Mestrado em Engenharia Biomédica – Informática Médica

1º Ano | 2022/2023 | 1º semestre

Aplicações Distribuídas

Trabalho Prático de Grupo – Sistema CODU

Grupo 4

Docente:

António Sousa

Hugo Silva PG50416

Gonçalo Carvalho PG50392

Tomás Lima PG50788

Índice

Introdução

A gestão de um sistema de comunicação para um serviço de emergências médicas é uma tarefa complicada e de alta responsabilidade. Quando estão em risco as vidas de pessoas, tudo tem de correr na perfeição e os meios acionados devem ser os adequados para cada situação.

Neste projeto, visou-se a criação de uma plataforma de resposta a emergências médicas, responsável pela mobilização das viaturas de socorro disponíveis, numa estrutura designada por Centro de Orientação de Doentes Urgentes (CODU). Nesta projeção semi-complexa, pretende-se que a ativação dos mecanismos de socorro seja feita de forma correta e adequada às situações, respeitando a coerência das ações, como por exemplo, não ativando um meio que se encontra já em missão e, portanto, indisponível. É também importante o registo dos dados das ocorrências e de todo o processo de resposta, para se poderem criar ferramentas estatísticas que permitam avaliar a eficiência do sistema.

Para melhor compreensão do sistema CODU e a função de cada meio de socorro por este gerido, consultou-se o site do INEM, que diz que, de uma forma geral, a ambulância é um meio de socorro capaz de realizar suporte básico de vida (SBV) e transporte de vítimas. A moto é um meio apenas com capacidade de prestar SBV, mas com maior agilidade em ambiente citadino. A viatura é um meio que transporta uma equipa médica munida de equipamento para suporte avançado de vida (SAV) para o local da ocorrência, mas sem capacidade de transporte da vítima. Já o helicóptero, é um meio de transporte e prestador de SAV.[1]

Planeamento e Estruturação

O primeiro passo deve ser sempre o definir de que forma se vai estruturar a aplicação, antes de se iniciar a escrita do código. Isto permite evitar a reescrita de partes do código devido a erros de planeamento e torna mais expedita a construção deste.

O cliente da aplicação estará diretamente em contacto somente com a Interface, que apresentará os métodos a serem utilizadas por este, de forma que se possa registar o evento e ativar os meios. A transferência das informações fornecidas pelo cliente para o Gestor (CODU) será feita através de um Servidor RMI (*Remote Method Invocation*), que tal como o nome diz, permite que objetos invoquem métodos de outros objetos a correr numa Máquina Virtual de Java diferente. O Gestor (CODU) ficará então responsável por assegurar a boa aplicação dos métodos associados às diferentes classes necessárias para o funcionamento do CODU: técnico operador, meios, evento, grau e local. Esta estrutura é ilustrada pelo diagrama da Figura 1.

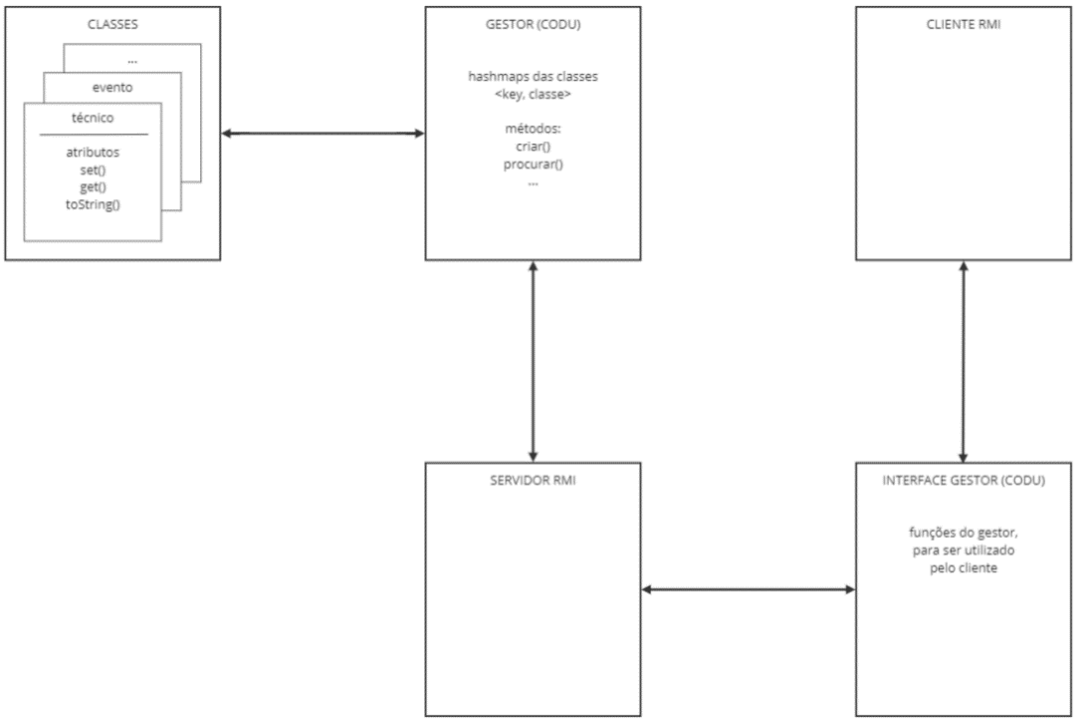

Figura 1 – Diagrama da estrutura básica a ser implementada em Java.

