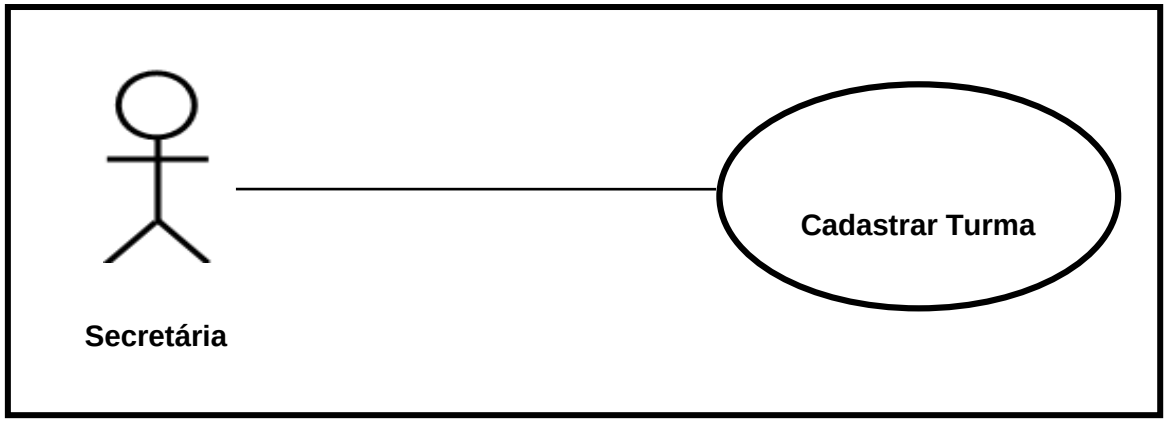


Figura 16 - Caso de Uso: Cadastrar Turma

Use Case Cadastrar Turma

Passos

1 - Ler o Código do Curso, Código do Professor, dia das aulas, hora de início e fim da término da aula

Se Turma cadastrada => MSG04 "Turma Cadastrada com Sucesso"

Exceção

Se algum campo não informado => MSG04 "Campo não informado"

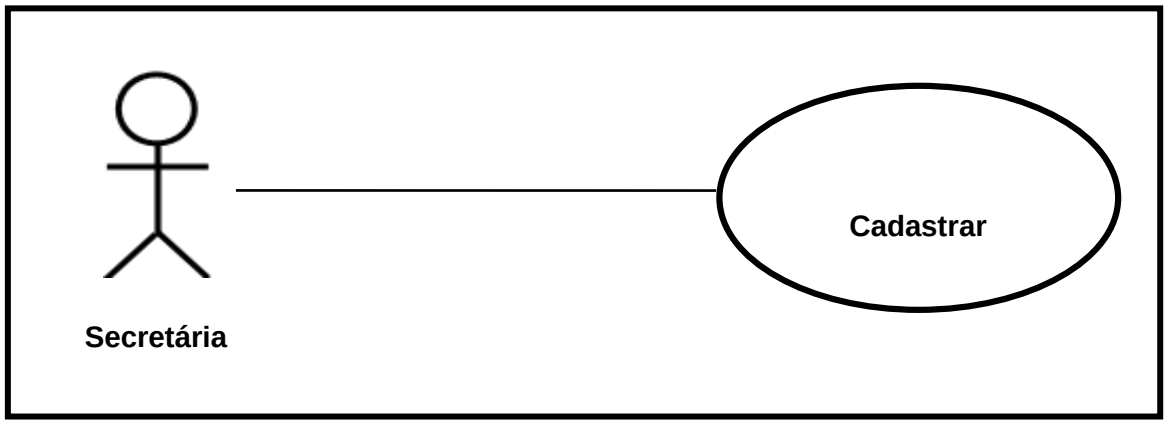


Figura 17 - Caso de Uso: Cadastrar Professor

Use Case Cadastrar Professor

Passos

1 - Ler os dados do Professor (Nome, Endereço, Cidade, Estado, Fone, RG, CPF, Data de Admissão, Observações)

Se Professor cadastrado => MSG05 "Professor Cadastrado com Sucesso" Exceção

Se campo não informado => MSG05 "Campo não informado"

Se Professor já Cadastrado => MSG05 "Professor já cadastrado"

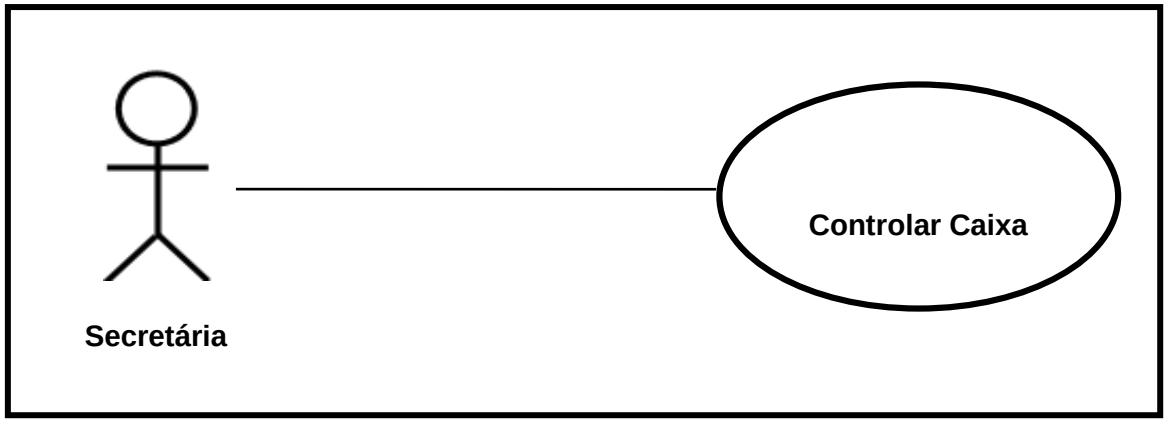


Figura 18 - Caso de Uso: Controlar Caixa

Use Case Controlar Caixa

Passos

1 - Ler todas as informações (Débito ou Crédito, Descrição da Movimentação, Data e Hora da Movimentação) de recebimentos e pagamentos da escola

Se movimentação controlada => MSG06 "Movimentação do Caixa feita com Sucesso"

Exceção

Se campo não informado => MSG06 "Campo não informado"

Se aluno não está em déb'ito => MSG06 "O Aluno já está quitado"

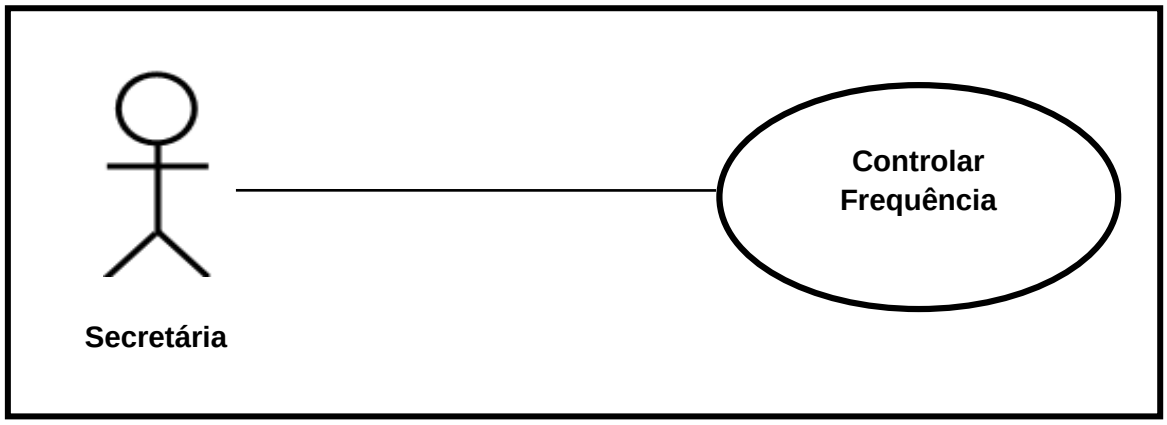


Figura 19 - Caso de Uso: Controlar Frequência

Use Case Controlar Freqüência

Passos

1 - Ler os Dados (Código da turma, Código do Aluno, Data da Aula e se está Presente)

Se presença controlada => MSG07 "Freqüência Controlada com Suc~sso" Exceção

Se campo não informado => MSG07 "Campo não Informado"

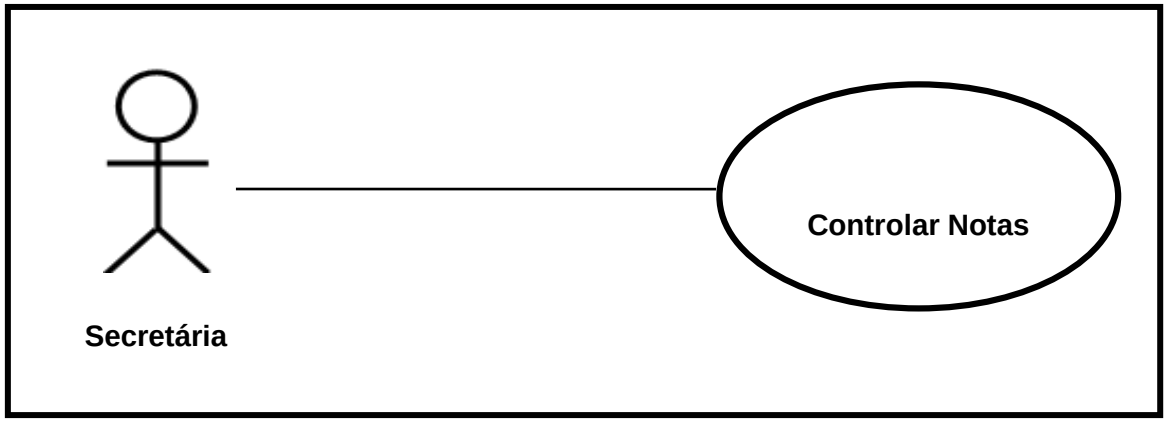


Figura 20 - Caso de Uso: Controlar Notas

Use Case Controlar Notas

Passos

1 - Ler os Dados (Código do Curso, Código do Aluno, Data da Nota)

Se Nota controlada => MSG08 "Nota Controlada com Sucesso" Exceção

Se campo não informado => MSG08 "Campo não Informado"

Se Nota com mesma data => MSG08 "Nota já Cadastrada. Falar com o Professor antes de Alterar"

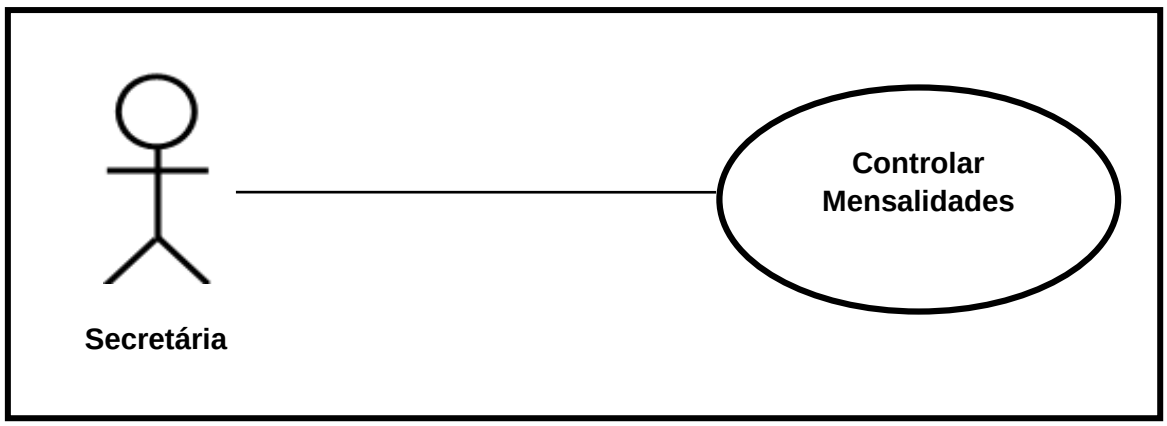


Figura 21 - Caso de Uso: Controlar Mensalidades

Use Case Controlar Mensalidades

Passos

1 - Ler os Dados (Data de Vencimento, Código do Aluno, Data de Pagamento, Valor, Situação)

Se Mensalidade controlada => MSG09 "Mensalidade Controlada com Sucesso" Exceção

Se campo não informado => MSG09 "Campo não Informado"

Se Mensalidade atrasada => MSG09 "Mensalidade Atrasada. O Valor está com juros"

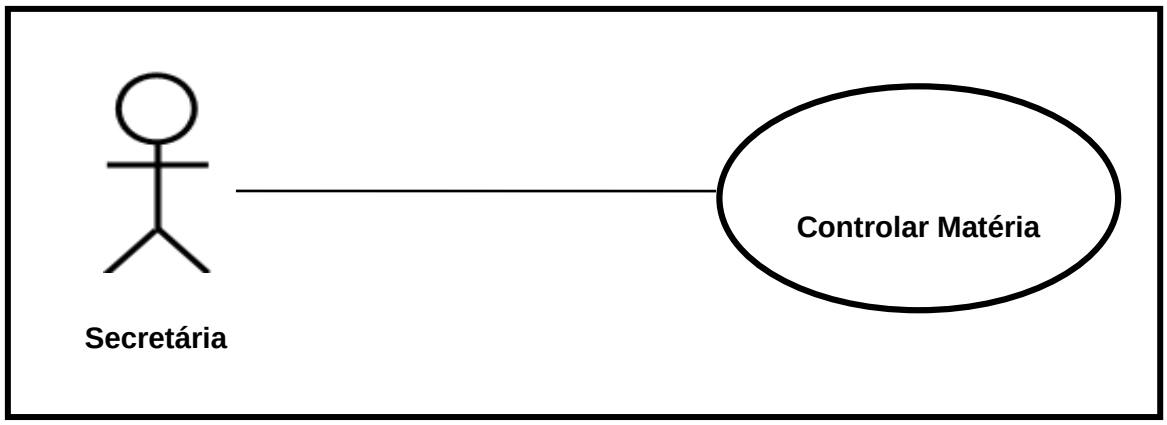


Figura 22 - Caso de Uso: Controlar Matéria

Use Case Controlar Matéria

Passos

1 - Ler os Dados (Código da turma, Data em que a matéria foi dada, Código do Professor, Descrição da Matéria)

Se Matéria Controlada => MSG1O "Matéria Controlada com Sucesso" Exceção

Se campo não informado => MSG1O "Campo não informado"

Se código da turma e data da matéria já cadastrados => MSG1O "Matéria controlada. Falar com o Professor antes de Alterar"

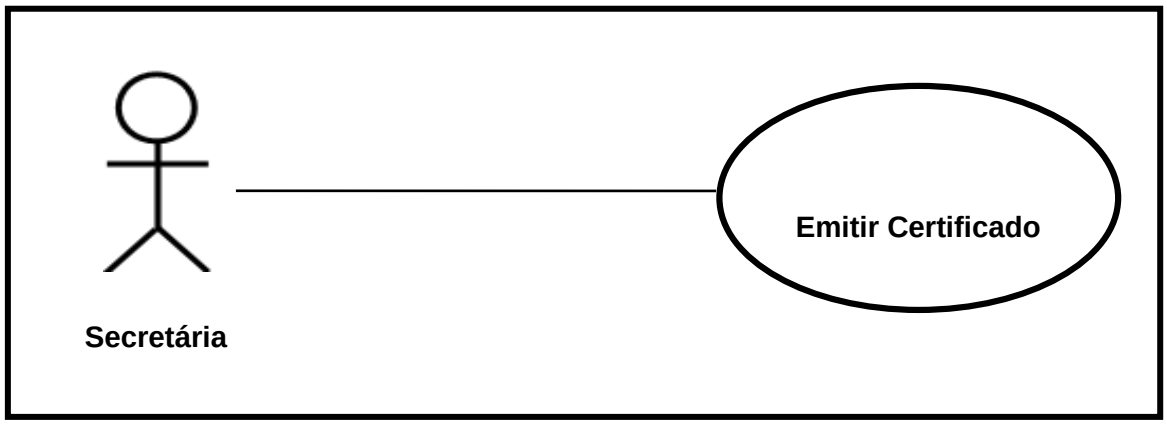


Figura 23 - Caso de Uso: Emitir Certificado

Use Case Emitir Certificado

Passos

1 - Localizar o Aluno no Arquivo Morto

2 - Imprimir o Certificado com os dados do Arquivo

Se certificado impresso => MSG11 "Certificado Impresso com Sucesso" Exceção

Se certificado já impresso antes => MSG11 "O Certificado já foi impresso antes"

Se a impressora não está pronta => MSG11 "Impressão do Certificado Cancelada. Acerte a Impressora"

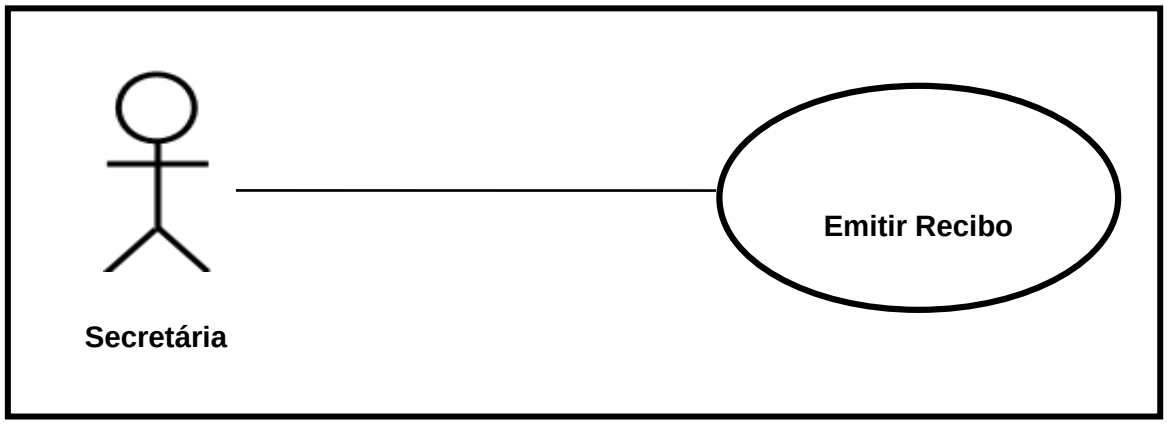


Figura 24 - Caso de Uso: Emitir Recibo

Use Case Emitir Recibo

Passos

1 - Localizar a Movimentação do *Caixa*

2 - Imprimir o Recibo com os dados da Movimentação do *Caixa*

Se Recibo impresso => MSGl2 "Recibo Impresso com Sucesso" Exceção

Se Recibo já impresso antes => MSG12 "O Recibo já foi impresso antes"

