UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
Bacharelado em Sistemas de Informação

ANÁLISE E PROJETO ORIENTADOS A OBJETOS

Profª Elisa Yumi

Sistema da Polícia Civil

Parte 3

Nº 1 – Hiero Martinelli – 79886646
Nº 2 - Daniele Hidalgo Boscolo – 7986625
Nº 3 - Eduardo Sigrist Ciciliato – 7986542

São Carlos/SP
2013

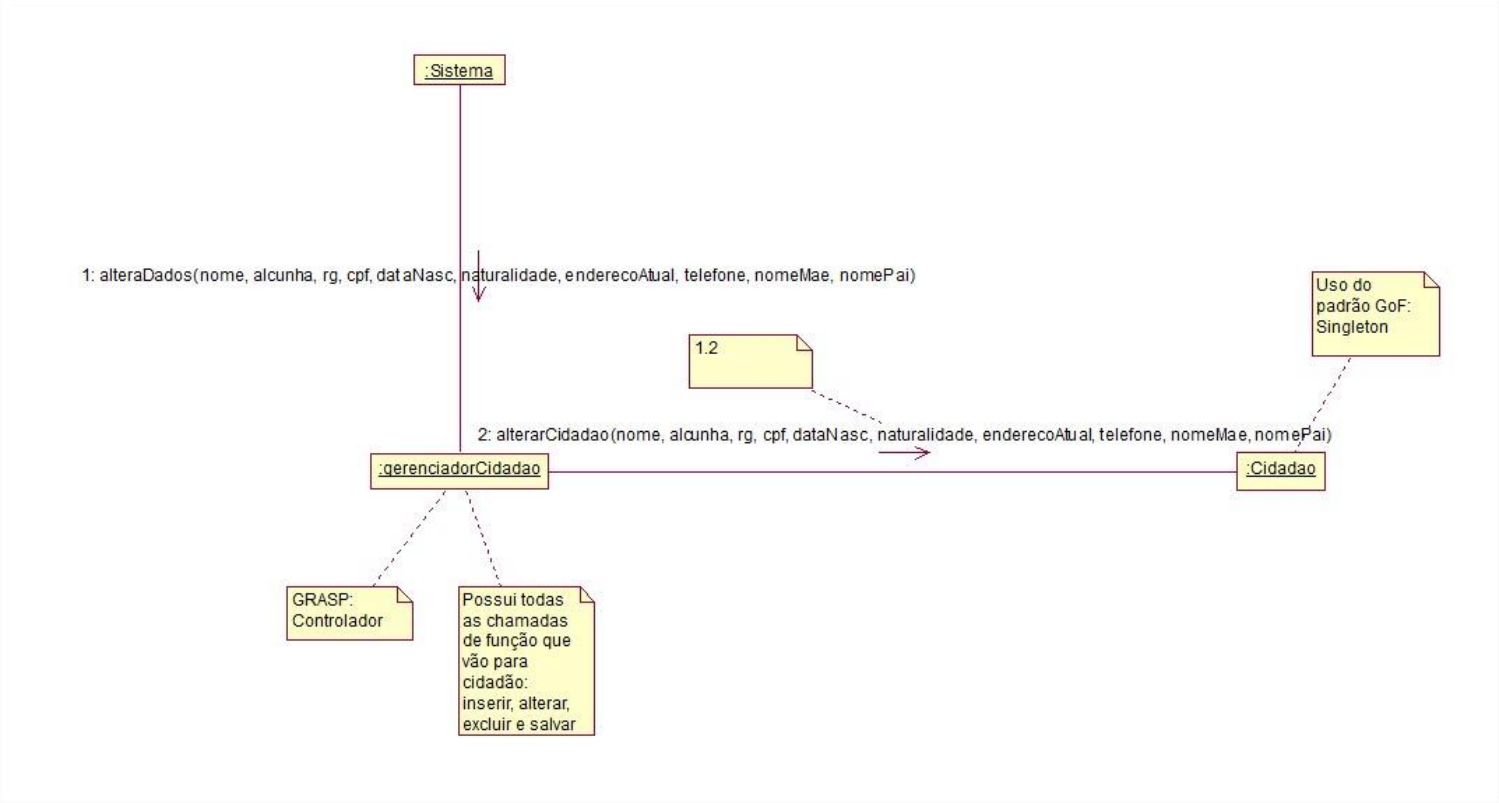
Sumário

Introdução

Apresentamos nesse projeto uma representação dos diagramas de comunicação e diagramas de classe do Sistema de Polícia Civil, baseados nos diagramas realizados nas outras partes do trabalho, bem como o código gerado pelo Rational Rose para o diagrama de classes.

