Uma Plataforma Simulacional para Análise de Redes Inter-Centrais com Encaminhamento Dinâmico

Luísa Jorge
José Craveirinha
Teresa Gomes

INESC-Coimbra
Research Report ET-N9

Julho 2000

Relatório Parcial (5) relativo ao projecto Praxis/P/EEI/13219/1998 - Um estudo sobre encaminhamento dinâmico multi-objectivo e dependente do estado em redes multi-serviço.

Índice

Descrição e implementação dum simulador para redes inter-centrais com encaminhamento dinâmico

1 Introdução

Sendo as redes inter-centrais com encaminhamento dinâmico demasiado complexas para serem estudadas analiticamente através de um modelo suficientemente rigoroso, nomeadamente em termos de desempenho dinâmico (tendo em conta a natureza estocástica dos fluxos de tráfego) recorremos a um modelo simulacional para fazer o seu estudo.

Neste relatório far-se-á uma revisão dos princípios e conceitos de simulação que estão na base do desenvolvimento de uma plataforma computacional para a análise de desempenho de redes inter-centrais com comutação por circuitos, com métodos de encaminhamento dinâmico. Por outro lado, apresentam-se as características relevantes da linguagem de simulação OMNeT++, que foi utilizada, bem como uma descrição dos aspectos essenciais e módulos do pacote de software desenvolvido neste âmbito.

2 Conceitos introdutórios sobre simulação

Uma referência geral para os conceitos de simulação aqui utilizados é [Law91].

Com a simulação pretende-se obter uma representação, tão realista quanto possível, da operação ao longo do tempo de um sistema real. É necessário para o efeito que seja gerada uma realização (função amostra) do comportamento do sistema. A partir da simulação são recolhidos dados como se se estivesse a observar o sistema real. Os dados gerados pela simulação são usados para estimar as medidas de desempenho do sistema.

Segue-se a definição de alguns termos para ajudar na compreensão e análise do sistema (como apresentado em [Banks96]).

Uma *entidade* é um objecto de interesse no sistema.

Um *atributo* é uma propriedade de uma entidade.

Um *recurso* é um objecto que fornece um serviço a uma entidade

Uma *actividade* representa um período de tempo de comprimento especificado.

O *estado do sistema* é armazenado num conjunto de variáveis de estado do sistema. As variáveis de estado são as variáveis necessárias para descrever o sistema relativamente aos objectivos do estudo.

Um *acontecimento* é definido como uma ocorrência instantânea que pode alterar o estado do sistema.

Os sistemas podem ser discretos ou contínuos. Um sistema discreto é um sistema no qual as variáveis de estado mudam apenas num conjunto discreto de pontos no tempo. Um sistema contínuo é um sistema no qual as variáveis de estado mudam continuamente ao longo do tempo. Na prática poucos sistemas são totalmente discretos ou contínuos, mas como existe sempre um tipo de mudanças que predomina para muitos sistemas, será possível normalmente classificar um sistema como contínuo ou discreto [Law91].

Os números aleatórios são o ingrediente básico na simulação de quase todos os sistemas discretos. Os números aleatórios numa linguagem de simulação são usados para gerar os instantes em que ocorrem os acontecimentos e para gerar outras variáveis aleatórias.

Uma sequência de números aleatórios deve ter as propriedades de independência e uniformidade, ou seja (muito genericamente) cada um dos números da série deve ser independente dos restantes e a probabilidade de obter qualquer dos valores no intervalo definido para esses números deve ser igual. Os números aleatórios que são usados numa linguagem de simulação são designados por números pseudo - aleatórios por serem gerados a partir de um método conhecido e reproduzível.

Na análise das saídas de uma simulação faz-se a análise dos dados gerados na simulação, com o objectivo de estimar o desempenho do sistema modelizado.