Se a impressora não está pronta => MSG12 "Impressão do Recibo Cancelada. Acerte a Impressora"

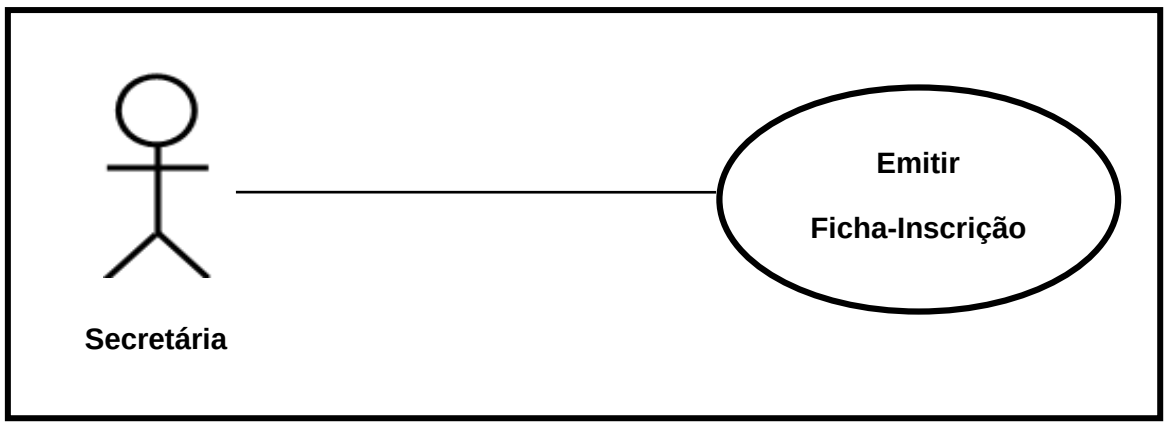


Figura 25 - Caso de Uso: Emitir Ficha-Inscrição

Use Case Emitir Ficha-Inscrição

Passos

1 - Localizar os dados do Aluno no Arquivo de Alunos

2 - Imprimir a ficha de inscrição com os dados do Aluno

Se Ficha de inscrição impressa => MSG13 "Ficha de Inscrição Impressa com sucesso"

Exceção

Se dados não Localizado => MSG13 "Aluno não Cadastrado"

Se a impressora não está pronta => MSG13 "Impressão da Ficha de Inscrição Cancelada. Acerte a Impressora"

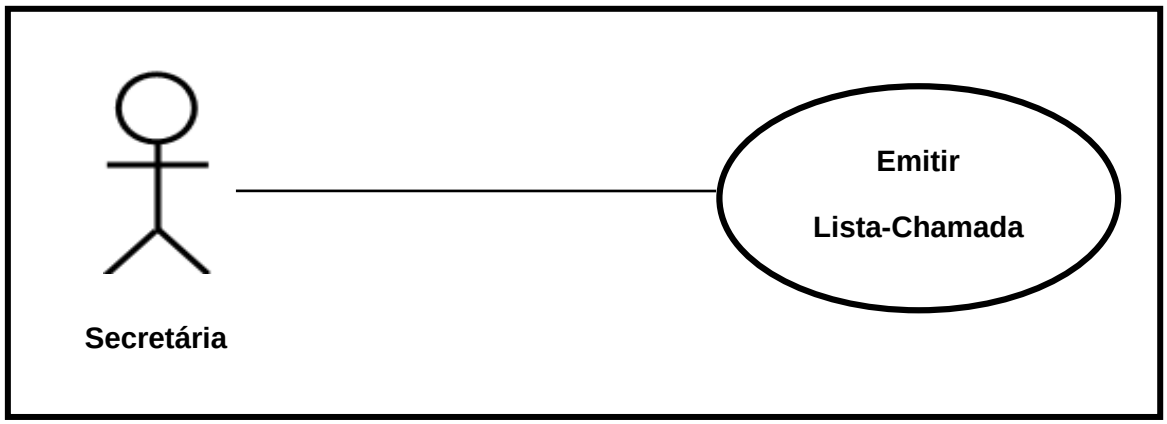


Figura 26 - Caso de Uso: Emitir Lista-Chamada

Use Case Emitir Lista-Chamada

Passos

1 - Localizar todos os Alunos da Turma

2 - Imprimir a Lista com todos os nomes de Alunos que freqüentam a turma

Se Lista de Chamada impressa => MSG14 "Lista de Chamada Impressa com sucesso"

Exceção

Se dados não Localizado => MSG14 "Turma não Cadastrado"

Se a impressora não está pronta => MSG14 "Impressão da Lista de Chamada Cancelada. Acerte a Impressora"

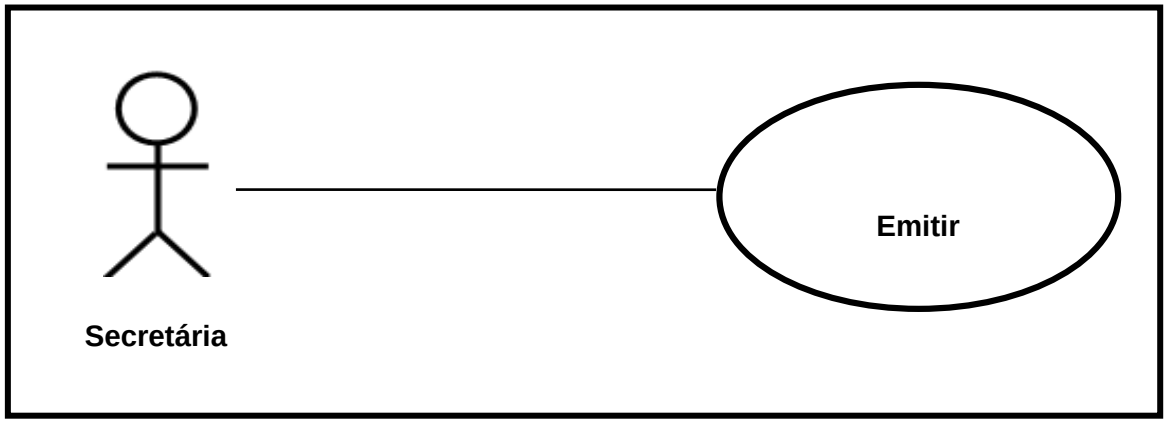


Figura 27 - Caso de Uso: Emitir Livro-Ponto

Use Case Emitir Livro-Ponto

Passos

1 - Ler o Código do Professor

2 - Calcular as Aulas que serão dadas pelo Professor

3 - Imprimir Livro Ponto

Se Livro-Ponto impresso => MSG16 "Livro-Ponto impresso com sucesso" Exceção

Se código do Professor não informado => MSG15 "Código não informado"

Se a impressora não está pronta => MSG15 "Impressão Cancelada. Ajuste a

impressora"

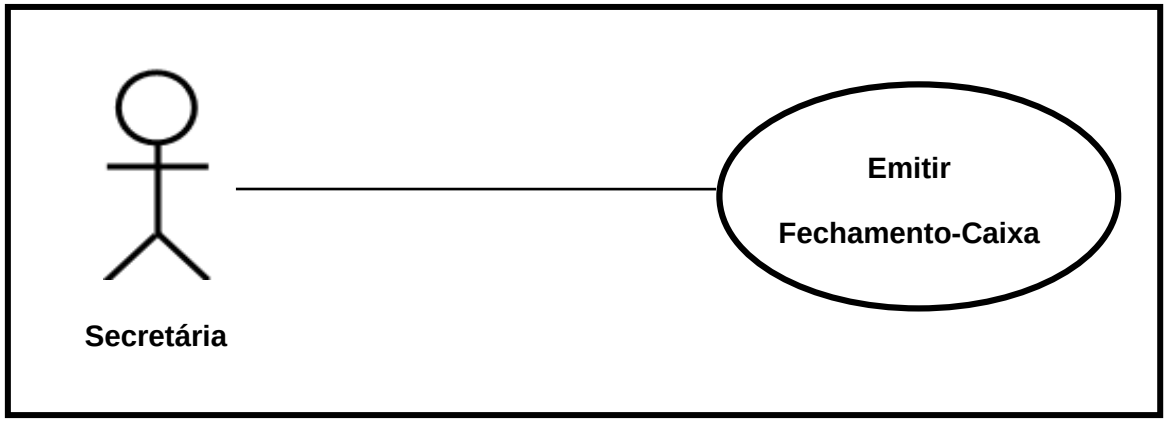


Figura 28 - Caso de Uso: Emitir Fechamento-Caixa

Use Case Emitir Fechamento-Caixa

Passos

1 - Ler a Data do Fechamento

2 - Calcular o Fechamento do Caixa

3 - Imprimir o Fechamento do Caixa

Se Fechamento do Caixa impresso => MSG16 "Fechamento de Caixa feito com sucesso"

Exceção

Se Data inválida => MSG16 "Data inválida"

Se Data do Fechamento for maior que a do Dia => MSG16 "Data inválida" Se a impressora não está pronta => MSG16 "Impressão Cancelada. Ajuste a impressora"

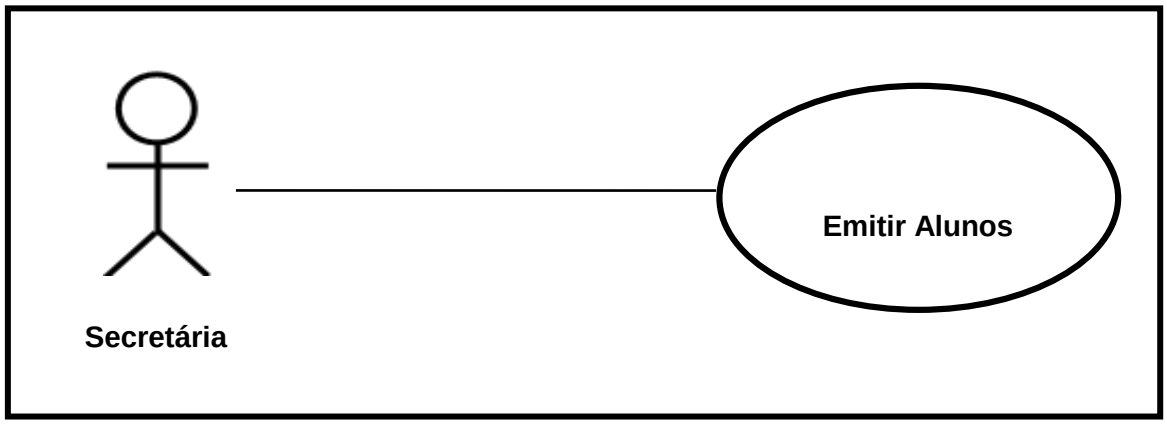


Figura 29 - Caso de Uso: Emitir Alunos do Curso

Use Case Emitir Alunos do Curso

Passos

1 - Ler o código do Curso

2 - Filtrar os Alunos do Curso

3 - Imprimir os Alunos Filtrados

Se Alunos Filtrados impressos => MSG17 "Listagem impressa com sucesso" Exceção

Se código do Curso inválido => MSG17 "Código de Curso Inválido"

Se não existem alunos nesse curso => MSG17 "Não existe Aluno nesse Curso" Se a impressora não está pronta => MSG17 "Impressão Cancelada. Ajuste a Impressora"

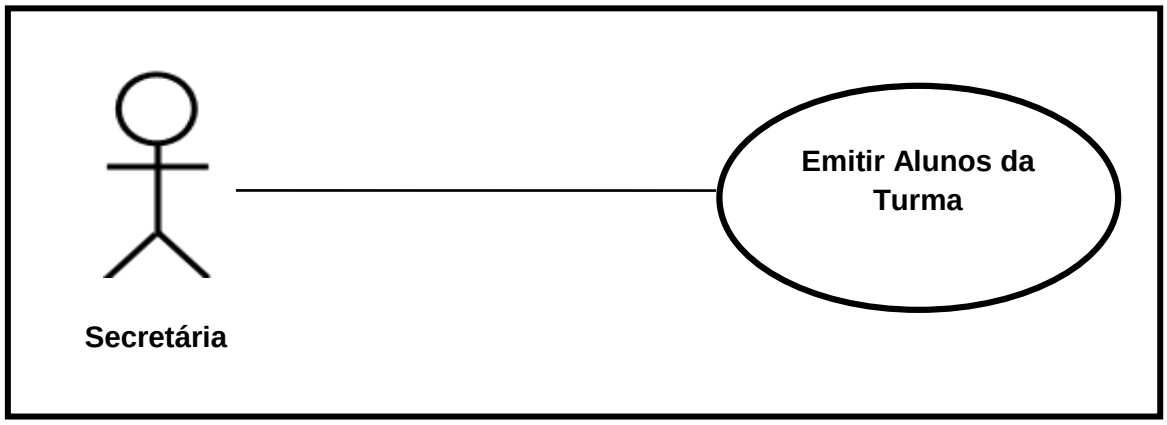


Figura 30 - Caso de Uso: Emitir Alunos da Turma

Use Case Emitir Alunos da Turma

Passos

1 - Ler o CÓdigo da Turma

2 - Filtrar os Alunos da Turma

3 - Imprimir os Alunos Filtrados

Se Alunos Filtrados impressos => MSG18 "Listagem impressa com sucesso" Exceção

Se código da Turma inválido => MSG18 "Código de Turma Inválido"

Se não existem alunos nessa Turma => MSG18 "Não existe Aluno nessa Turma" Se a impressora não está pronta => MSG18 "Impressão Cancelada. Ajuste a Impressora"

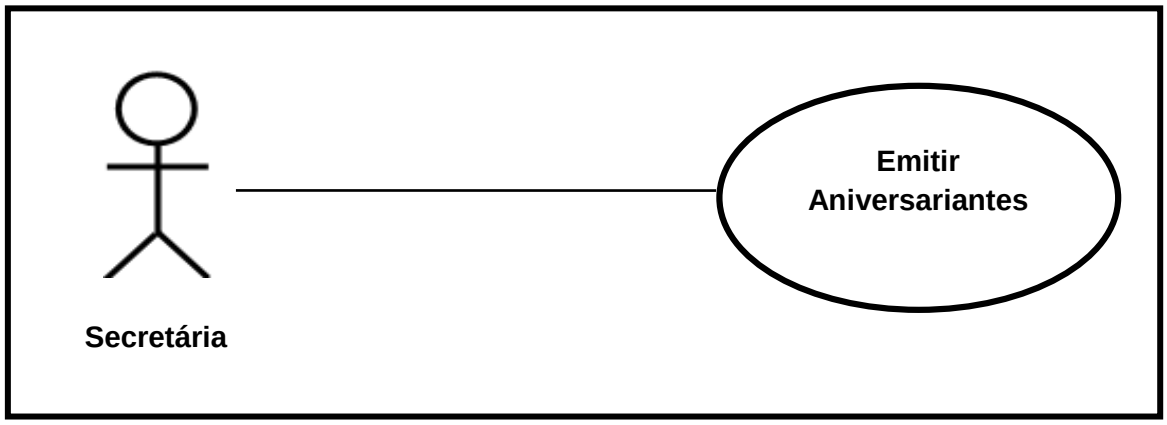


Figura 31 - Caso de Uso: Emitir Aniversariantes

Use Case Emitir Aniversariantes

Passos

1 Filtrar Todos os Alunos que fazem aniversário no mês

2 - Imprimir Alunos Filtrados

Se Alunos Filtrados Impressos => MSG19 "Lista impressa com Sucesso" Exceção

Se a impressora não está pronta => MSG19 "Impressão Cancelada. Ajuste a impressora"

Se não filtrou nenhum Aluno => MSG19 "Não Existe Aluno que faz aniversário neste mês"

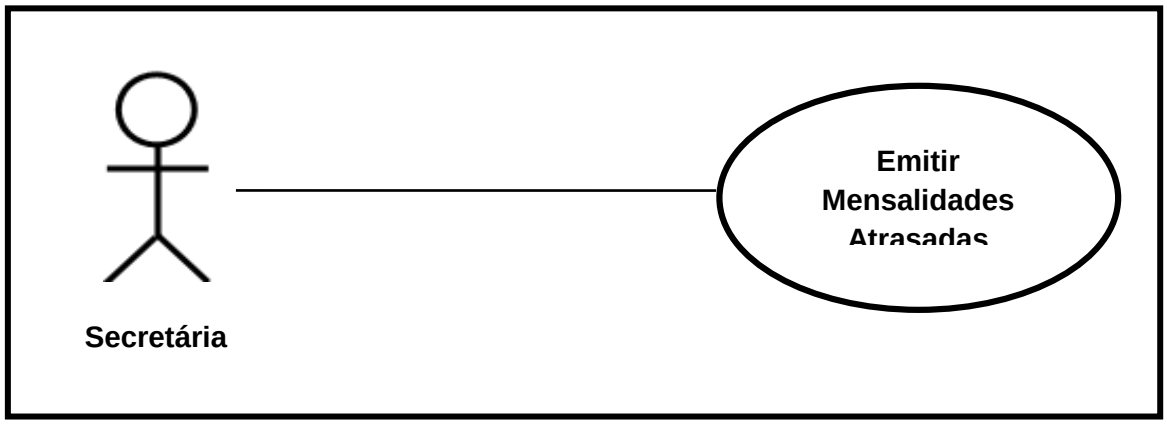


Figura 32 - Caso de Uso: Emitir Mensalidades Atrasadas

Use Case Emitir Mensalidades Atrasadas

Passos

1 - Filtrar Todos os Alunos que tem mensalidades Atrasadas

2 - Imprimir Alunos Filtrados

Se Alunos Filtrados Impressos => MSG20 "Lista impressa com Sucesso" Exceção

Se a impressora não está pronta => MSG20 "Impressão Cancelada. Ajuste a impressora"