I - Diagramas de Comunicação

1. Operação “alteraDados”:



Neste diagrama foram utilizados os padrões GRASP Controlador, Especialista e Coesão Alta e o padrão GoF Singleton.

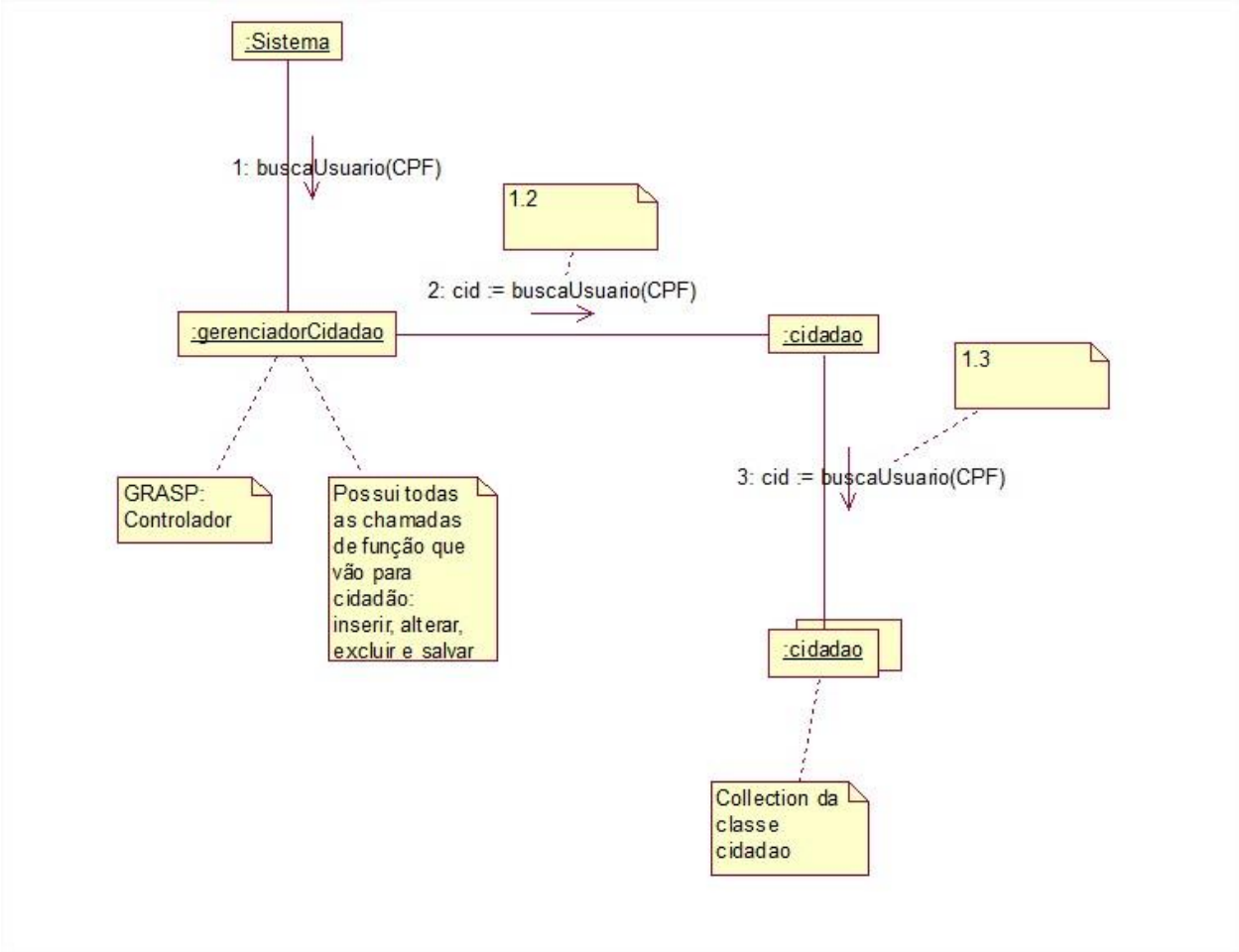
A operação alterarCidadao é de responsabilidade da classe Cidadao, já que todos os dados que estão sendo alterados são atributos dessa mesma classe. (Especialista/Coesão Alta)