Depois da ideia inicial definida, criou-se um segundo diagrama onde se detalhou mais pormenorizadamente as classes a criar, assim como os seus atributos, métodos e relações a ser estabelecidas entre si.

As três principais classes são aquelas que estão diretamente relacionadas com a ocorrência e representam as componentes físicas do sistema: Técnico (quem atende a chamada), MeioSocorro (viatura mobilizada) e o Evento em si. Para facilitar na caracterização e articulação destas, foram criadas duas outras classes – Grau e Local – que fazem a quantificação da gravidade da ocorrência e a normalização das informações relativas às posições, respetivamente. O diagrama completo é disponibilizado em anexo (ver Anexo 1).

Para melhor perceber as relações estabelecidas entre as diferentes classes, criou-se um terceiro diagrama, representativo destas, apresentado na Figura 2. Como se pode verificar, a classe Evento adquire o papel principal neste sistema, relacionando-se diretamente com todas as outras classes. Justificando as relações adotadas entre classes:

- **Evento - Técnico**: relação de N para 1, já que um evento tem apenas um técnico atendedor associado, sendo que esse mesmo técnico pode atender a vários eventos.
- **Evento – MeioSocorro**: relação N para N, visto que um determinado meio pode ser usado em vários eventos e um evento pode necessitar do envio de vários meios simultâneos.
- **Evento – Grau**: relação de 1 para N, pois o mesmo Grau pode descrever vários eventos, mas um evento tem apenas um grau associado.
- **Evento – Local**: relação N para 1, já que cada evento pode ter apenas um local associado, sendo que no mesmo local pode haver a ocorrência de vários eventos.
- **MeioSocorro – Grau**: relação é N para N, pois, apesar de ser 1 para N nos casos da moto, helicóptero e veículo, a ambulância é indicada para responder a três graus.
- **MeioSocorro – Posto (Local)**: relação N para 1, visto que cada meio pode ter apenas um posto associado, sendo que o mesmo posto pode gerir vários meios em simultâneo.

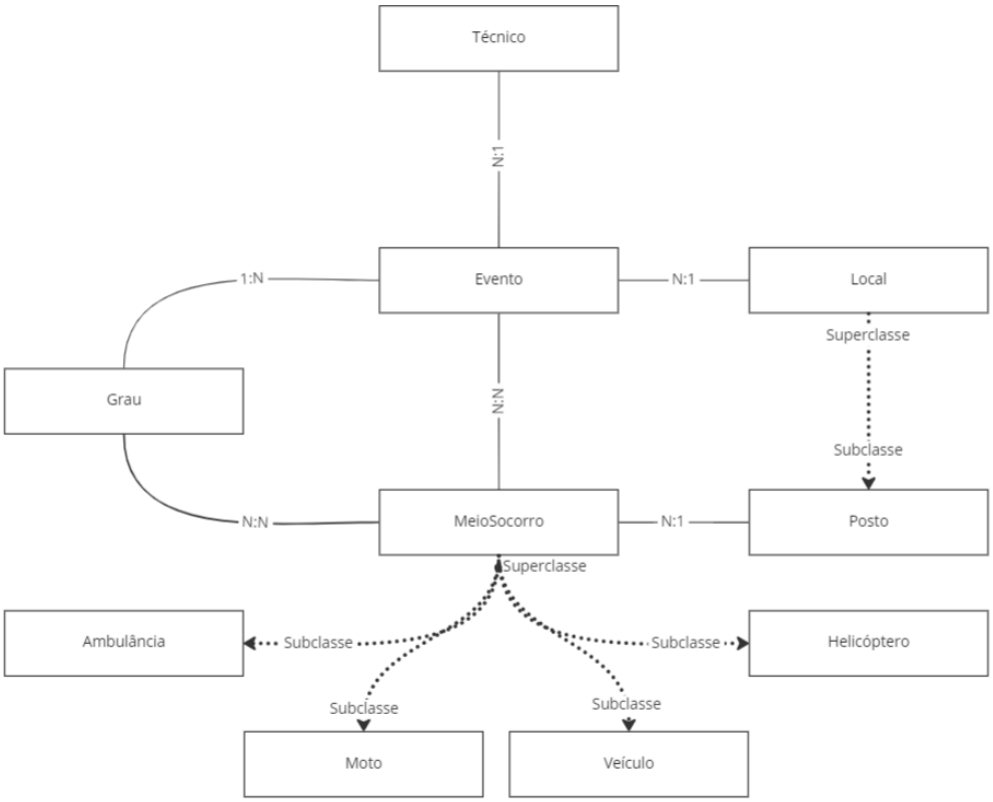

Figura 2 – Diagrama de relações entre classes.

Este planeamento é baseado, naturalmente, em alguns pressupostos, como:

- O tempo de atendimento do técnico é nulo.
- Cada evento apenas apresenta 1 vítima.
- Cada evento necessita de apenas um meio de socorro capaz de transportar a vítima (ambulância ou helicóptero) ou então um meio de transporte e outro de assistência (moto ou veículo).
- A viagem do meio de socorro até ao local de ocorrência é feito em linha reta, e consequentemente, a distância percorrida por este corresponde a 2 vezes a distância entre esse local e o posto.
- Os postos têm sempre capacidade para a vítima, visto que o meio de socorro responsável pelo transporte retorna sempre a vítima para o seu próprio posto.
- O helicóptero consegue pousar para responder a eventos em qualquer local.