Se não filtrou nenhum Aluno => MSG20 "Não Existe Aluno com mensalidade atrasada"

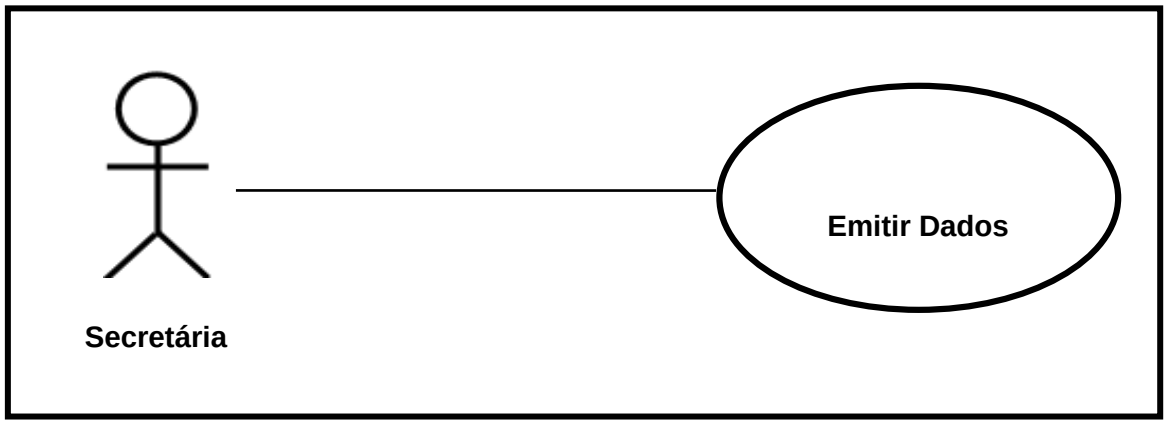


Figura 33 - Caso de Uso: Emitir Dados Professor

Use Case Emitir Dados Professor

Passos

1 - Ler o Código do Professor

2 - Imprimir todos os Dados do Professor (Cadastrais, .Curso e Turma)

Se Dados impressos => MSG21 "Dados impressos com Sucesso" Exceção

Se código do Professor errado => MSG21 "Professor não Cadastrado"

Se a Impressora não está pronta => MSG21 "Impressão Cancelada. Ajuste a impressora"

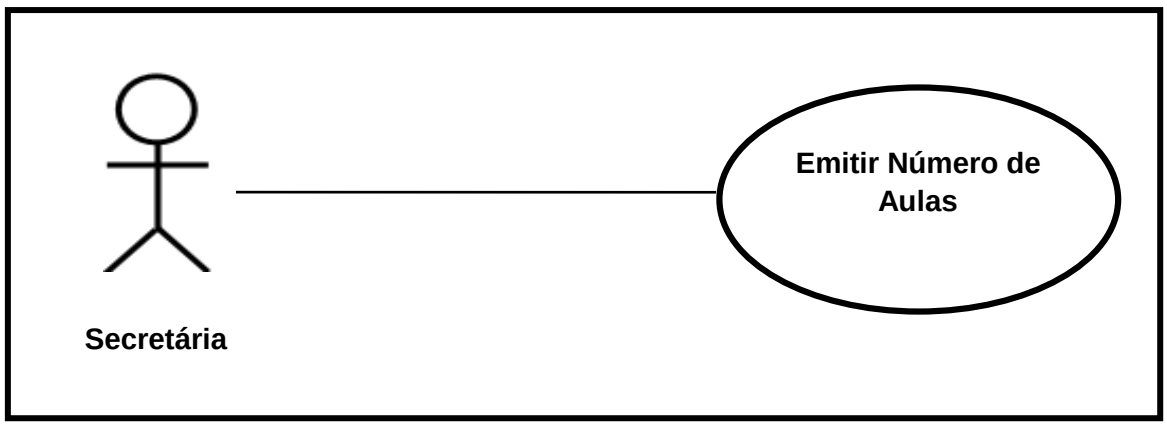


Figura 34 - Caso de Uso: Emitir Número de Aulas

Use Case Emitir Número de Aulas

Passos

1 - Ler o código do Professor

2 - Ler o código da Turma

3 - Calcular o quantidade de aulas dadas pelo professor para a turma 4 - Imprimir Número de Aulas

Se Número de Aulas impresso => MSG22 "Número de Aulas impresso com Sucesso"

Exceção

Se código do Professor ou Turma inválido => MSG22 "Código inválido"

Se a impressora não está pronta => MSG22 "Impressão Cancelada. Ajuste a impressora"

Se o Professor não deu Aula => MSG22 "O professor não deu aula neste Período"

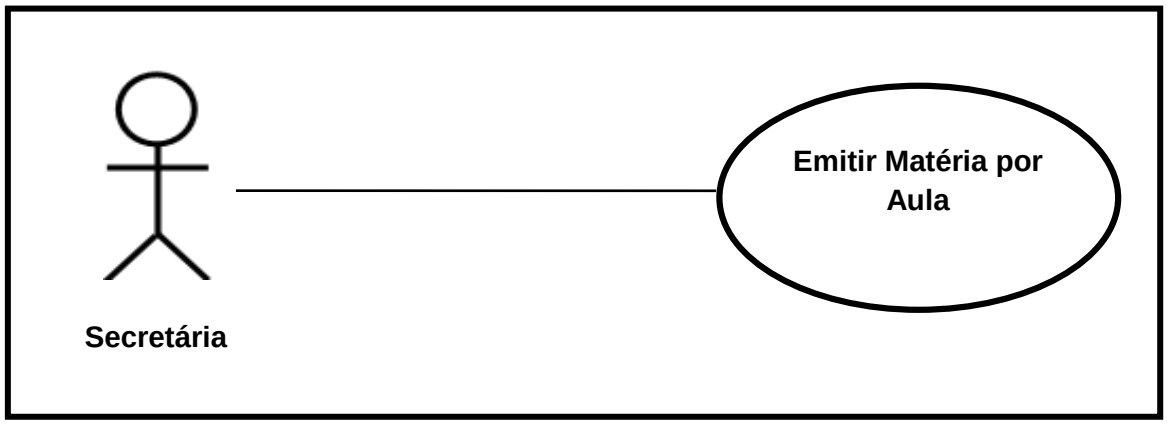


Figura 35 - Caso de Uso: Emitir Matéria por Aula

Use Case Emitir Matéria por Aula

Passos

1 - Ler o código do Professor

2 - Filtrar todas as matérias dadas pelo professor em aula 3 - Imprimir os dados Filtrados

Se Dados Filtrados impressos => MSG23 "Matéria Dada impressa" Exceção

Se a impressora não está pronta => MSG23 "Impressão Cancelada. Ajuste a impressora"

Se código do Professor inválido => MSG23 "Professor não Cadastrado"

Se não tiver matéria => MSG23 "Não existe matéria dada registrada pelo

Professor"

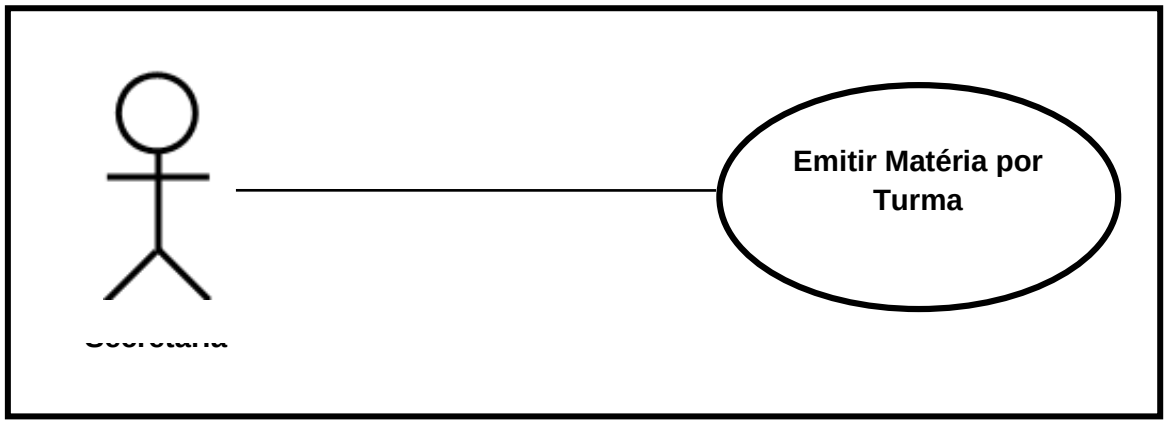


Figura 36 - Caso de Uso: Emitir Matéria por Turma

Use Case Emitir Matéria por Turma

Passos

1 - Ler o código da Turma

2 - Filtrar todas as matérias dadas pelo professor em aula para a Turma 3 - Imprimir os dados Filtrados

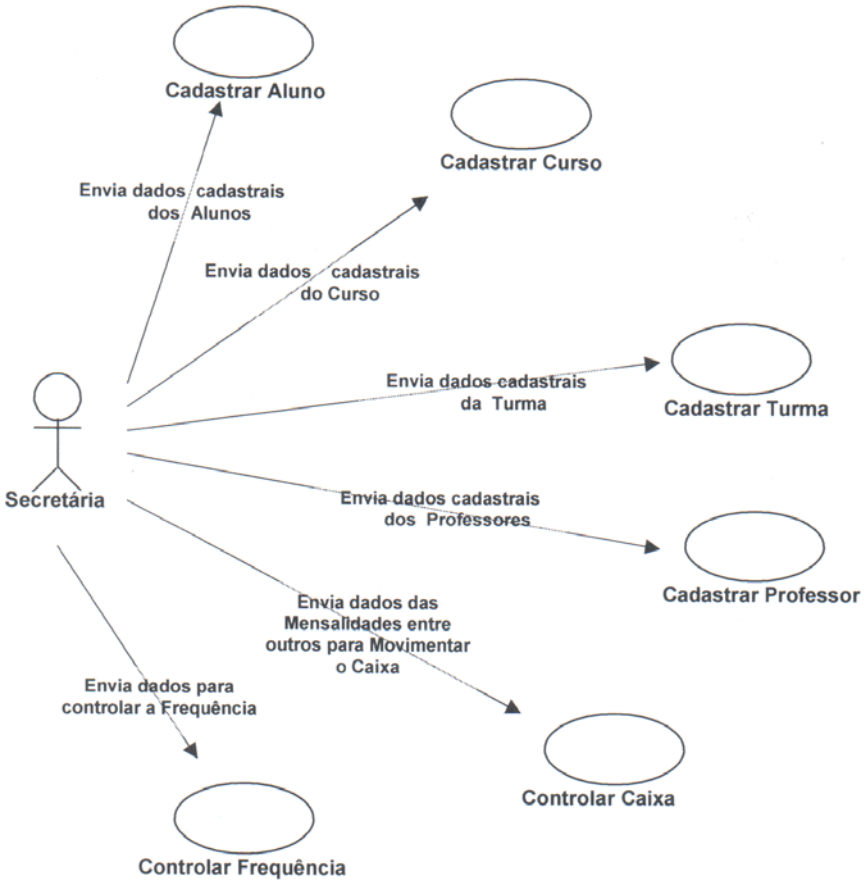
Se Dados Filtrados impressos => MSG24 "Matéria Dada impressa" Exceção

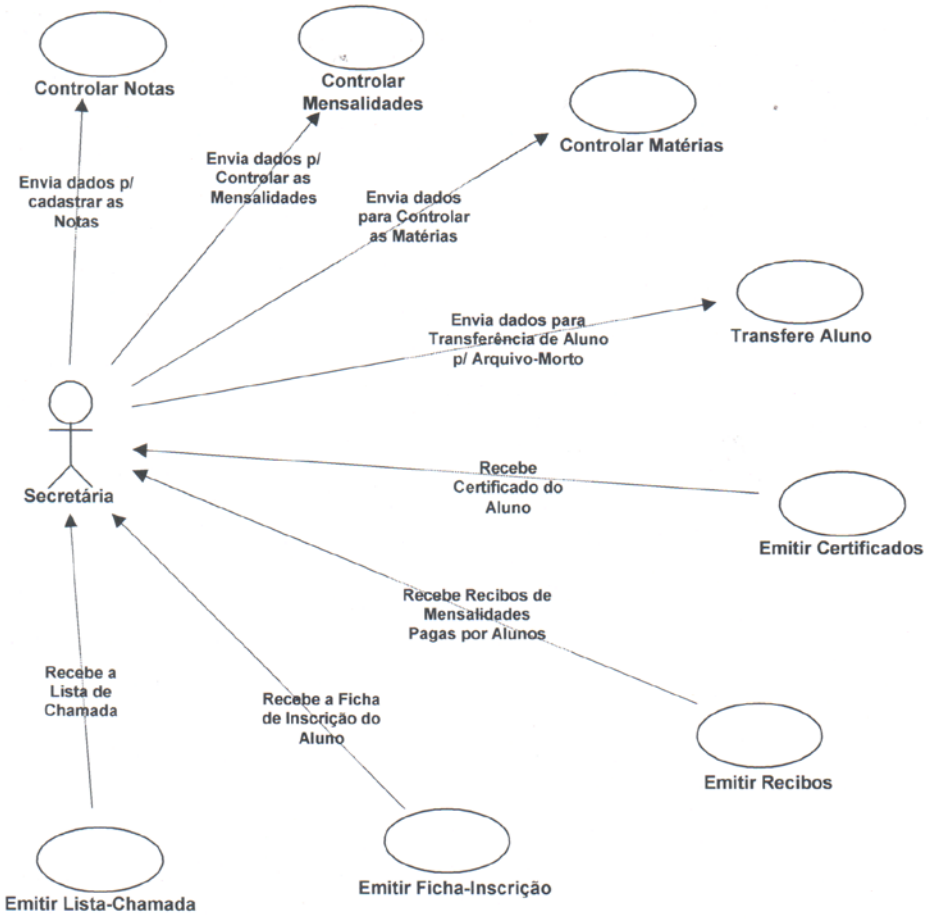
Se a impressora nlo está pronta => MSG24 "Impressão Cancelada. Ajuste a impressora"

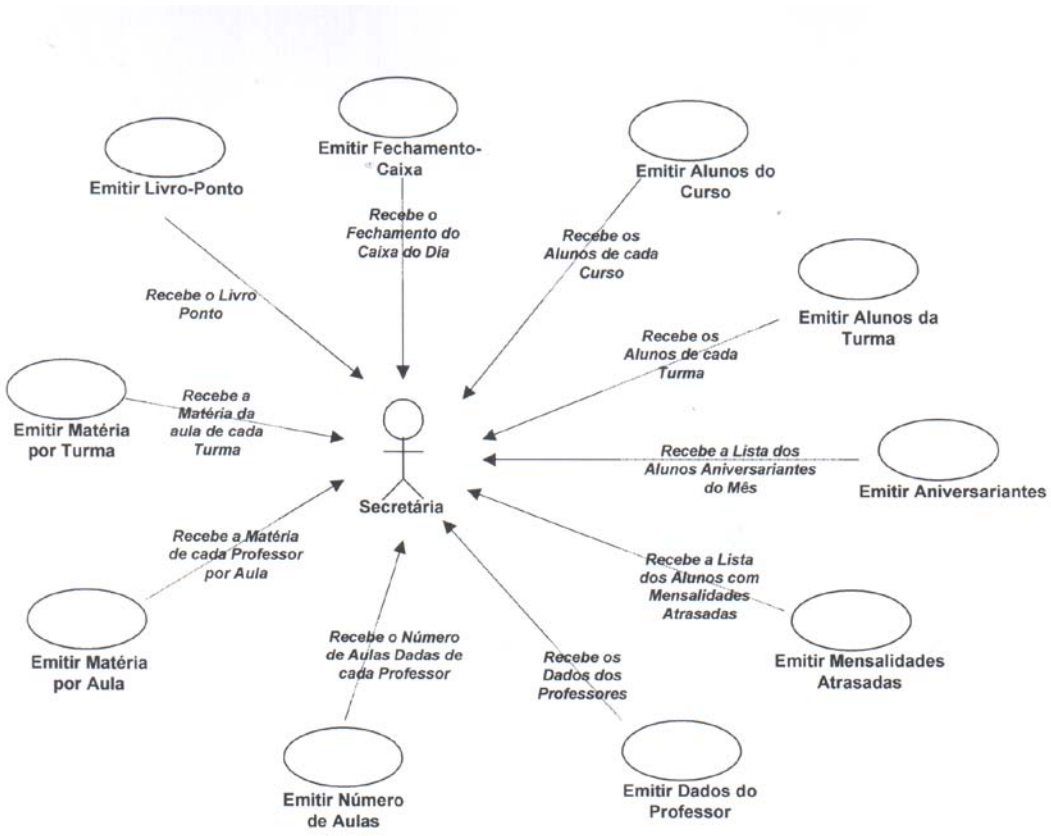
Se código da Turma inválido => MSG24 "Turma não Cadastrado"

Se não tiver matéria => MSG24 "Não existe matéria dada registrada para Turma"