Existe um controlador gerenciadorCidadao que manda as mensagens para a classe Cidadao e recebe as respostas de Cidadao. Isso foi feito para tirar um pouco do peso do sistema, permitindo que outras partes realizem o envio de mensagens a partir do ponto inicial e assim evitando que o sistema esteja sobrecarregado. (Controlador)

O padrão GoF Singleton é utilizado para garantir que haverá apenas uma instância de Cidadao alocada ao mesmo tempo, garantindo assim economia de

memória, já que só se alterará os dados de uma pessoa ao mesmo tempo nesse caso.

2. Operação “buscaUsuario”

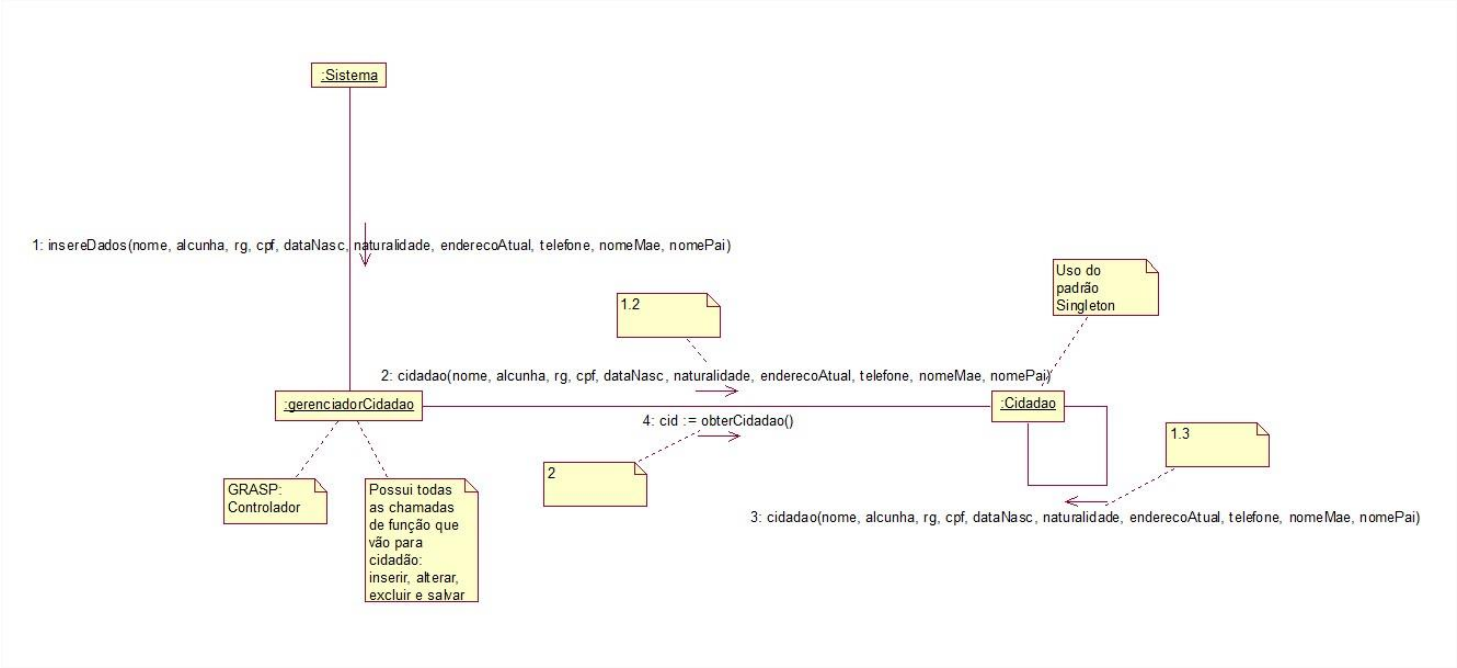


Neste diagrama foram utilizados os padrões GRASP Controlador, Especialista e Coesão Alta.