Atributos

Cada uma das classes necessita de um atributo chave, que será único para cada uma das instâncias que lhes são determinadas. Numa análise mais detalhada para os atributos de cada classe, tem-se:

Técnico

- ID: inteiro, usado como chave primária e definido com AUTOINCREMENT;
- nome: *string* com o nome do técnico;
- dataNascimento: para efeitos administrativos;
- atendimentosEfetuados: lista com os id's dos eventos atendidos pelo técnico, para efeitos estatísticos;
- totalMeiosEnviados: *hashmap* com chaves em formato *string* que identificam o tipo de meio e objeto do tipo inteiro, com o número total de meios desse tipo mobilizados por este técnico, para efeitos estatísticos.

Evento

- ID: inteiro, usado como chave primária e definido com AUTOINCREMENT;
- nomeDoente: *string* com o nome da vítima;
- idadeDoente: inteiro com a informação da idade da vítima (geralmente, numa emergência médica é perguntada a idade e não a data de nascimento);
- timestampAtiv: atributo que estará sempre associado à criação de um evento, com informação da data e hora da sua ativação;
- técnico: chave estrangeira relativa ao id do técnico responsável pelo atendimento do evento;
- localOcorrencia: atributo que vai buscar informação sobre a sua estrutura à classe Local, que será tratada mais tarde (trata-se, basicamente, de um sistema de coordenadas);

- grau: atributo que vai buscar informação sobre a sua estrutura à classe Grau, que será tratada mais tarde (dá a informação sobre o estado do doente com base no tipo de suporte que é necessário prestar, para melhor identificar os meios a ativar);
- meios_mob: *hashmap* com chaves em formato inteiro, referentes ao id do meio que terá sido mobilizado nesse evento, e objetos igualmente do tipo inteiro, com a informação dos quilómetros percorridos por esse meio neste evento;
- timestampConcl: atributo que estará sempre associado à conclusão de um evento, com informação da data e hora do seu término, ou seja, do momento em que a vítima chega ao local capaz de a tratar.

MeioSocorro

- ID: inteiro, usado como chave primária e definido com AUTOINCREMENT;
- localOrigem: atributo que vai buscar informação sobre a sua estrutura à subclasse (da classe Local) Posto, referente à posição de estacionamento do meio quando está inativo. Servirá para calcular a distância percorrida pelo meio em cada evento e para apoiar na decisão de quais meios ativar em cada evento específico, baseando-se na maior proximidade a este.
- estado: *boolean* referente ao estado atual do meio, ou seja, se se encontra disponível/desmobilizado (true) ou em missão/mobilizado (false);
- eventosAtendidos: lista de inteiros que inclui os ID's (chaves) de todos os eventos atendidos pelo meio em questão;
- evento_atual: inteiro, equivalente ao ID (chave) do evento que o meio se encontra a atender;
- grau: lista de objetos da classe Grau, que definem os tipos de eventos que o meio está capacitado para responder;
- totalMobilizações: inteiro com informação do número total de mobilizações já efetuadas por este meio, para efeitos estatísticos;
- totalDistância: inteiro com informação da distância total percorrida pelo meio até ao momento em mobilizações, para efeitos estatísticos.

Para a classe MeioSocorro, foram criadas quatro subclasses que definem os quatro tipos de meios de socorro disponíveis no CODU (ambulância, moto, veículo e helicóptero). Nestes, é definido o grau/graus para os quais esse meio em específico está orientado, através do atributo *grau* que vai buscar a sua forma à classe Grau.

Grau

A classe Grau é do tipo *enum* e define os quatro graus possíveis para descrever a gravidade de um evento ou as orientações de cada tipo de meio. Os quatro níveis foram criados com base no suporte que é necessário prestar à vítima (SBV – suporte básico de vida; SAV – suporte avançado de vida) e a urgência no resgate.

Definiu-se ainda os meios mais indicados para responder a eventos de cada tipo grau possível, bem como também regras a aplicar para o caso destes não estarem disponíveis, ilustradas na Figura 3.

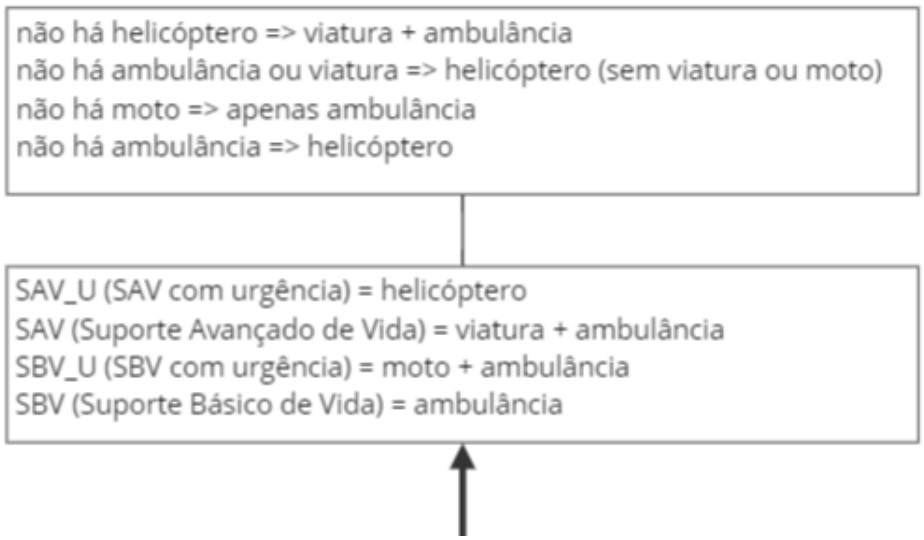

Figura 3 - Descrição completa do significado dos graus, associação aos tipos de meio e regras a aplicar na indisponibilidade de algum tipo de meio