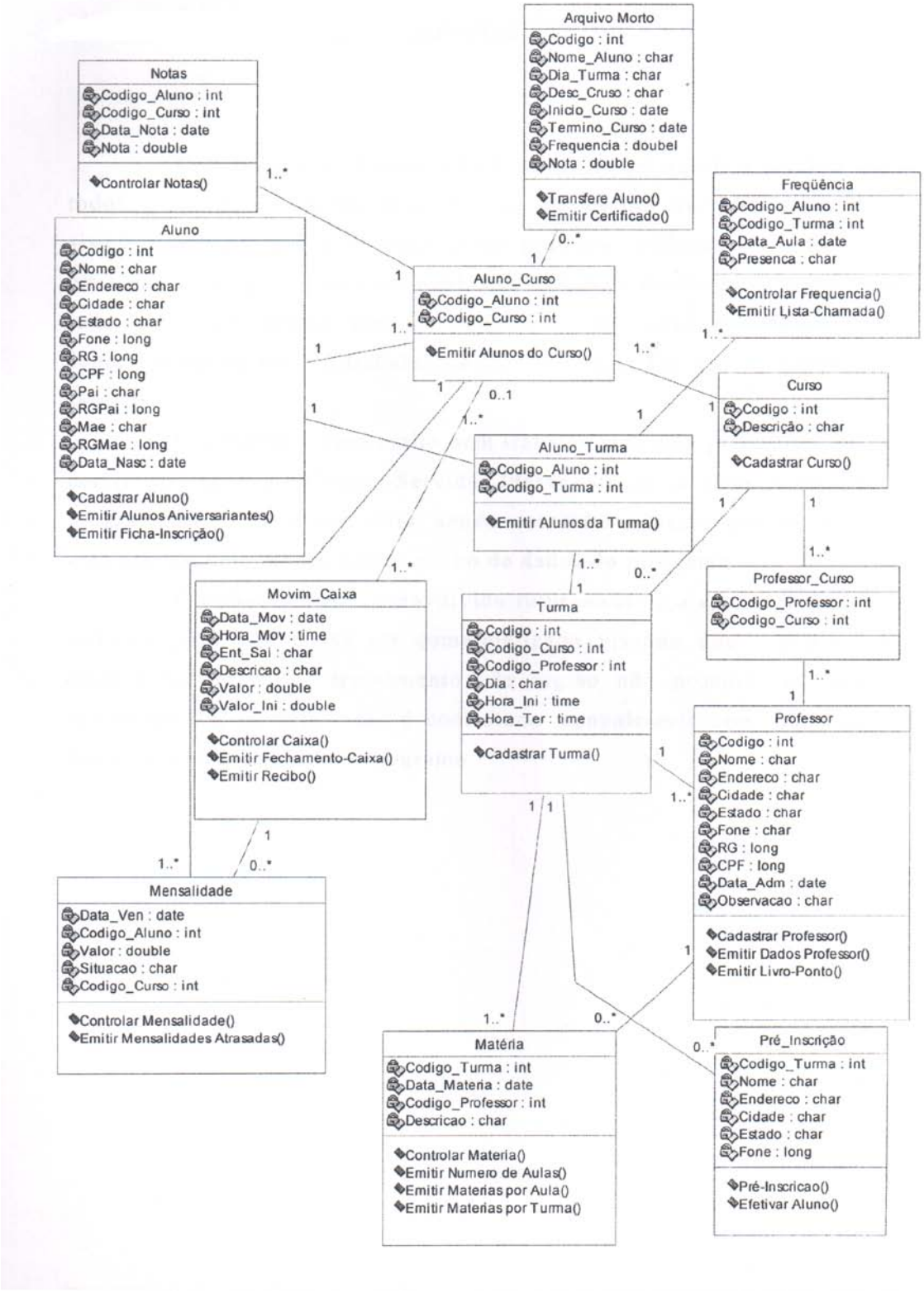
4.3 - Diagramas de USE CASE




Controlar Notas
Controlar Mensalidades
Controlar Matérias
Envia dados p/ cadastrar as Notas
Envia dados p/ Controlar as Mensalidades
Envia dados para Controlar as Matérias
Envia dados para Transferência de Aluno p/ Arquivo-Morto
Transfere Aluno
Secretária
Recebe Certificado do Aluno
Emitir Certificados
Recebe Recibos de Mensalidades Pagas por Alunos
Recebe a Lista de Chamada
Recebe a Ficha de Inscrição do Aluno
Emitir Recibos
Emitir Lista-Chamada
Emitir Ficha-Inscrição


Emitir Livro-Ponto
Emitir Fechamento-Caixa
Emitir Alunos do Curso
Recebe o Livro Ponto
Recebe o Fechamento do Caixa do Dia
Recebe os Alunos de cada Curso
Emitir Alunos da Turma
Recebe os Alunos de cada Turma
Emitir Matéria por Turma
Recebe a Matéria da aula de cada Turma
Recebe a Lista dos Alunos Aniversariantes do Mês
Emitir Aniversariantes
Secretária
Recebe a Matéria de cada Professor por Aula
Recebe a Lista dos Alunos com Mensalidades Atrasadas
Emitir Matéria por Aula
Recebe o Número de Aulas Dadas de cada Professor
Recebe os Dados dos Professores
Emitir Mensalidades Atrasadas
Emitir Número de Aulas
Emitir Dados do Professor