A operação buscaUsuario é realizada na classe Cidadao dentro de uma collection de Cidadao. Isso garante que a classe responsável pela busca seja a mesma classe que possui os dados necessários para realizá-la. (Especialista/Coesão Alta)

Existe um controlador gerenciadorCidadao que tira a sobrecarga do sistema e garante que as mensagens sejam transferidas para a classe responsável. (Controlador)

3. Operação “insereDados”

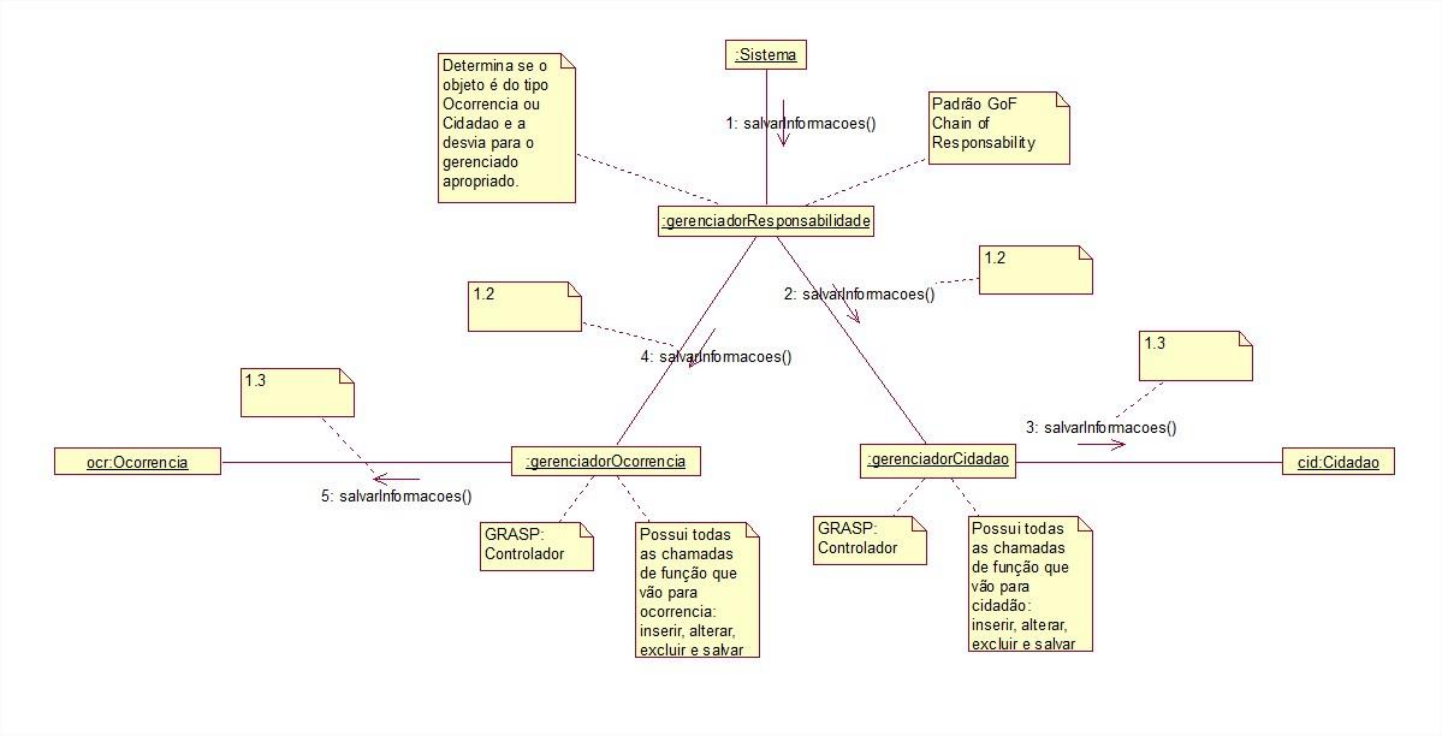


Neste diagrama foram utilizados os padrões GRASP Controlador, Especialista, Criador e Coesão Alta e o padrão GoF Singleton.