Local e Posto

A classe Local contém dois atributos, *x* e *y*, inteiros, que serão usados como um sistema de coordenadas para calcular as distâncias entre os postos dos meios de socorro e os eventos. Para esta classe foi criada a subclasse Posto, para representar o local em que os meios estão estacionados quando inativos. Esta usará igualmente o "sistema de coordenadas" *x* e *y*, acrescentando os atributos:

- descrição: *string* com o nome do posto e que servirá de *primary key;*
- total_mob: inteiro com informação de todas as mobilizações de meios feitas a partir daquele posto.

Métodos

No que diz respeito aos métodos, todas as classes incluem os respetivos *get()'s*, *set()'s* e *toString()*. Estes são responsáveis por devolver um dado atributo de uma dada instância da classe, definir um desses atributos para uma instância e para devolver toda a informação de uma instância no formato *string*, respetivamente.

Para além destes, faz-se agora uma descrição de todos os outros métodos, organizados pelas classes a que estão subordinados:

Técnico

- *+addAtendimento(int id)*: quando um evento é criado, o seu ID é adicionado à lista de atendimentos efetuados pelo técnico que atendeu o pedido (atributo atendimentosEfetuados);
- *+addTotalMeiosEnviados(ArrayList<String> meiosEnviados)*: quando há a criação de um evento e são mobilizados meios, este método é responsável por atualizar o hashmap totalMeiosEnviados do técnico que atendeu ao pedido, identificando o tipo de meio enviado (chave) e acrescentando uma unidade (ou mais) ao objeto inteiro correspondente ao total. Este método é invocado dentro do método *mobilizaMeios(int id_evento, HashMap<Integer,Integer> meiosDisponiveisProximos)* da classe GestorCODU que será descrito mais adiante. Neste momento importa apenas notar que como argumento é passado o *array* com os tipos de meios enviados em formato *string.*

Os métodos seguintes são os que estão dependentes da classe Técnico, mas que se encontram definidos na classe GestorCODU e declarados no GestorInterface, sendo, portanto, aqueles que poderão ser invocados pelo cliente:

- +*criaTecnico(String nome)*: cria uma nova instância da classe Técnico, necessitando como argumento apenas do nome do técnico que se quer adicionar, sendo que o ID é gerado automaticamente por AUTOINCREMENT;
- +*procuraTecnico(String nome)*: com o nome de técnico que se pretende procurar passado como argumento, devolve, numa lista, os id's de técnicos cujo nome contém essa *string*;
- +*editDNTecnico(int id, LocalDate dn)*: permite alterar a data de nascimento do técnico com o id passado como 1º argumento, sendo o 2º argumento a data para a qual se pretende alterar;
- +*imprimeTecnico(int id)*: fazendo uso do método *toString()*, devolve toda a informação do técnico com o ID passado como argumento;
- +*idTecnicos()*: retorna uma lista com os ID's de todos os técnicos registados.

MeioSocorro

- +*addEventosAtendidos(int id)*: adiciona à lista de eventos atendidos por este meio (eventosAtendidos) o evento com o ID passado como argumento;
- +*add_mobilizacao()*: responsável por aumentar em uma unidade o número total de mobilizações realizadas pelo meio em questão (total_mobilizacoes);
- +*add_distancia(int distancia)*: adiciona a distância passada como atributo e relativa a um novo evento ao total de distância percorrida pelo meio (total_distancia).

Os métodos que poderão ser invocados pelo cliente são:

- +*alteraPosto(int id, String descricao)*: dados o ID do meio e a descrição do posto (key) para o qual se pretende alterar o meio, verifica se esse meio e posto e existem e procede à alteração;
- +*procuraMeioPosto(String descricao)*: dada a descrição do posto no qual se pretende procurar, devolve uma lista com os ID's dos meios associados a esse posto;

- +*procuraMeioTipo(String tipo)*: devolve uma lista com os ID's dos meios do tipo passado como argumento;
- +*procuraMeioLivre()*: retorna um *array* com os ID's de todos os meios que se encontram disponíveis (estado=True);
- +*imprimeMeio(int id)*: dado o ID do meio, devolve toda a informação disponível do meio com recurso ao método *toString()*;
- +*idMeios()*: devolve uma lista com os ID's de todos os meios registados no sistema.

Fazendo uso dos três métodos de procura de meio explicados acima, criaram-se quatro outros métodos de procura que são combinações destes:

- +*procuraMeioTipoLivre(String tipo)*: dado o tipo de meio que se quer procurar, devolve uma lista com os ID's dos meios desse tipo que se encontram disponíveis;
- +*procuraMeioPostoLivre(String descricao)*: dado o posto em que se pretende procurar, devolve uma lista com os ID's dos meios desse posto que se encontram disponíveis;
- +*procuraMeioPostoTipo(String descricao, String tipo)*: como argumentos devem ser dados a descrição do posto e o tipo de meio, ambos em formato *string*. É devolvida uma lista com os ID's dos meios do tipo pretendido e subordinados a esse posto;
- +*procuraMeioLivrePostoTipo(String descricao, String tipo)*: faz o mesmo do referido acima, mas devolve apenas os ID's dos meios disponíveis.