4.4 – Diagrama de Classes



4.5 – Diagramas de Seqüência

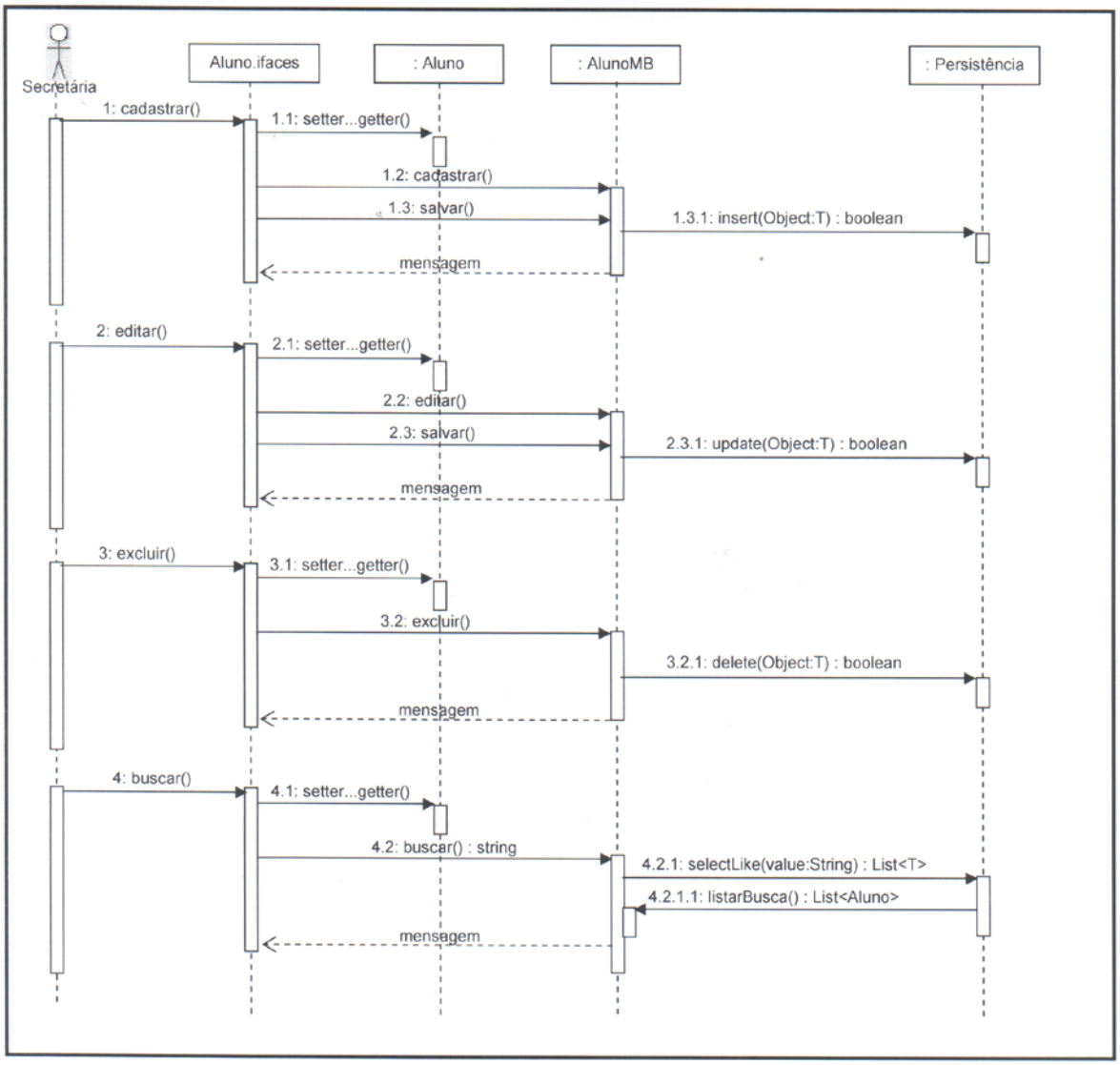


Diagrama de Sequência: Manter Aluno

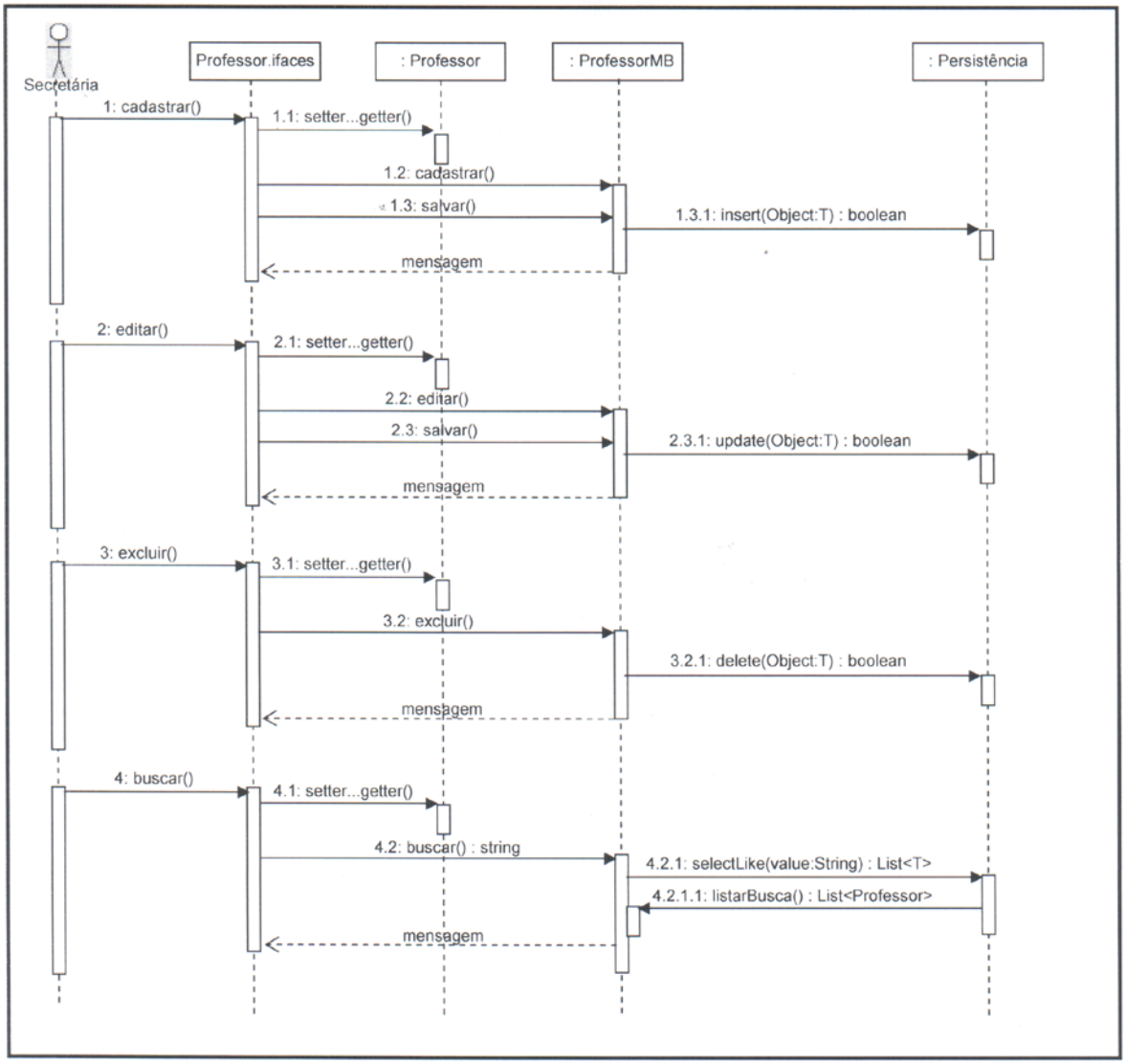


Diagrama de Sequência: Manter Professor

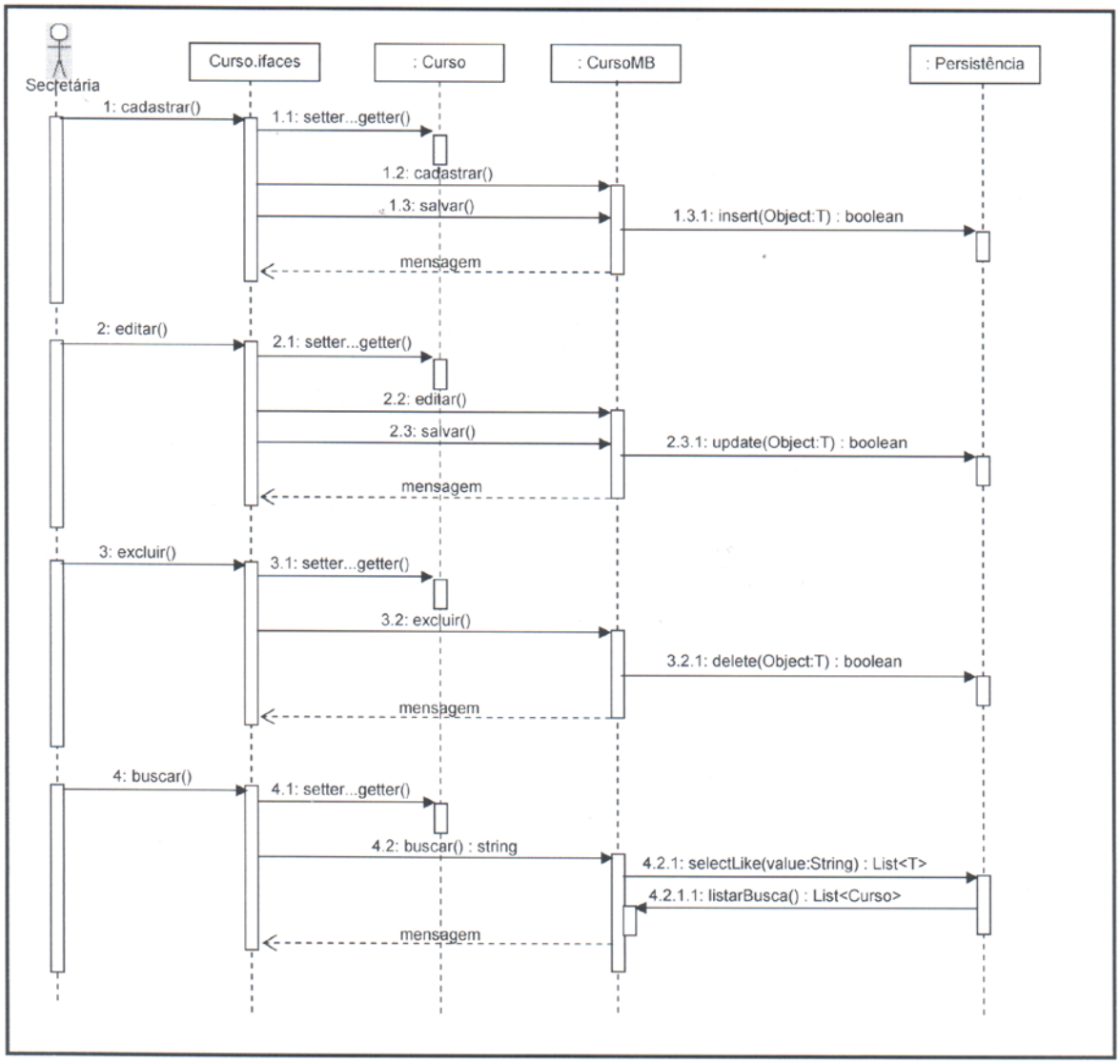


Diagrama de Sequência: Manter Curso

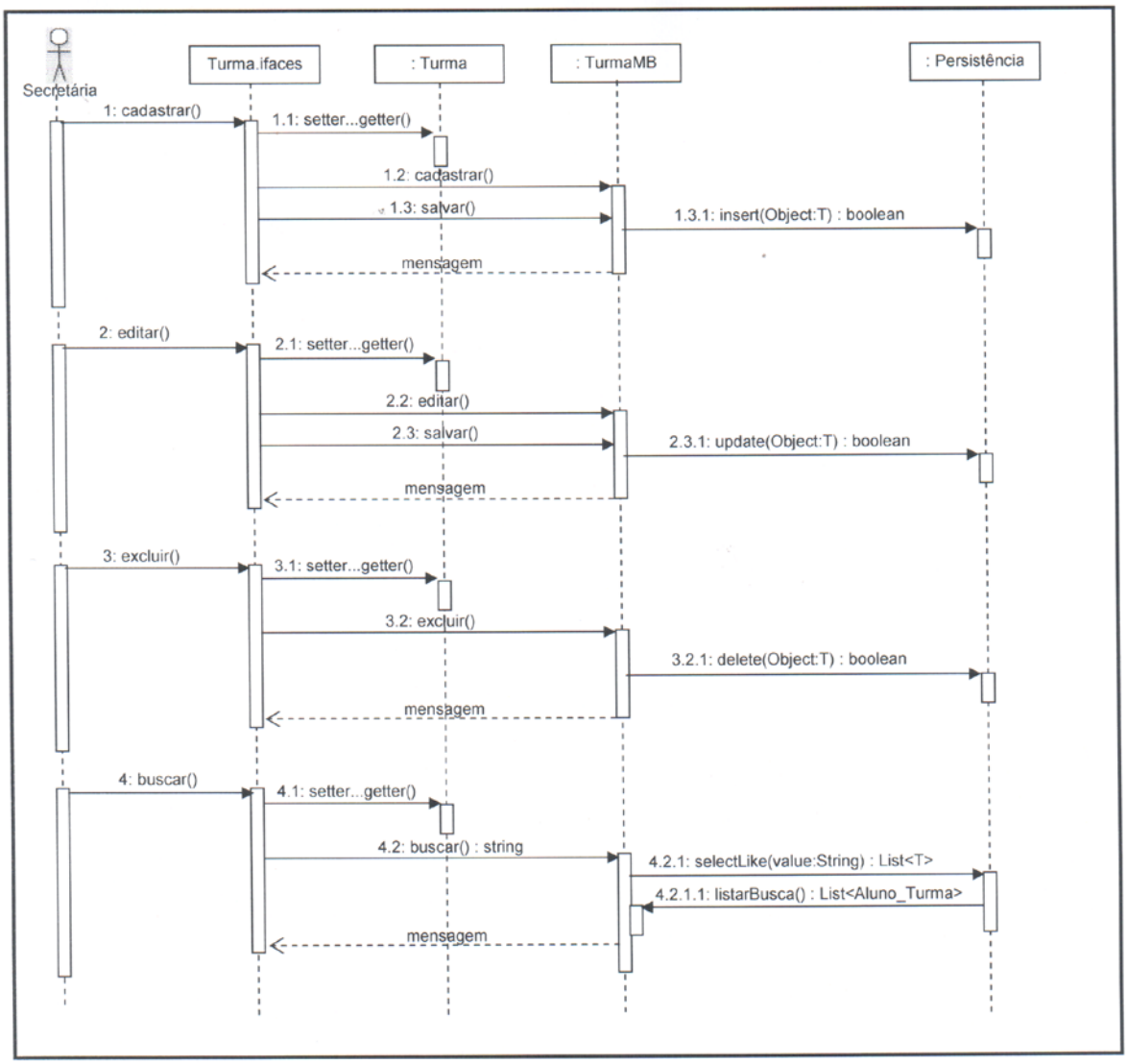


Diagrama de Sequência: Manter Turma

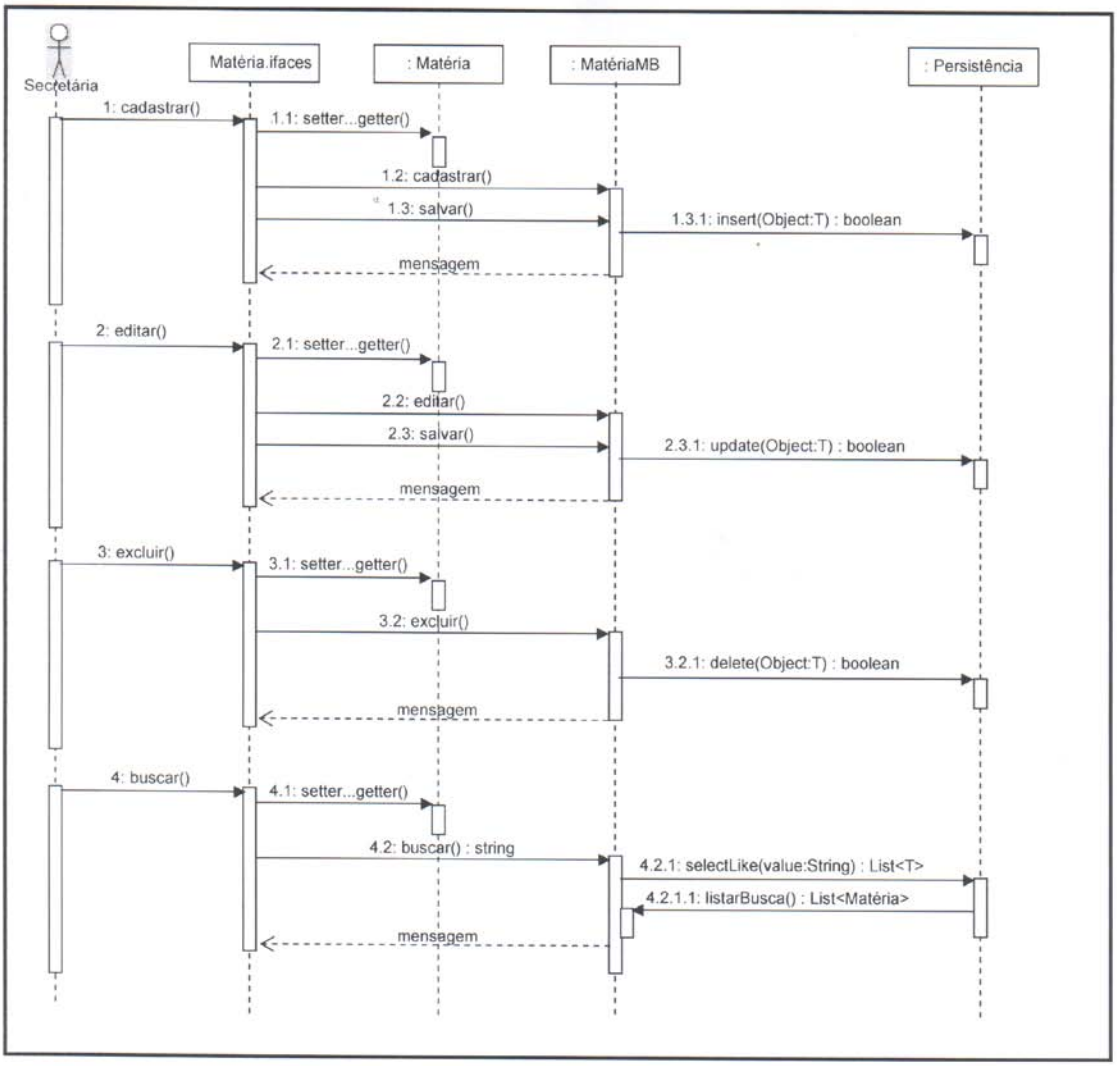


Diagrama de Sequência: Manter Matéria

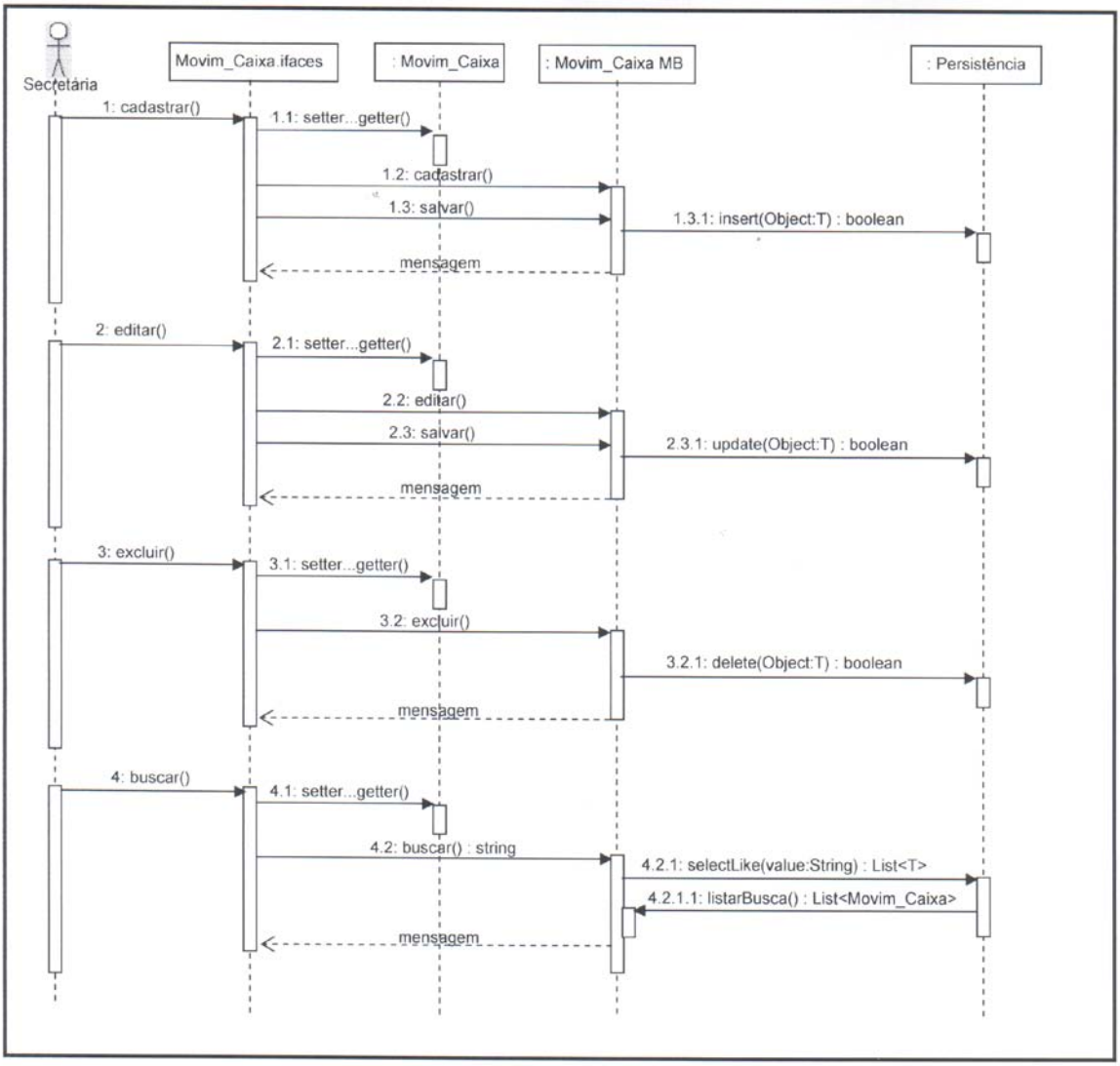


Diagrama de Sequência: Manter Movimentação de Caixa

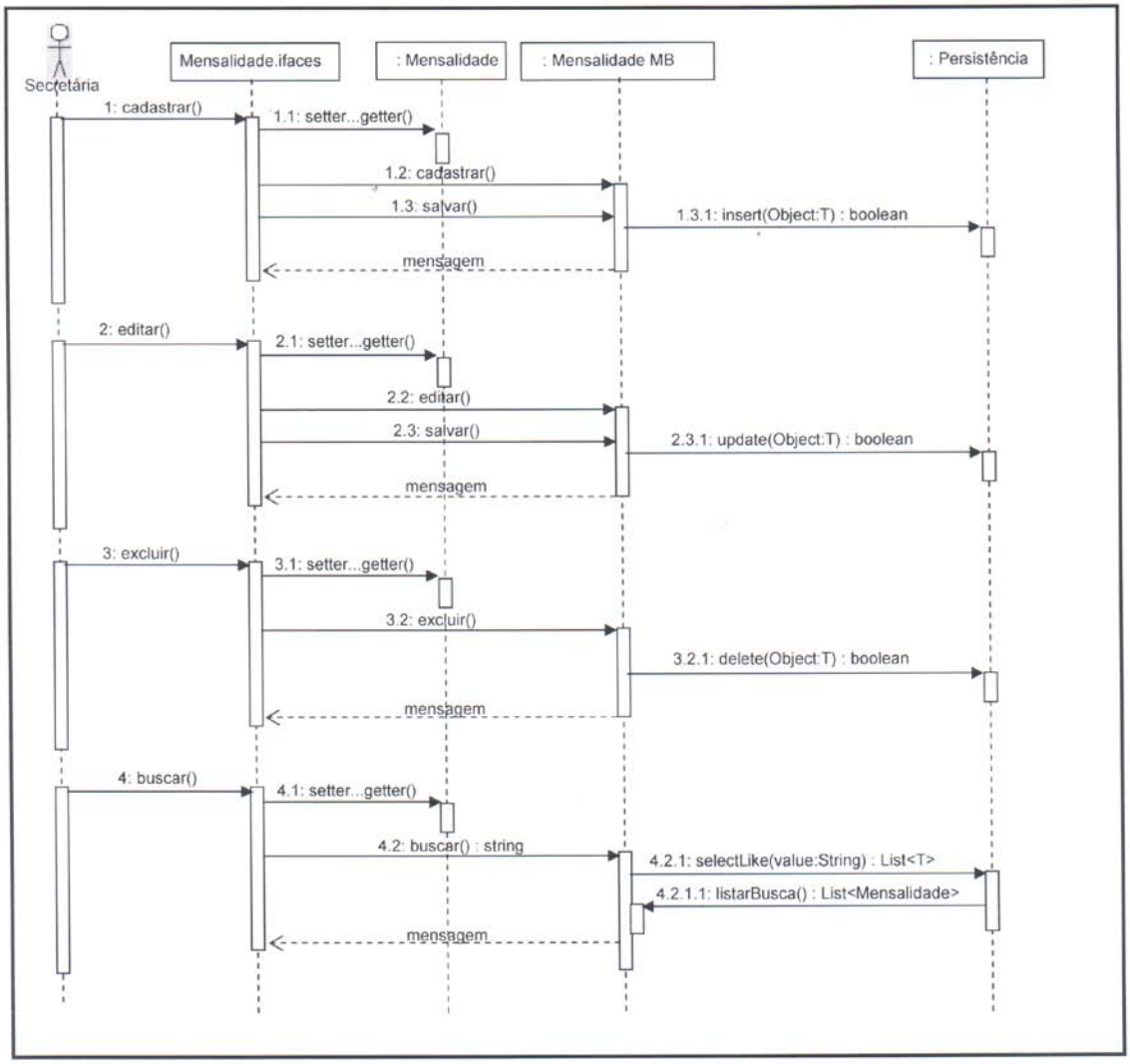


Diagrama de Sequência: Manter Mensalidades

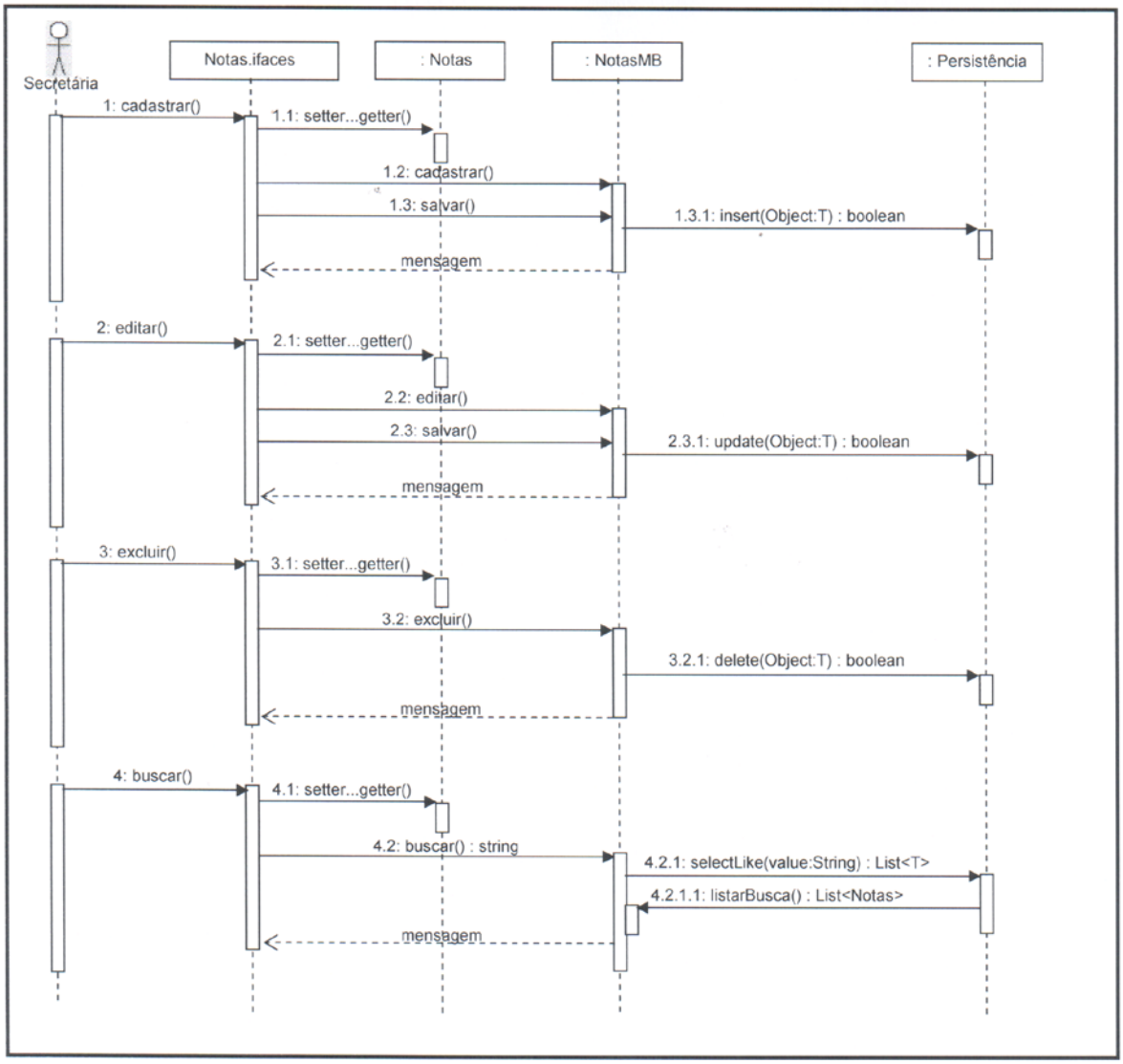


Diagrama de Sequência: Manter Notas

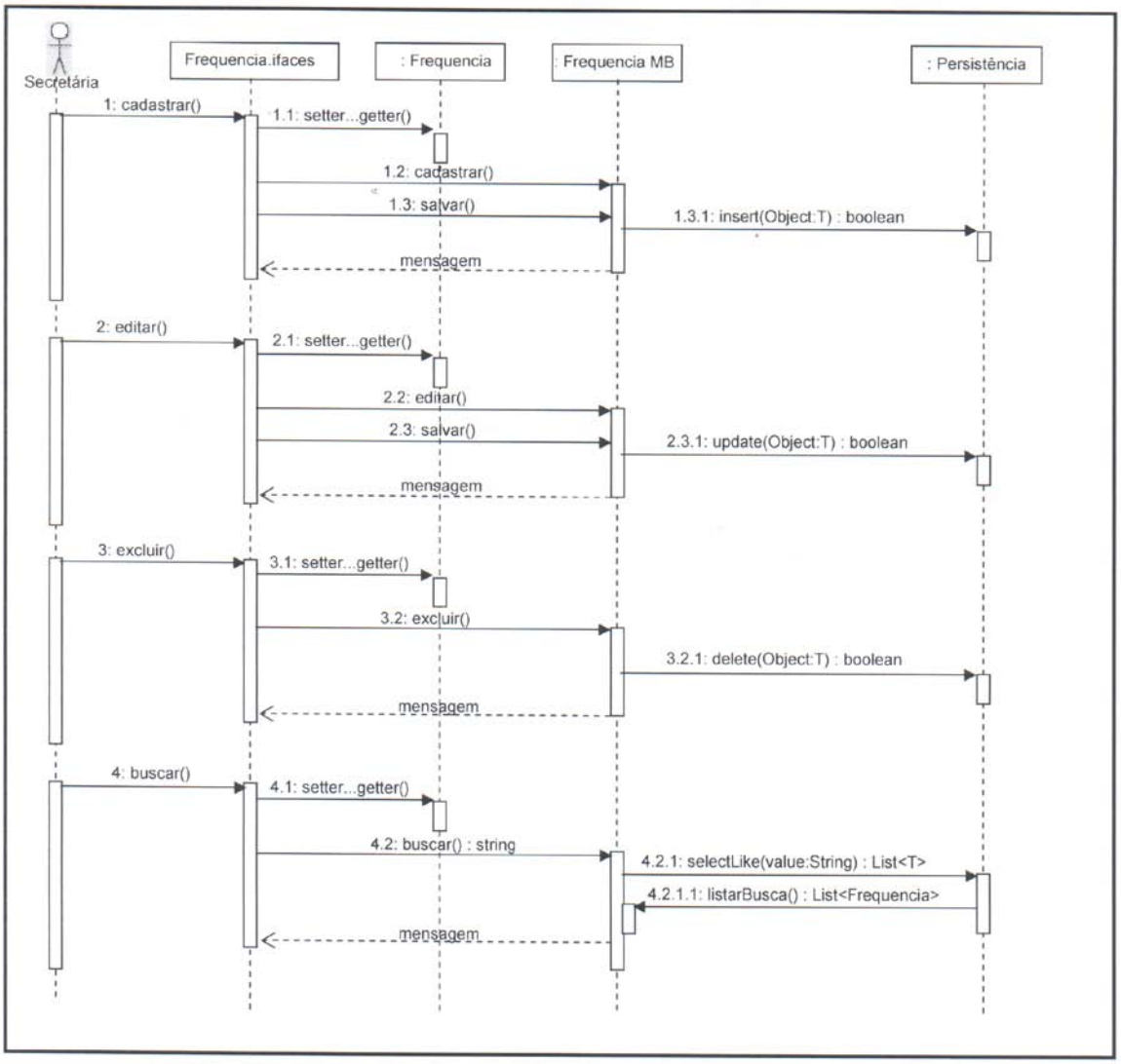


Diagrama de Sequência: Manter Frequência

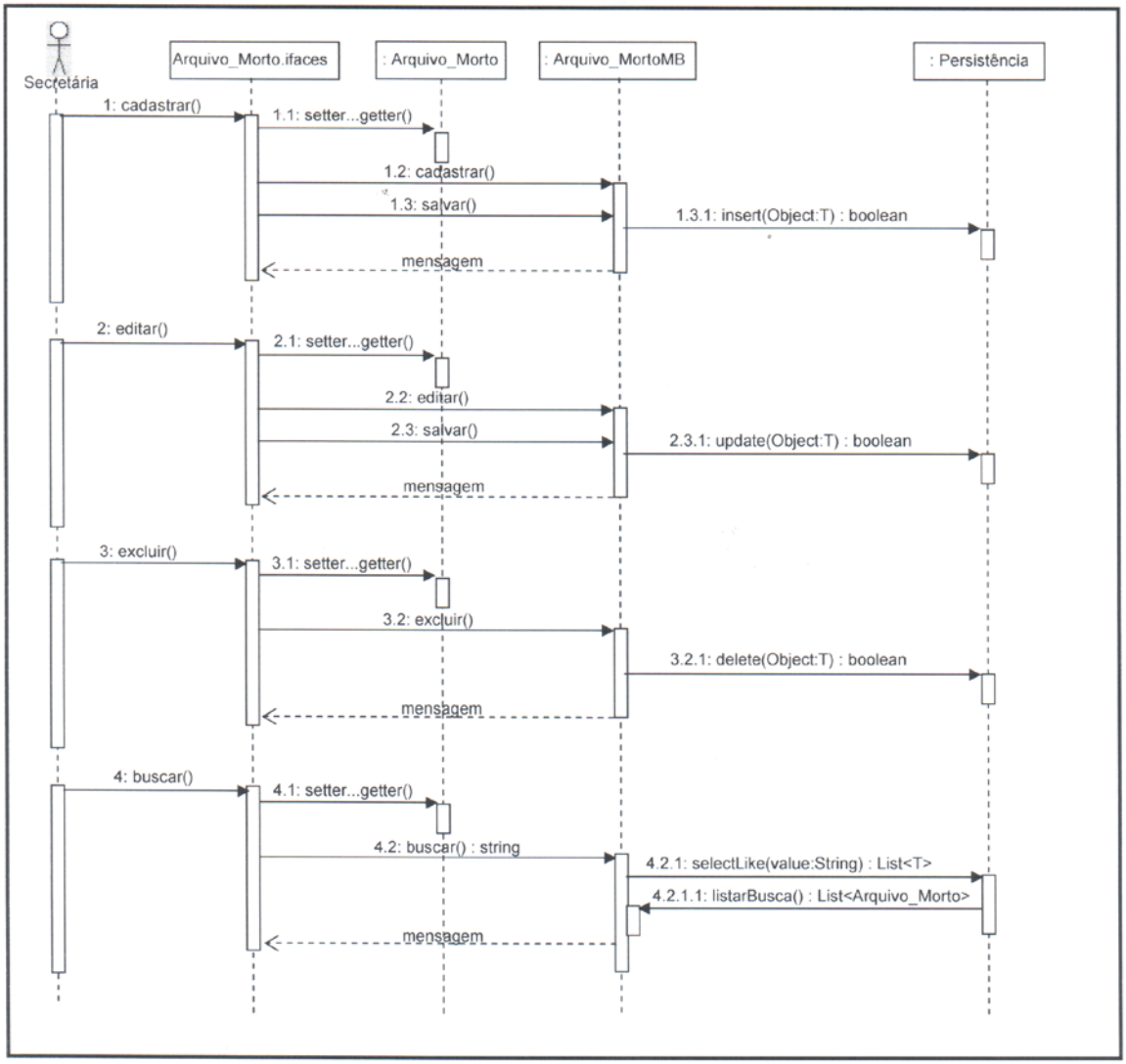


Diagrama de Sequência: Manter Arquivo Morto

4.6 - Diagrama WBS

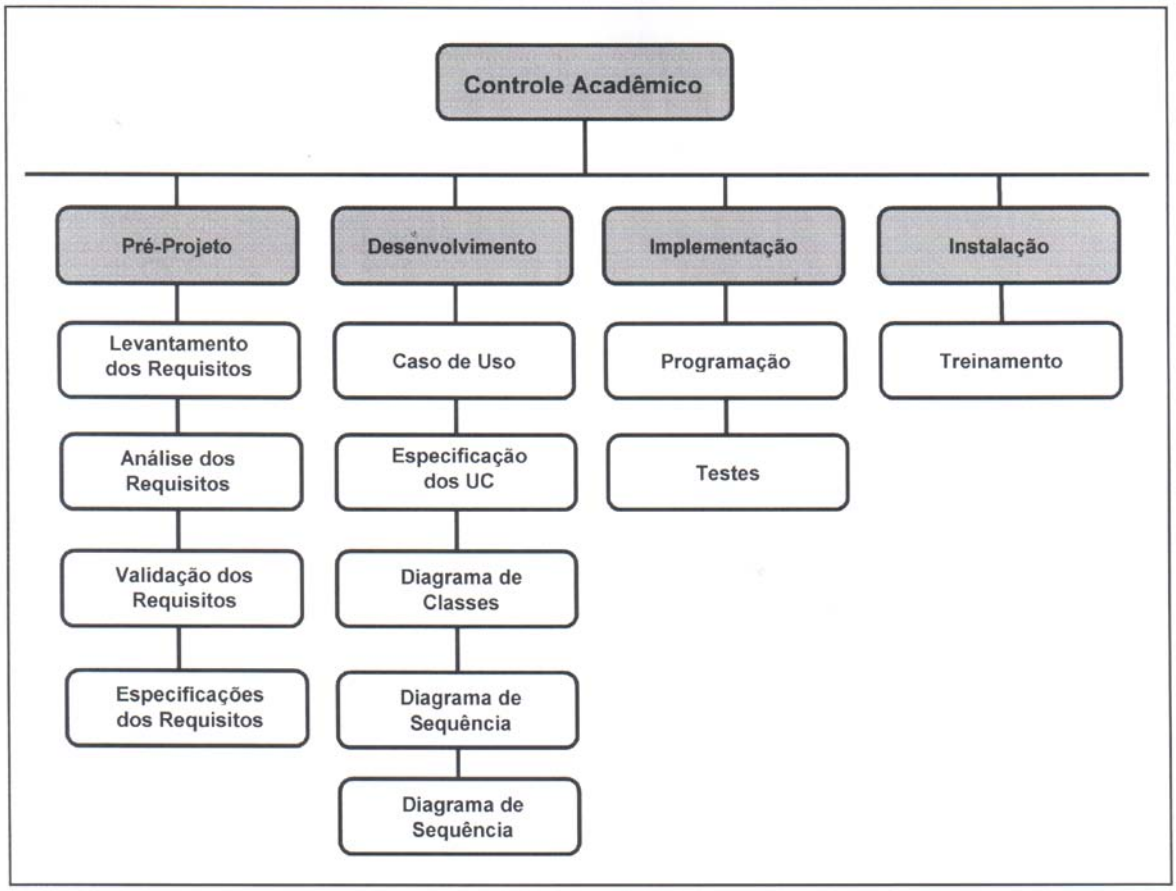


Diagrama WBS

4.7 – Diagrama de Gantt

Cronograma	MAR		ABR		MAI		JUN		JUL		AGO		SET		OUT		NOV	
Elaboração do Pré-Projeto																		
Revisão Bibliográfica - conceito e teoria																		
Elaboração e Apresentação da qualificação																		
Projeto, desenvolvimento da interface de software																		
Elaboração de testes																		
Redação Preliminar																		