A operação insereDados é realizada pela classe Cidadao que se responsabiliza por criar um objeto da classe Cidadao, se este já existir o mesmo objeto é utilizado, economizando memória e facilitando o acesso ao cidadão cadastrado, caso seja realizada uma operação com o mesmo cadastro. (Especialista/Criador/Coesão Alta/Singleton)

Existe um controlador gerenciadorCidadao que envia as mensagens e recebe respostas da classe Cidadao, aliviando o trabalho do sistema. (Controlador)

4. Operação “salvarInformacoes”



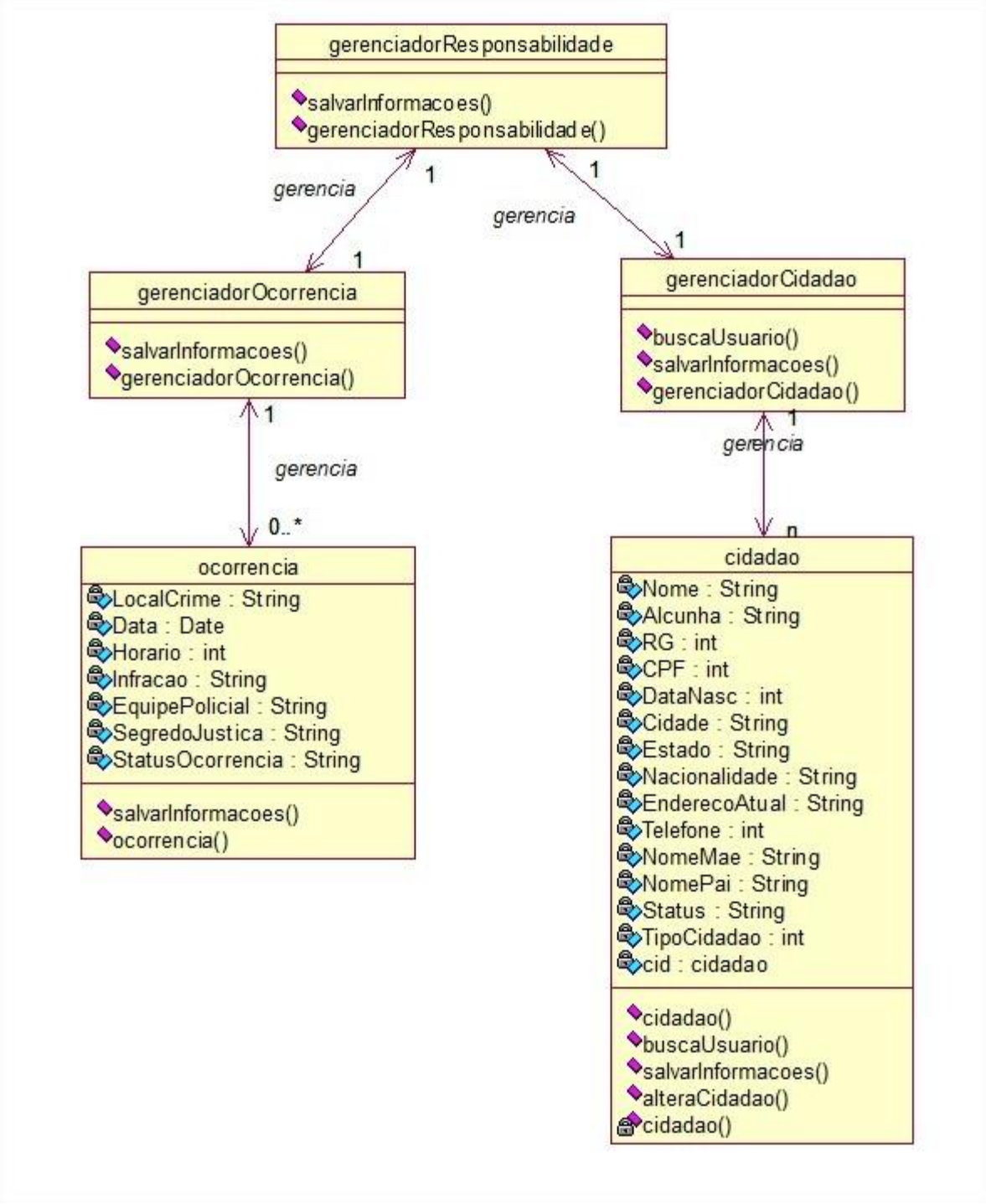
Neste diagrama foram utilizados os padrões GRASP Controlador, Especialista e Coesão Alta e o padrão GoF Chain of Responsability.