Em cada subclasse correspondente a cada tipo de meio, existe o respetivo método de criação de um meio desse tipo. Por exemplo, na subclasse Ambulância existe o método *+criaAmbulancia(String descricao)*, no qual se deve passar como argumento a descrição do posto a que se pretende associar a nova ambulância. O ID é obtido por AUTOINCREMENT.

Evento

Todos estes métodos encontram-se definidos na classe GestorCODU e declarados no GestorInterface, sendo que serão todos passíveis de ser invocados pelo cliente:

- *+novoEvento(int tecnico, int x, int y, Grau grau)*: cria um novo evento, necessitando do ID do técnico que atende, as coordenadas da ocorrência e o grau respetivo.
- *+novoEventoAleatorio()*: método criado para efeitos de testagem. De entre os técnicos registados no sistema e dos graus, sorteia um de cada e gera um local aleatórios com recurso ao método *+localAletatorio()*. Com estas informações, cria um evento usando o método *+novoEvento(int tecnico, int x, int y, Grau grau).*
- *+meioMaisProximo(int id, ArrayList<Integer> lista)*: dado o ID do evento, obtém-se a sua localização. O segundo argumento é uma lista de ID's de meios (normalmente serão sempre os meios do tipo pretendido e que se encontram disponíveis). Calcula-se a distância entre o local da ocorrência e o posto de cada um destes meios com recurso ao método *+distancia(Local l1, Local l2)* e retira-se o ID do meio para o qual esta é mínima. Devolve-se uma lista com duas entradas: ID do meio e distância que o separa do evento;
- *+procuraMeiosDisponiveis(int id)*: cria um *hashmap* com o ID dos meios como chave e a distância que os separa do evento como objeto. Faz uso dos métodos *+procuraMeioTipoLivre()* e *+meioMaisProximo()* e das regras de substituição de meio referidas anteriormente para selecionar os meios mais adequados e próximos para o evento;
- *+mobilizaMeios(int id_evento, HashMap<Integer,Integer> meiosproximos)*: dados o ID do evento e o *hashmap* dos meios selecionados retornado pelo método *+procuraMeiosDisponiveis()*, este método é responsável por, para cada um desses meios, alterar o estado para *false* (mobilizado), adicionar uma unidade ao total de mobilizações do meio, adicionar uma unidade ao total de mobilizações a partir do posto do meio e atualizar o atributo evento_atual do meio com o id do evento. Isto tudo, com recurso aos métodos da classe MeioSocorro *setEstado(), add_mobilizacao(), getLocalOrigem(), addTotal_mob()* e *setEvento_atual()*;
- *+desmobilizaMeio(int id)*: faz o contrário do método anterior, verificando primeiro se o meio que se pretende desmobilizar se encontrava de facto mobilizado. Procede ainda à adição da distância percorrida pelo meio neste evento ao total_distancia usando o método *+add_distancia()* e, caso este meio se trate de um meio de transporte (ambulância ou helicóptero), adiciona o timestampConcl do momento em que é invocado;

- +*adicionaInfoEvento(int id, String nome, int idade)*: permite a adição de informação adicional (nome e idade do doente) ao evento, posterior ao momento da sua criação, já que muitas vezes não é possível registar estes dados no momento da chamada urgente. É necessário indicar o ID do evento ao qual se quer adicionar a informação, sendo verificada a existência deste;
- +*imprimeEvento(int id)*: dado o ID do evento, devolve toda a informação disponível sobre este com recurso ao método *toString()*;
- +*idEventos()*: devolve uma lista com os ID's de todos os eventos registados no sistema.

Grau

- +*getNumVal()*: os diferentes graus foram definidos por *enum*, pelo que se adicionou a cada um valor numérico. Este método devolve o valor numérico associado a cada tipo de grau;
- +*randomGrau()*: retorna um valor de grau aleatório de entre os definidos. Utilizado no método +*novoEventoAleatorio()*.

Local

- +*localAletatorio()*: gera coordenadas *x* e *y* aleatórias, para serem usadas no método +*novoEventoAleatorio()*;
- +*distancia(Local l1, Local l2)*: calcula a distância entre dois locais na matriz.

Posto

- +*addTotal_mob()*: adiciona uma unidade ao total de mobilizações feitas a partir do posto em questão (total_mob).

Para serem usados pelo cliente, foram criados os seguintes métodos relativos a esta subclasse:

- +*criaPosto(String descricao, int x, int y)*: permite a criação de um novo posto, sendo indicadas a descrição e a localização (atributos *x* e *y*). É sempre verificado se já existe algum posto com a mesma descrição (atributo chave);
- +*imprimePosto(String descricao)*: dada a descrição do posto, devolve toda a informação disponível sobre este com recurso ao método *toString()*;
- +*procuraPosto(String descricao)*: devolve uma lista com as descrições de todos os postos que incluam a *string* passada como argumento;
- +*descricaoPostos()*: devolve uma lista com descrições de todos os postos registados no sistema.