Redação Final e Encadernação																		
Entrega Final																		

5 – Conclusão e Trabalhos Futuros

Conclui que o sistema S.C.A. proposto irá suprir a maioria senão todos os controles feitos manualmente na escola que está adotando o sistema, pois passará a ser feito quase tudo no computador, centralizando em uma única pessoa, no caso a SECRETÁRIA, a movimentação da Escola toda, mas permitindo que o Supervisor, no caso o DONO, possa supervisionar de perto o trabalho da secretária sem Ter que atrapalhar o seu serviço.

O sistema poderá muito bem trabalhar em uma plataforma de rede por ser um Software Cliente-Servidor (desenvolvido em uma plataforma de programação Cliente-Servidor), sendo necessário apenas que se crie uma área comum pelo menos para o banco de dados do programa.

O trabalho a ser desenvolvido futuramente é o aperfeiçoamento do software para que possa ser comercializado visando que a maioria das escolas de cursos de treinamentos da região não possui um sistema informatizado, ou seja, tudo é controlado manualmente como na Empresa que está sendo instalado o programa.

6 - Bibliografia

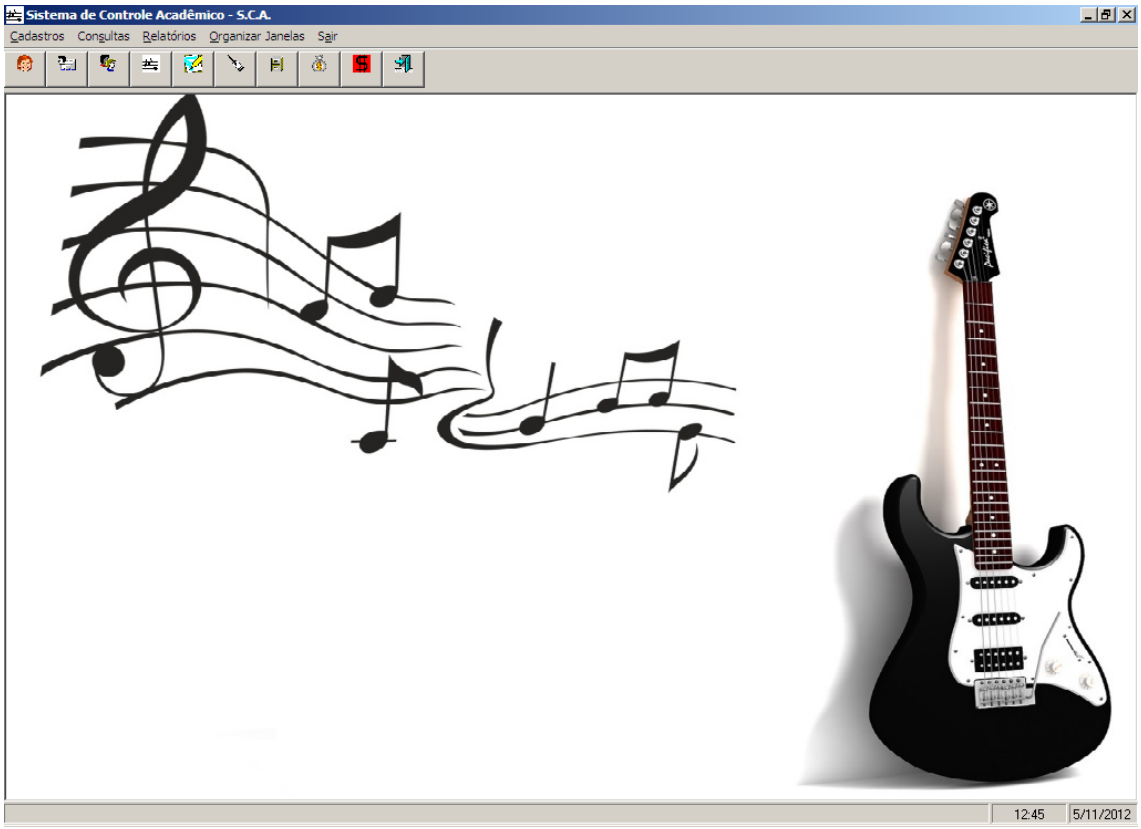
[H.E.E.] ERIKSSON, Hans-Erik & PENKER, Magnus. UML Toolkit. Editora Wiley, 1998.

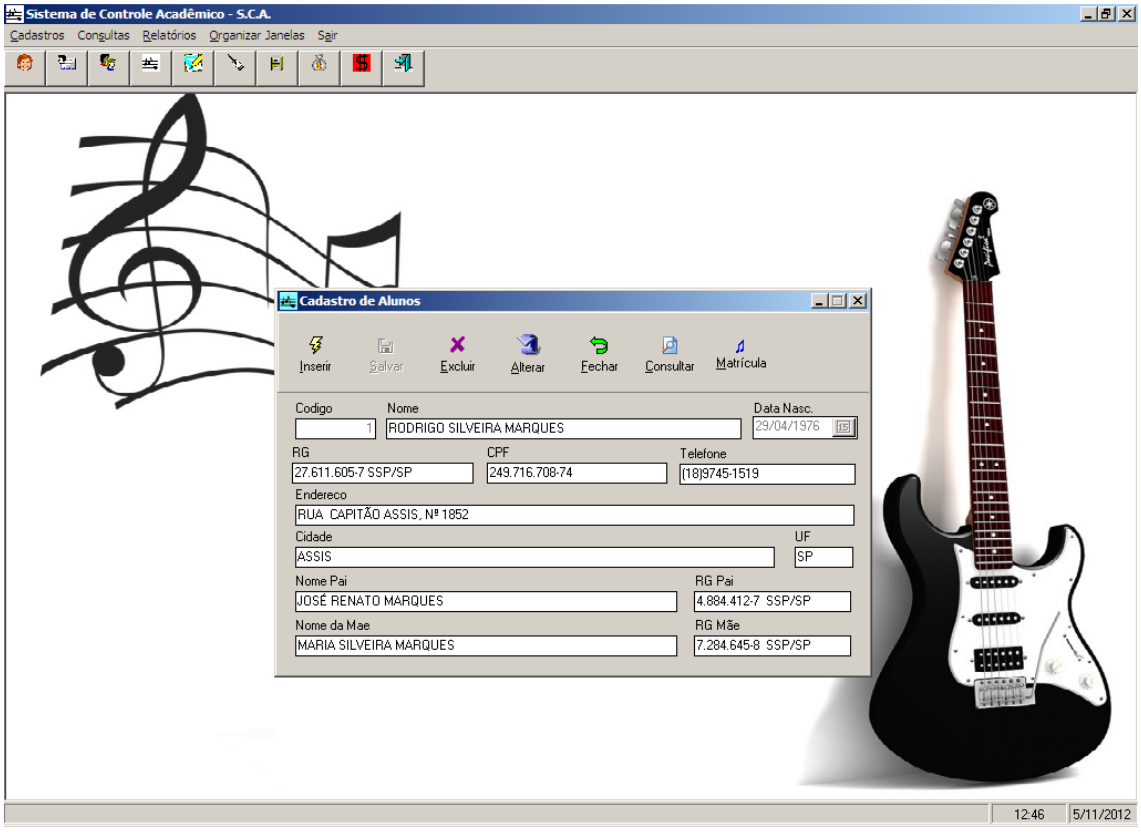
[R.P.] PRESSMAN, Roger. Engenharia de Software.3ª ed. Editora McGrawHill, 1995.