Como pode-se notar, todos os métodos são enviados para as classes mais apropriadas para cuidar deles. (Especialista/Coesão Alta)

O gerenciadorResponsabilidade redireciona as chamadas de acordo com qual das classes é responsável por aquele chamado (Cidadao ou Ocorrencia). (Controlador/Chain of Responsability).

Além disso, cada classe possui seu próprio gerenciador que redireciona as mensagens e recebe as respostas de modo a nunca sobrecarregar ninguém. (Controlador/Coesão Alta)

II - Diagrama de Classes



III - Código Gerado

Classe cidadao

```
public class cidadao {
   private String Nome;
   private String Alcunha;
   private int RG;
   private int CPF;
   private int DataNasc;
   private String Cidade;
   private String Estado;
   private String Nacionalidade;
   private String EnderecoAtual;
   private int Telefone;
   private String NomeMae;
   private String NomePai;
   private String Status;
   private int TipoCidadao;
   private cidadao cid;
   public ÁrvoreDeRelacionamentos theÁrvoreDeRelacionamentos;
   public gerenciadorCidadao theGerenciadorCidadao;

   private cidadao() {}
   public cidadao(String nome, String alcunha, int RG, int CPF, String
Cidade, String Estado, String Nacionalidade, String EnderecoAtual,
String Telefone, String NomeMae, String NomePai, String Status, int
TipoCidadao, int DataNasc) {}

   public void buscaUsuario(int CPF) {}
   public void salvarInformacoes() {}

   public void alteraCidadao(String nome, String alcunha, int RG, int
CPF, int DataNasc, String Cidade, String Estado, String Nacionalidade,
```

```
String EnderecoAtual, String Telefone, String NomeMae, String NomePai,
String Status, int TipoCidadao) {}

    public int getNome () {
      return nome;
    }
    public void set Nome (int Nome) {
      this.nome = nome;
    }
    [...]//Para cada atributo existe um get e set igual
}
```

Classe gerenciadorOcorrencia

```
public class gerenciadorOcorrencia
{
   public ocorrencia theOcorrencia[];
   public gerenciadorResponsabilidade theGerenciadorResponsabilidade;

   public gerenciadorOcorrencia() {}
   public void salvarInformacoes() {}
}
```

Classe gerenciadorCidadao

```
public class gerenciadorCidadao
{
   public cidadao theCidadao;
   public gerenciadorResponsabilidade theGerenciadorResponsabilidade;

   public gerenciadorCidadao() {}
   public void buscaUsuario(int CPF) {}
   public void salvarInformacoes() {}
}
```

Classe gerenciadorResponsabilidade

```
public class gerenciadorResponsabilidade
```

```
{
   public gerenciadorCidadao theGerenciadorCidadao;
   public gerenciadorOcorrencia theGerenciadorOcorrencia;

   public gerenciadorResponsabilidade(){}
   public void salvarInformacoes() {}
}
```

Classe ocorrencia

```
public class ocorrencia
{
   private String LocalCrime;
   private Date Data;
   private int Horario;
   private String Infracao;
   private String EquipePolicial;
   private String SegredoJustica;
   private String StatusOcorrencia;
   public gerenciadorOcorrencia theGerenciadorOcorrencia;

   public ocorrencia() {}
   public void salvarInformacoes() {}
}
```

Observações:

Os métodos get e set não estão declarados no código devido à repetição que estas partes dos códigos causariam. No momento da implementação, esses métodos serão recolocados no código.

Nos diagramas de comunicação não foi possível colocar os números 1.1 ou 1.2 para operações que continuavam dentro de outras. Nesse caso foram usados notes para considerar essas continuações. Além disso, não foi possível colocar as mensagens iniciais que aparecem antes das classes iniciais dos diagramas, o Rational

Rose só aceita fazer uma ligação entre duas classes e não permite fazer retas a apenas um objeto.

Conclusão

Este trabalho auxiliou bastante na compreensão dos diversos diagramas já estudados em sala de aula, bem como na expansão dos conhecimentos adquiridos anteriormente e de como as primeiras fases de projeto refletem no final do mesmo.
Conseguimos perceber que a tarefa de projetar um sistema não é fácil e requer muita experiência e dedicação de todos os envolvidos.

O aprendizado adquirido na ferramenta Rational Rose com certeza refletirá de maneira positiva no nosso futuro como projetistas.