Interface

A interface GestorInterfaces foi usada para registar as exceções das diferentes classes utilizadas. No caso de métodos que tinham como objetivo a criação de novas entradas, foi nas diversas exceções verificando se não existiam entradas que já tivessem um determinado ID que seja considerado único. Por exemplo, não podem existir técnicos nem meios com o mesmo ID para evitar confusão, sendo, portanto, a exceção que trata disto a *JaExisteException*.

Também foram criadas exceções para o caso de um método tentar editar ou procurar algum ID não existente, esta por sua vez denominada *NaoExisteException*, uma vez que, como será de prever, não faz sentido tentar editar ou imprimir informação referente a algo que não existe.

É de referir que tanto a *JaExisteException* como a *NaoExisteException* foram duas classes que tiveram de ser criadas, uma vez que não fazem parte das exceções já contempladas nas bibliotecas que utilizamos.

Por último, existem ainda as exceções *RemoteException* e *InterruptedException*. A *InterruptedException* é utilizada para o caso de um processo ter de ser interrompido permitindo assim parar imediatamente o processo quando este tipo de erros ocorre. Já a *RemoteException,* é uma superclasse comum referente a diversos problemas de comunicação que possam ocorrer quando métodos remotos são utilizados. Com a combinação de todas estas exceções consegue-se parar erros quando estes se formam,

impedindo assim que haja conflitos, informação perdida, ou melhorando a eficiência ao nem permitir a execução de métodos a que estes estão ligados.

Conclusão

Analisando o funcionamento final do sistema criado, verifica-se que está muito longe de ser capaz de responder a necessidades de uma aplicação real. Para tal, seria necessário alterar e até criar algumas funcionalidades adicionais, como:

- Local: Mudar o sistema de duas dimensões em que os meios podem viajar em linha reta até onde quiserem para um mapa real, com estradas e caminhos que cada veículo pode percorrer, bem como os locais em que os helicópteros podem pousar para prestar auxílio (e neste caso, seria necessária uma ambulância para transportar a vítima desde o local da ocorrência ao local que o helicóptero pode pousar).
- Moto: Por se tratar de um meio vocacionado para o trânsito citadino, será necessário delimitar a área de atuação deste, tendo em conta não só a área das cidades, mas também as condições do trânsito no momento, se possível (pois este meio é mais importante quanto maior o volume de trânsito, como nas horas de ponta).
- Helicóptero: Enquanto a utilização de veículos terrestres é quantificada através da distância que este percorre, os veículos aéreos utilizam o tempo de voo e a quantidade de ciclos (1 ciclo equivale a 1 decolagem e 1 pouso) para gerir o seu tempo de vida e manutenções. Deste modo, seria mais adequado trocar a distância percorrida deste meio para tempo de voo e número de ciclos.
- Posto: Melhor descrever as competências destes locais. Por exemplo, uma delegação da Cruz Vermelha, tratando-se de uma estação de primeiros socorros, não tem a mesma capacidade de resposta que um hospital, que pode realizar operações.
- Meio de Socorro: Dar a capacidade de um meio de socorro transportar uma vítima para um Posto diferente do seu, ou porque está mais próximo do local de ocorrência, ou porque o seu não tem capacidade para prestar o auxílio que a vítima necessita.
- Vítimas: Uma classe com uma relação N-N com a classe Evento, ou seja, poder fazer com que os eventos apresentem mais que uma vítima, com a possibilidade de diferentes vítimas apresentarem diferentes Graus no mesmo Evento, sendo para isso necessário enviar vários meios, potencialmente do mesmo tipo.

- Técnico: Consideramos que o tempo que o técnico demora a recolher informação da chamada telefónica é nula, porém, como é óbvio, esta demora tempo. Assim, seria também necessário ter em conta a disponibilidade do técnico para atender uma nova chamada.

Por fim é de referir que apesar de terem sido feitas diversas simplificações na realização deste trabalho, a arquitetura e os meios utlizados, ainda que simplistas desempenham as suas funções adequadamente, podendo muito bem servir de arquitetura base para um projeto completamente funcional tendo em conta as melhorias acima propostas.

Referências

[1] "Meios de Emergência", INEM, Consultado a 06/11/2022, https://www.inem.pt/category/cidadaos/meios-de-emergencia/