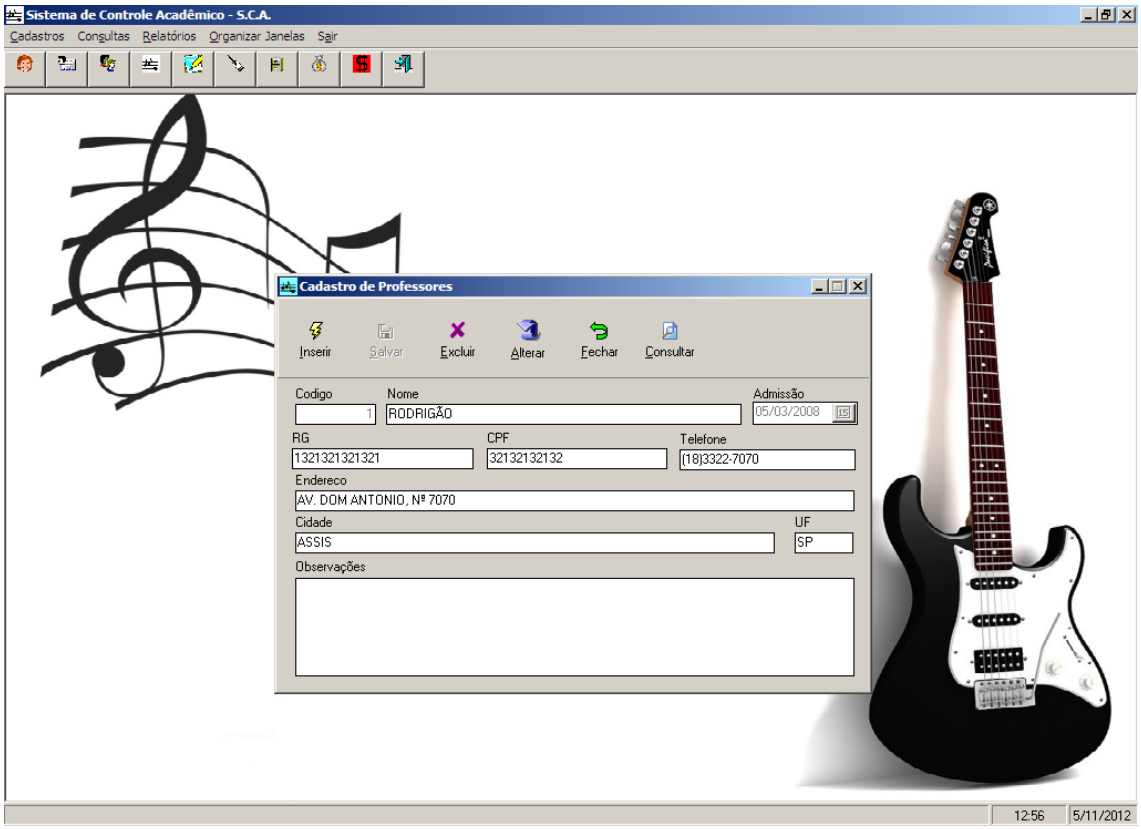
[D.O.] OSIER, Dan. Aprenda em 21 dias Delphi 2.0/Dan Osier; tradução de João E. N. Tortello. Rio de Janeiro: Campus, 1997.

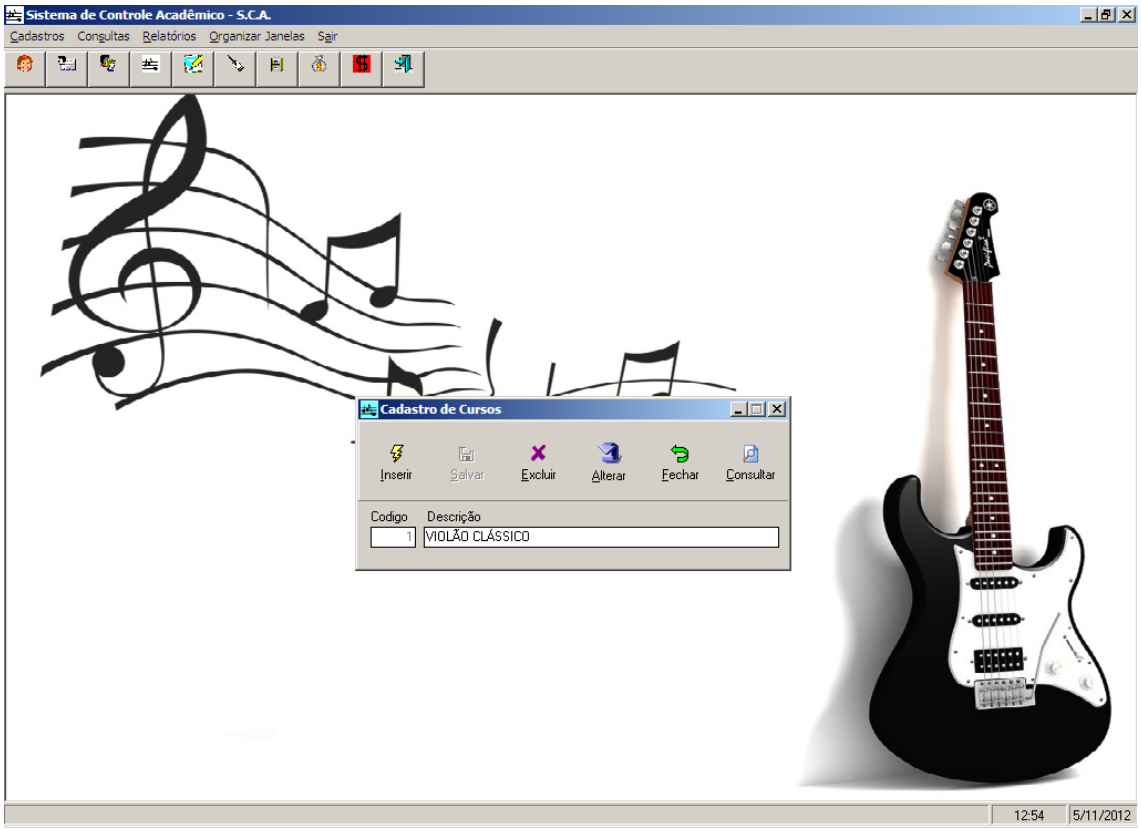
[B.I.] Fundamentos de Delphi 3.0 Client/Server. Copyright © Borland International, Inc. All Rights Reserved: Borland, 1998.

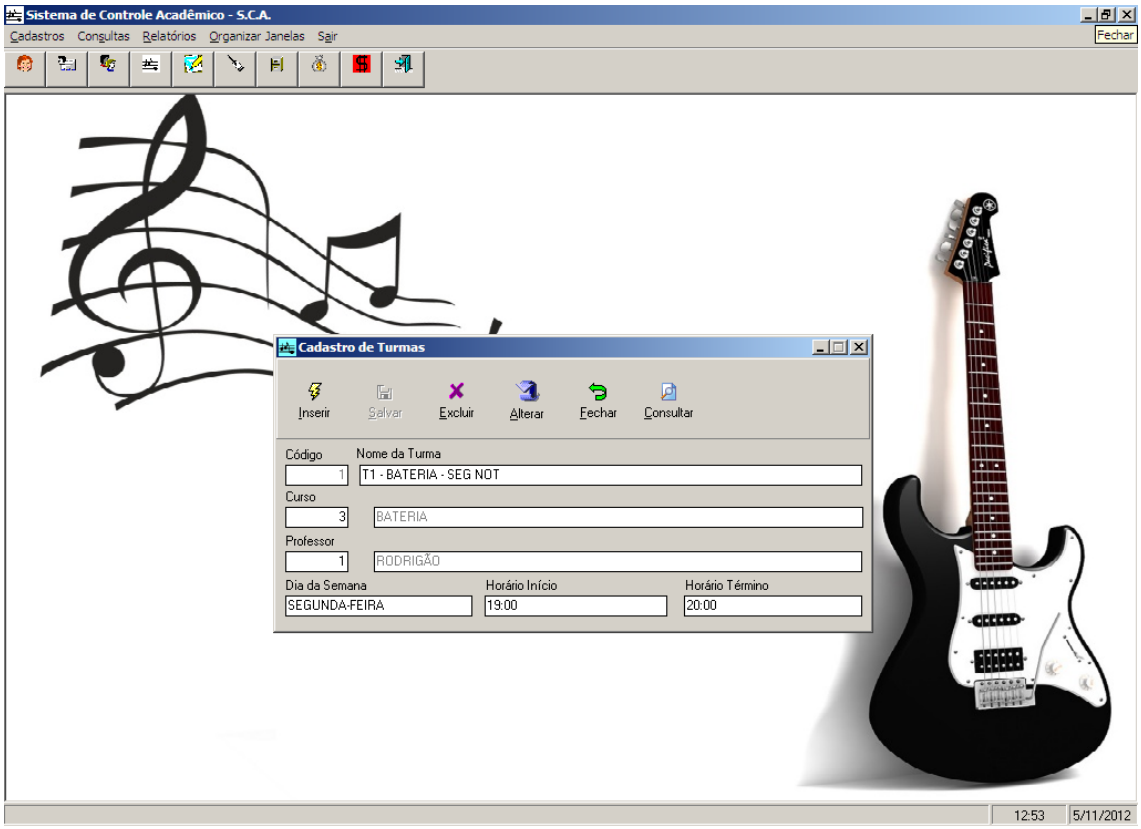
7 – Anexos

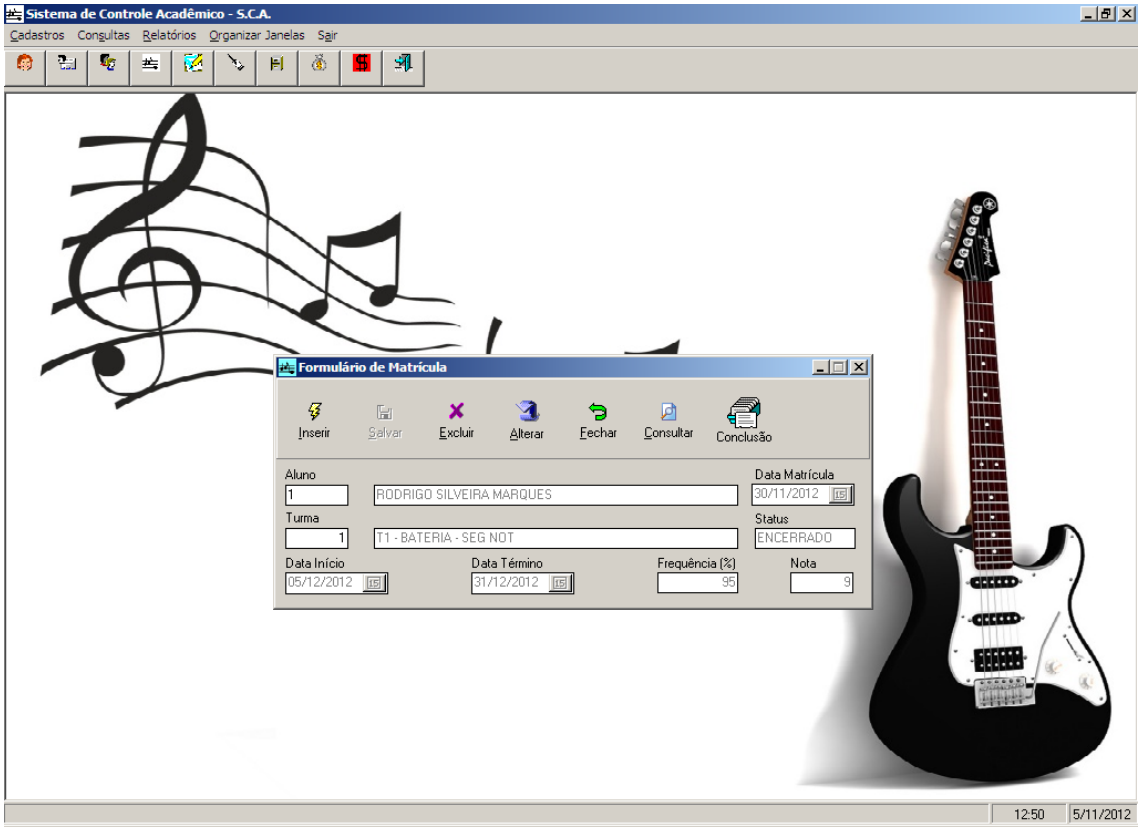

Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
12:45
5/11/2012

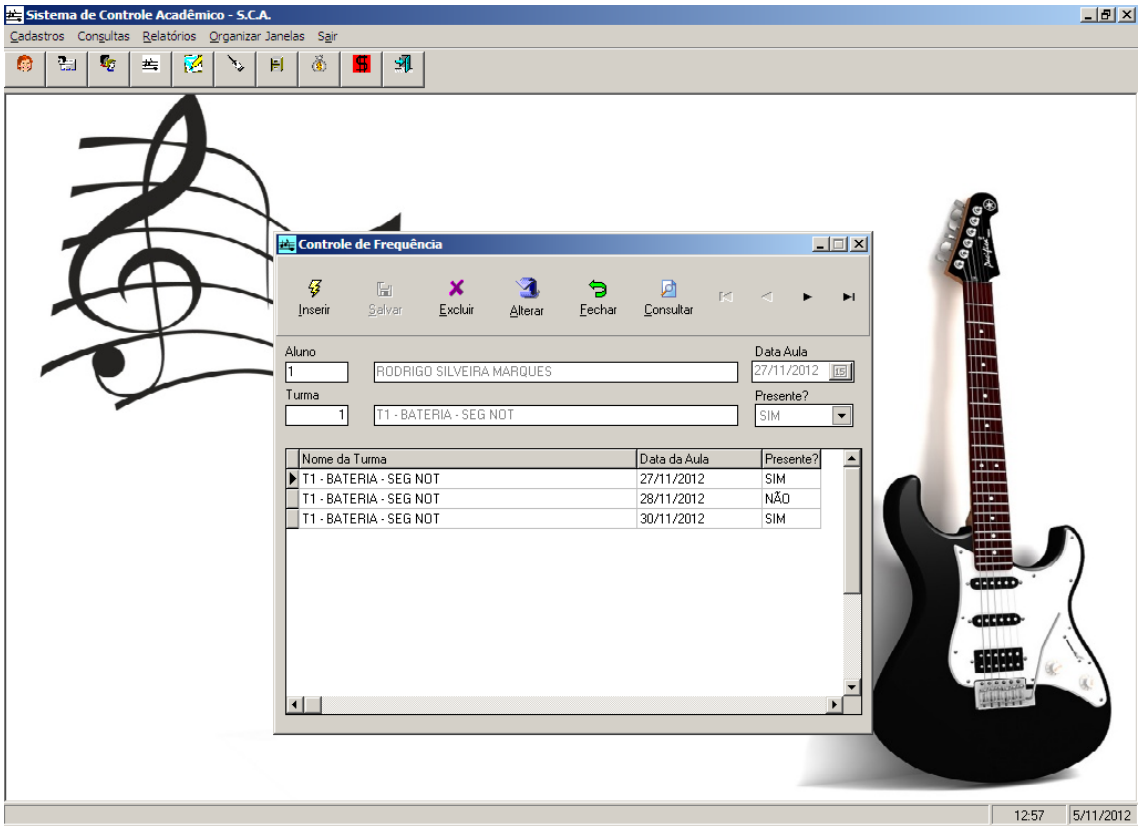

Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Cadastro de Alunos
Inserir
Salvar
Excluir
Alterar
Fechar
Consultar
Matrícula
Codigo
1
Nome
RODRIGO SILVEIRA MARQUES
Data Nasc.
29/04/1976
RG
27.611.605-7 SSP/SP
CPF
249.716.708-74
Telefone
(18)9745-1519
Endereco
RUA CAPITÃO ASSIS, Nº 1852
Cidade
ASSIS
UF
SP
Nome Pai
JOSÉ RENATO MARQUES
RG Pai
4.884.412-7 SSP/SP
Nome da Mae
MARIA SILVEIRA MARQUES
RG Mãe
7.284.645-8 SSP/SP
12:46
5/11/2012