Anexos

Anexo 1 – Diagrama de classes completo

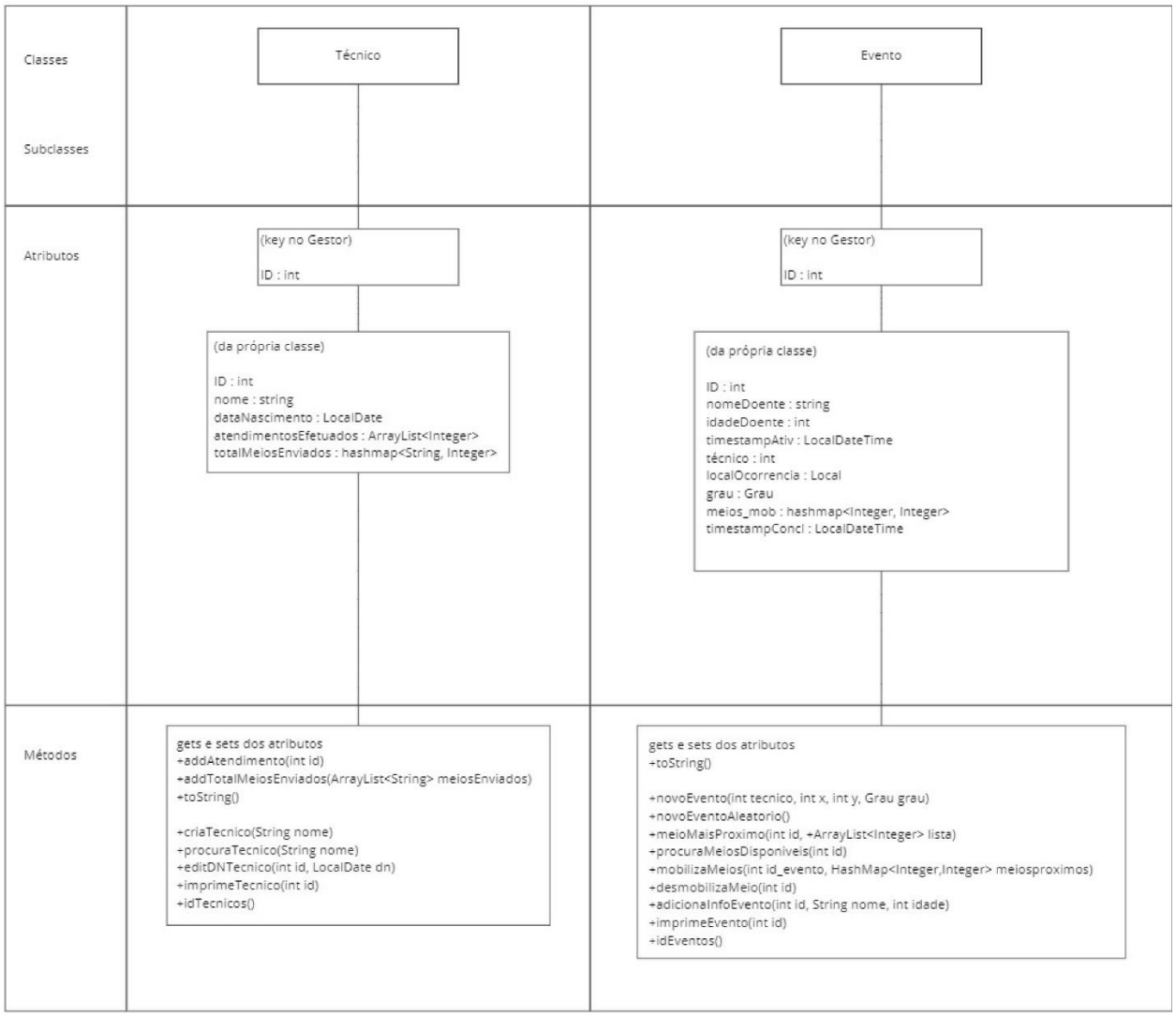

Figura 1 – Atributos e métodos das classes Tecnico e Evento.

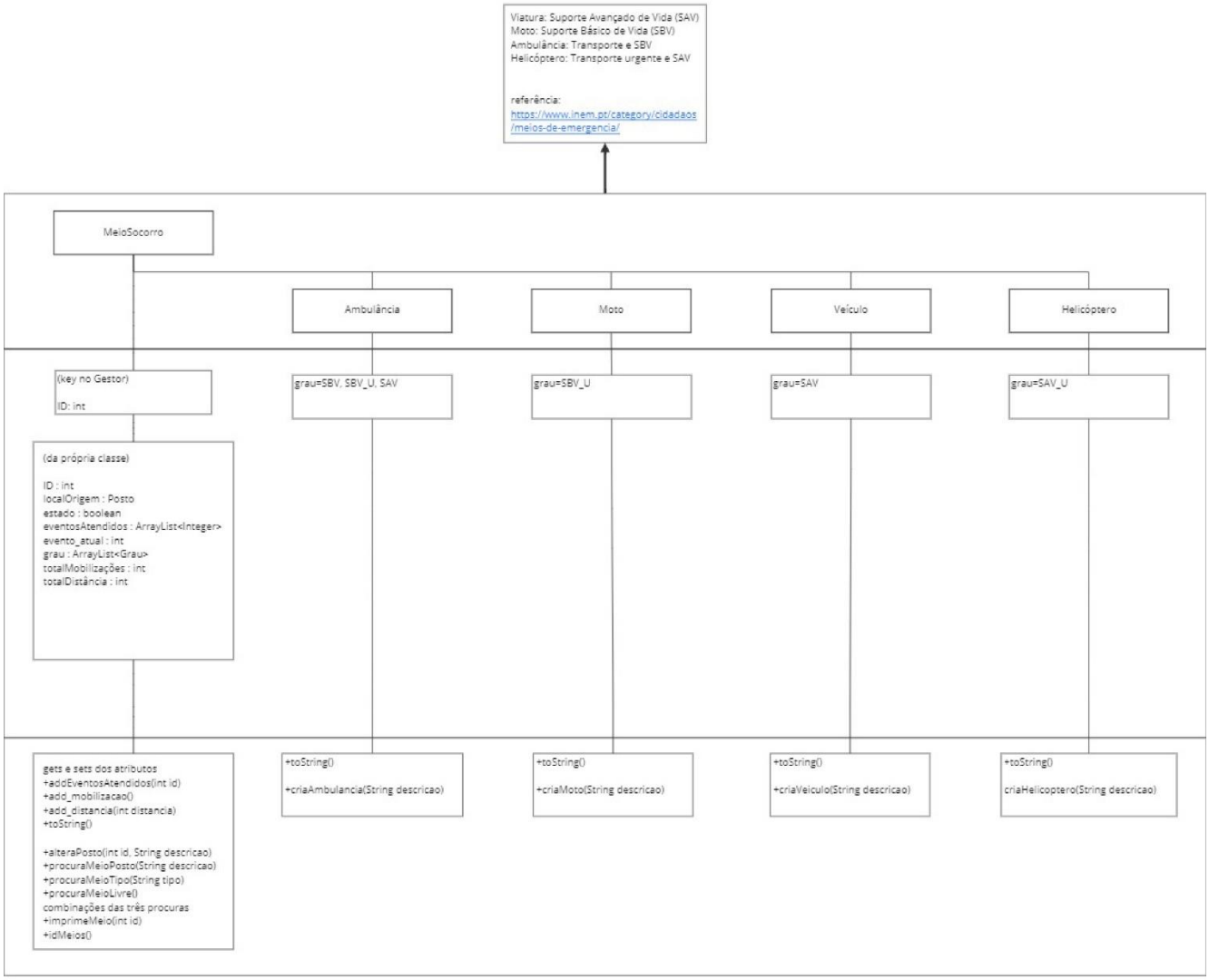

Figura 2 – Subclasses, atributos e métodos da classe MeioSocorro.

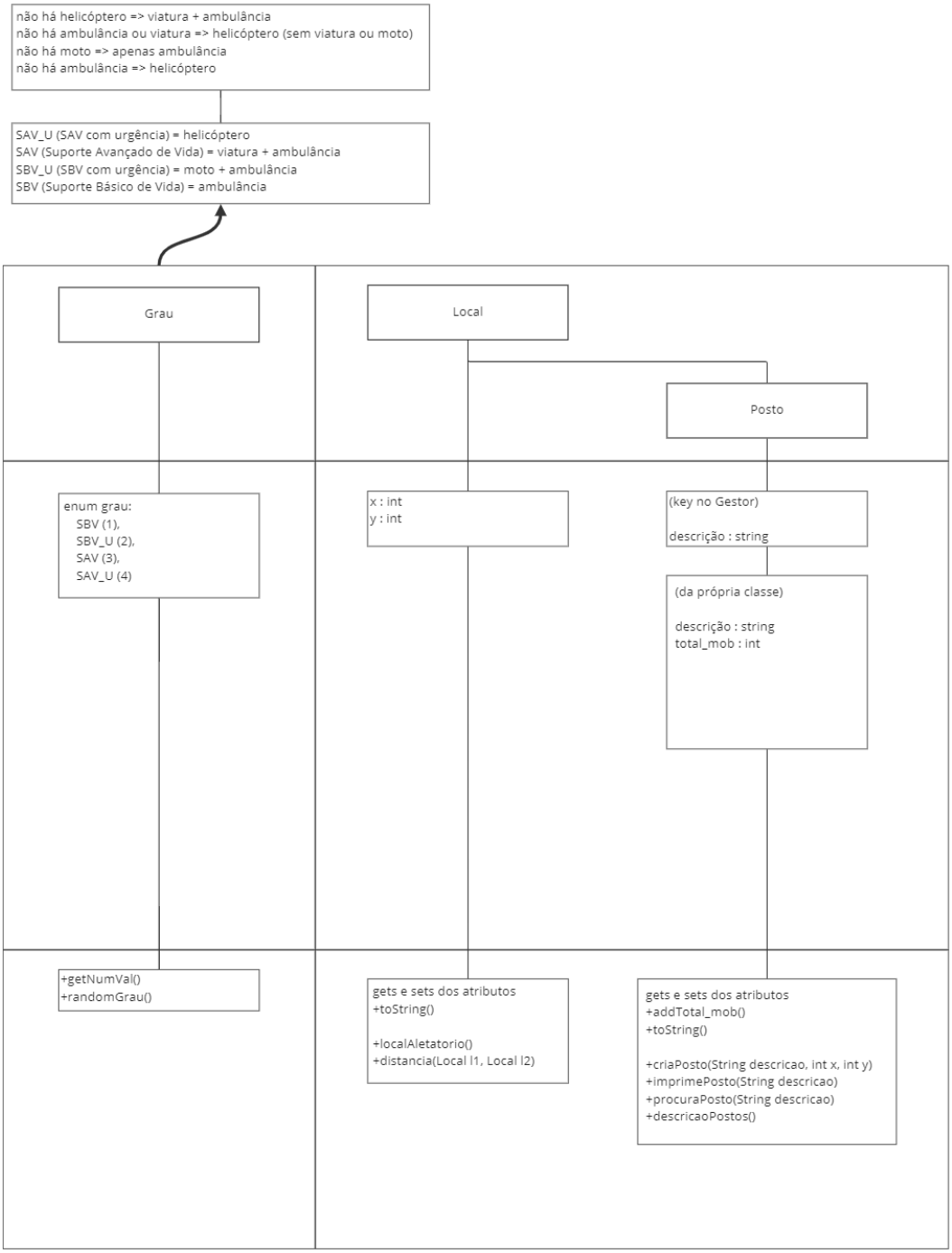

Figura 3 – Subclasses, atributos e métodos das classes Grau e Local.