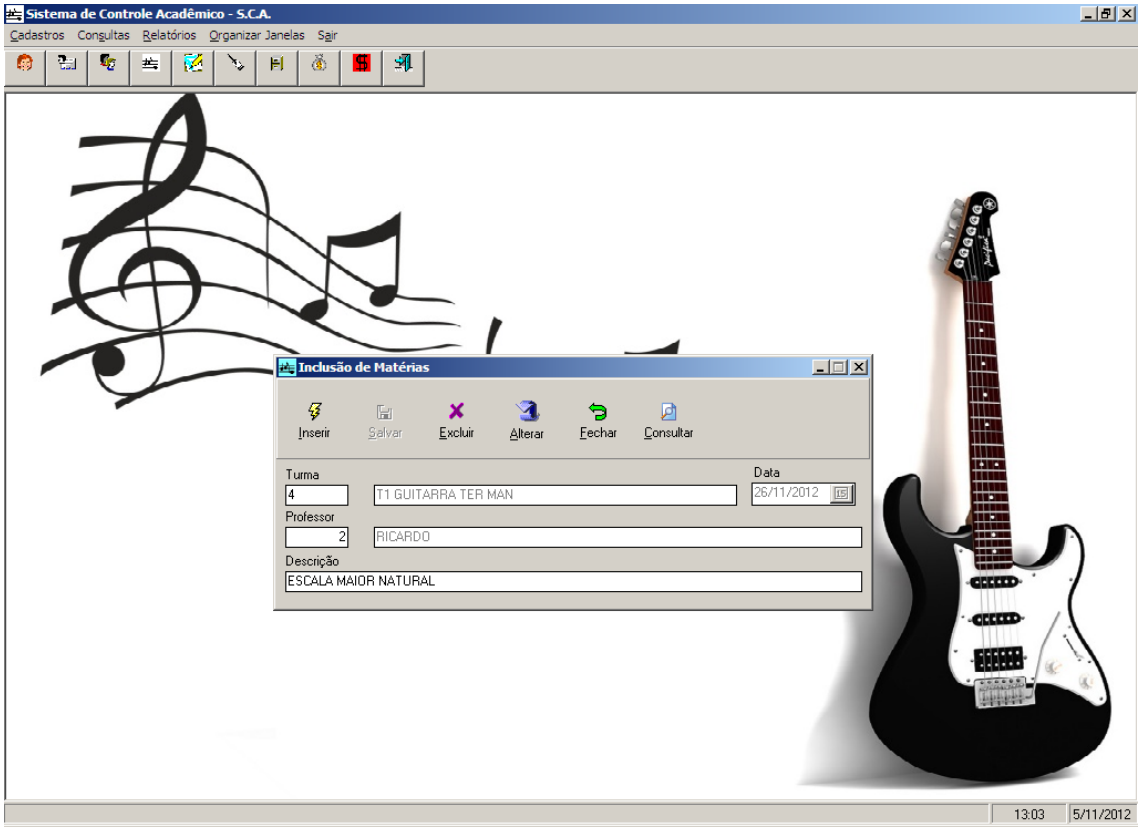

Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Cadastro de Professores
Inserir
Salvar
Excluir
Alterar
Fechar
Consultar
Codigo
Nome
Admissão
1
RODRIGÃO
05/03/2008
RG
CPF
Telefone
1321321321321
32132132132
(18)3322-7070
Endereco
AV. DOM ANTONIO, Nº 7070
Cidade
UF
ASSIS
SP
Observações
12:56
5/11/2012

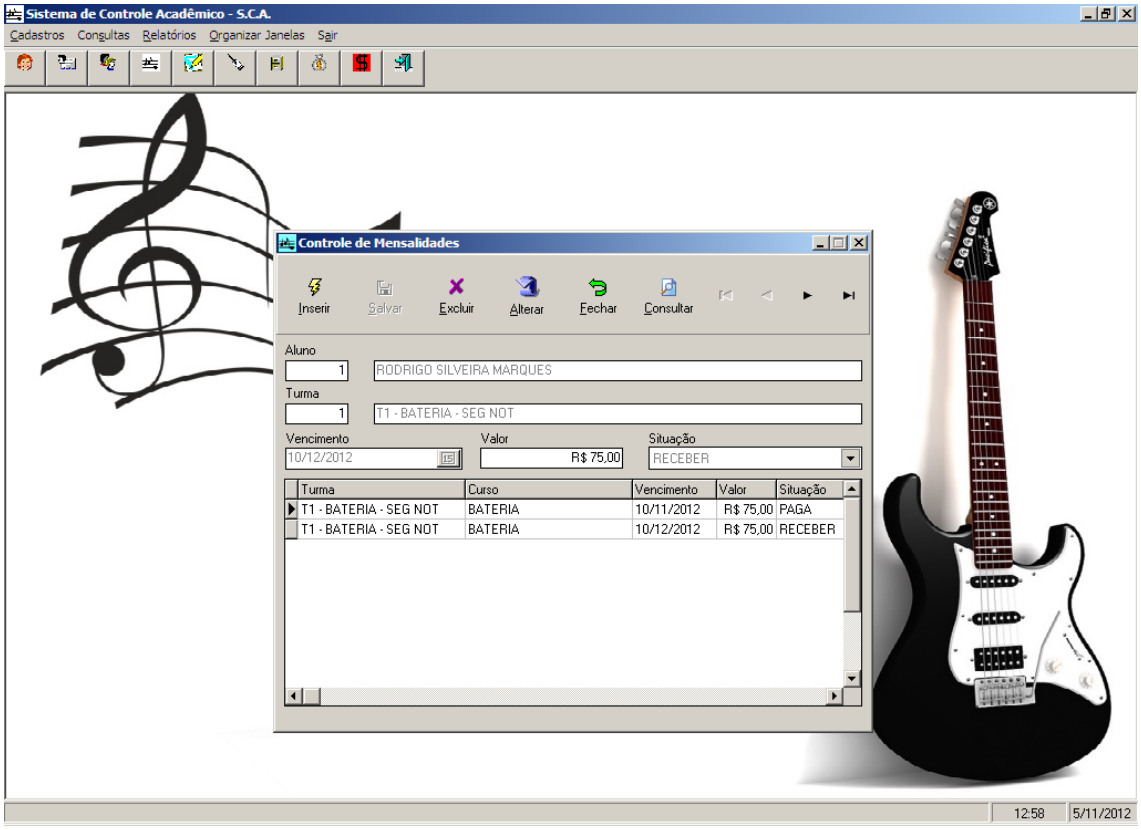

Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Cadastro de Cursos
Inserir
Salvar
Excluir
Alterar
Fechar
Consultar
Codigo
Descrição
1
VIOLÃO CLÁSSICO
12:54
5/11/2012

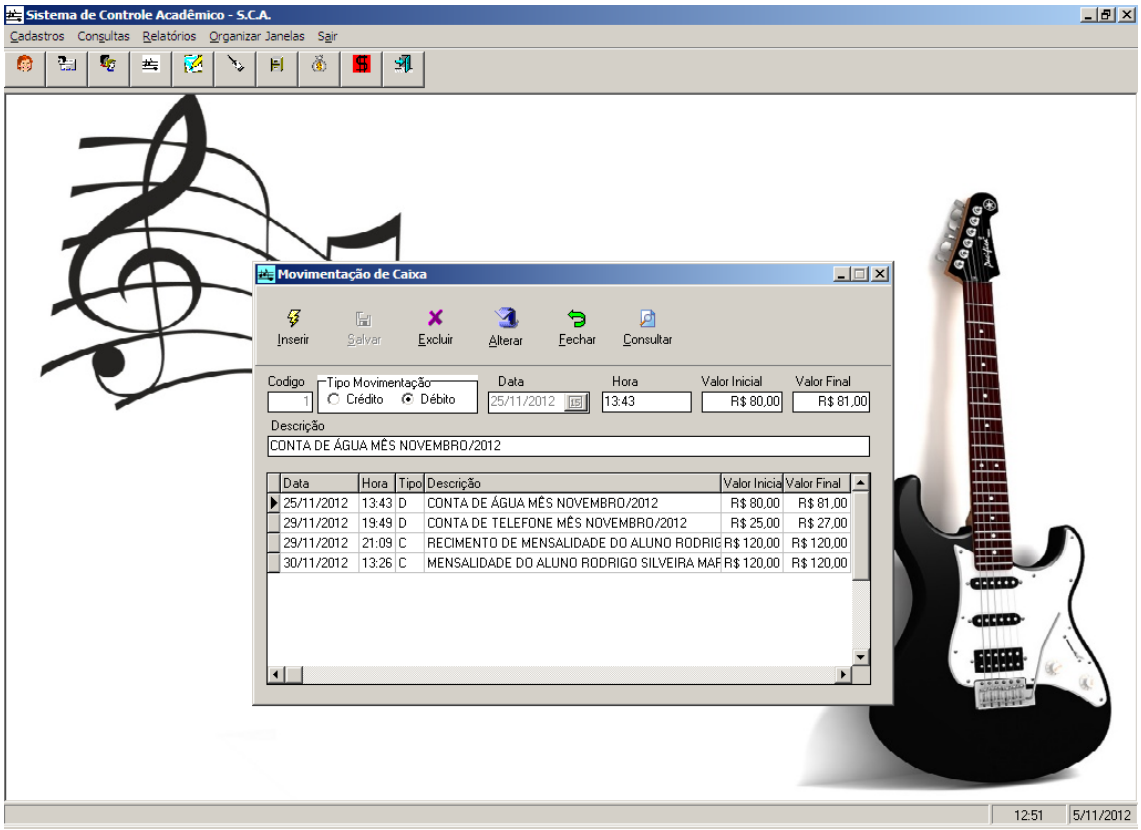

Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Fechar
Cadastro de Turmas
Inserir
Salvar
Excluir
Alterar
Fechar
Consultar
Código
Nome da Turma
1
T1 - BATERIA - SEG NOT
Curso
3
BATERIA
Professor
1
RODRIGÃO
Dia da Semana
Horário Início
Horário Término
SEGUNDA-FEIRA
19:00
20:00
12:53
5/11/2012


Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Formulário de Matrícula
Inserir
Salvar
Excluir
Alterar
Fechar
Consultar
Conclusão
Aluno
1
RODRIGO SILVEIRA MARQUES
Data Matrícula
30/11/2012
Turma
1
T1 - BATERIA - SEG NOT
Status
ENCERRADO
Data Início
05/12/2012
Data Término
31/12/2012
Frequência (%)
95
Nota
9
12:50
5/11/2012

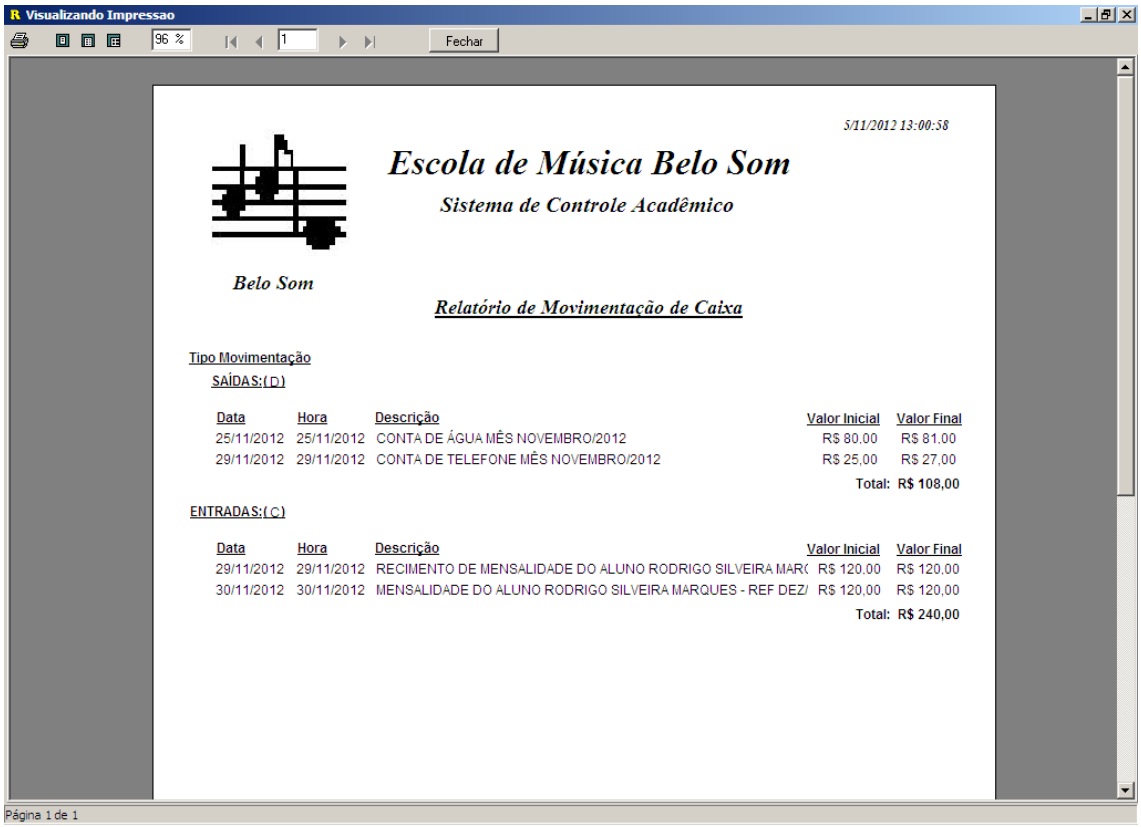

Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Controle de Frequência
Inserir Salvar Excluir Alterar Fechar Consultar
Aluno
1
RODRIGO SILVEIRA MARQUES
Data Aula
27/11/2012
Turma
1
T1 - BATERIA - SEG NOT
Presente?
SIM
Nome da Turma
Data da Aula
Presente?
T1 - BATERIA - SEG NOT
27/11/2012
SIM
T1 - BATERIA - SEG NOT
28/11/2012
NÃO
T1 - BATERIA - SEG NOT
30/11/2012
SIM
12:57
5/11/2012


Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Inclusão de Matérias
Inserir Salvar Excluir Alterar Fechar Consultar
Turma
4
T1 GUITARRA TER MAN
Data
26/11/2012
Professor
2
RICARDO
Descrição
ESCALA MAIOR NATURAL
13:03
5/11/2012


Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Controle de Mensalidades
Inserir Salvar Excluir Alterar Fechar Consultar
Aluno
1
RODRIGO SILVEIRA MARQUES
Turma
1
T1 - BATERIA - SEG NOT
Vencimento
10/12/2012
Valor
R$ 75,00
Situação
RECEBER
Turma Curso Vencimento Valor Situação
T1 - BATERIA - SEG NOT BATERIA 10/11/2012 R$ 75,00 PAGA
T1 - BATERIA - SEG NOT BATERIA 10/12/2012 R$ 75,00 RECEBER
12:58
5/11/2012


Sistema de Controle Acadêmico - S.C.A.
Cadastros Consultas Relatórios Organizar Janelas Sair
Movimentação de Caixa
Inserir Salvar Excluir Alterar Fechar Consultar
Codigo
1
Tipo Movimentação
Crédito
Débito
Data
25/11/2012
Hora
13:43
Valor Inicial
R$ 80,00
Valor Final
R$ 81,00
Descrição
CONTA DE ÁGUA MÊS NOVEMBRO/2012
Data Hora Tipo Descrição Valor Inicia Valor Final
25/11/2012 13:43 D CONTA DE ÁGUA MÊS NOVEMBRO/2012 R$ 80,00 R$ 81,00
29/11/2012 19:49 D CONTA DE TELEFONE MÊS NOVEMBRO/2012 R$ 25,00 R$ 27,00
29/11/2012 21:09 C RECIMENTO DE MENSALIDADE DO ALUNO RODRIG R$ 120,00 R$ 120,00
30/11/2012 13:26 C MENSALIDADE DO ALUNO RODRIGO SILVEIRA MAF R$ 120,00 R$ 120,00
12:51
5/11/2012


Visualizando Impressao
96 %
1
Fechar
Escola de Música Belo Som
Sistema de Controle Acadêmico
Belo Som
Listagem de Alunos por Turma
Turma:
1 -T1 - BATERIA - SEG NOT
Curso
BATERIA
Professor
RODRIGÃO
Dia Semana
SEGUNDA-FEIRA
Hora Início
19:00
Hora Término
20:00
Cod. Aluno
Nome do Aluno
Status Curso
5
CARLOS JOSE DOS SANTOS
MATRICULADO
Total: 1
2 -T2 - BATERIA - SEG NOT
BATERIA
Professor
RODRIGÃO
Dia Semana
SEGUNDA-FEIRA
Hora Início
20:00
Hora Término
21:00
Cod. Aluno
Nome do Aluno
Status Curso
2
TESTE
MATRICULADO
Total: 1
3 -T3 - BATERIA - SEG NOT
BATERIA
Professor
RICARDO
Dia Semana
SEGUNDA
Hora Início
21:00
Hora Término
22:00
Cod. Aluno
Nome do Aluno
Status Curso
6
MARCOS JOSE DOS SANTOS
MATRICULADO
4
FELIPE DA SILVA
MATRICULADO
Total: 2
Página 1 de 1


Visualizando Impressao
96 %
1
Fechar
5/11/2012 13:00:58
Escola de Música Belo Som
Sistema de Controle Acadêmico
Belo Som
Relatório de Movimentação de Caixa
Tipo Movimentação
SAÍDAS:(D)
Data
Hora
Descrição
Valor Inicial
Valor Final
25/11/2012
25/11/2012
CONTA DE ÁGUA MÊS NOVEMBRO/2012
R$ 80,00
R$ 81,00
29/11/2012
29/11/2012
CONTA DE TELEFONE MÊS NOVEMBRO/2012
R$ 25,00
R$ 27,00
Total: R$ 108,00
ENTRADAS:(C)
Data
Hora
Descrição
Valor Inicial
Valor Final
29/11/2012
29/11/2012
RECIMENTO DE MENSALIDADE DO ALUNO RODRIGO SILVEIRA MAR(
R$ 120,00
R$ 120,00
30/11/2012
30/11/2012
MENSALIDADE DO ALUNO RODRIGO SILVEIRA MARQUES - REF DEZ/
R$ 120,00
R$ 120,00
Total: R$ 240,00
Página 1 de 1