2.1 Simulação por acontecimentos discretos

A simulação por acontecimentos discretos está relacionada com a modelação da evolução do sistema ao longo do tempo por uma representação na qual as variáveis de estado mudam de valor, instantaneamente, em pontos separados no tempo. Os modelos de simulação por acontecimentos discretos têm uma forma característica de representação do tempo através de um relógio de simulação. O relógio de simulação é a designação para a variável que no modelo de simulação dá o valor corrente do tempo simulado. O avanço do relógio de simulação pode ser de um dos dois tipos seguintes:

- Para o próximo acontecimento;
- Incrementos fixos – avanço do tempo simulado por incrementos constantes.

O avanço utilizado mais frequentemente (quase exclusivamente) é o avanço do relógio de simulação para o próximo acontecimento. Neste caso o relógio de simulação avança de acontecimento em acontecimento sendo actualizado o estado do sistema em cada avanço. Este processo continua até que seja satisfeita uma condição de paragem previamente especificada.

Neste tipo de simulação existe uma lista de acontecimentos, também designada por lista de acontecimentos futuros, que é uma lista de notificações para acontecimentos futuros, ordenada pelo tempo de ocorrência desses acontecimentos.

2.2 Outros tipos de simulação

A simulação contínua relaciona-se com a modelação de um sistema, ao longo do tempo, por uma representação na qual as variáveis de estado mudam continuamente em função do tempo.

A simulação combinada contínua - discreta é usada em sistemas que não são completamente discretos nem completamente contínuos e nos quais haja necessidade de construir um modelo com aspectos de simulação contínua e também de simulação por acontecimentos discretos.

A simulação de Monte Carlo[1] segundo Law e Kelton [Law91] é um esquema que utiliza números aleatórios e que é usado para resolver certos problemas estocásticos ou determinísticos onde a representação explícita da passagem do tempo não tem grande importância. Daí que as simulações de Monte Carlo sejam geralmente estáticas e não dinâmicas.

É importante notar que um modelo de simulação discreto nem sempre é usado para modelar um sistema discreto, assim como um modelo de simulação contínuo nem sempre é usado para modelar um sistema contínuo. A escolha do tipo de simulação a usar é uma função das características do sistema e dos objectivos do estudo.

2.3 Tipos de software de simulação para redes de comunicação

Podem considerar-se três tipos principais de software de simulação para simular redes de comunicação [Law94]:

1. Linguagens de simulação de âmbito geral: pacote de simulação que pode ser utilizado em sistemas de diversas áreas e tipos, mas que tem características especiais viradas para redes de comunicação. A maior vantagem de muitas destas linguagens é a sua capacidade para modelar quase todos os tipos de redes de comunicação.
2. Linguagens de simulação orientada para comunicações: pacote de simulação orientada especificamente para o estudo de redes de telecomunicações.
3. Simulador orientado para comunicações: pacote de simulação que permite simular uma rede numa classe específica de redes de comunicação, sem necessidade de programação. É seleccionada uma rede particular para simulação (dentro do domínio do pacote) escolhendo itens dos menus, preenchendo caixas de diálogo e pelo uso de gráficos. Um exemplo de um simulador orientado para comunicações é o COMNET III.

As linguagens de simulação usam uma das duas aproximações básicas seguintes:

- Aproximação orientada para o acontecimento;
- Aproximação orientada para o processo.

---

[1] É frequente encontrar autores que definem simulação de Monte Carlo qualquer simulação que envolva o uso de números aleatórios.

Para modelar o sistema, na aproximação orientada para o acontecimento, começa-se por identificar os acontecimentos característicos e depois escreve-se um conjunto de rotinas de acontecimentos que dão uma descrição detalhada das mudanças de estado que ocorrem na altura de cada acontecimento. A simulação evolui ao longo do tempo pela execução de acontecimentos (rotinas de acontecimento), ordenados por ordem crescente do tempo da sua ocorrência. Uma propriedade básica de uma rotina de acontecimentos é que não há passagem do tempo de simulação durante a sua execução.

Na aproximação orientada para o processo, este é entendido como uma sequência ordenada pelo tempo de acontecimentos inter-relacionados, separados pela passagem do tempo, os quais descrevem a experiência completa de uma entidade, enquanto ela flui ao longo do sistema. Um sistema ou modelo de simulação pode ter vários tipos diferentes de processos. Existe uma rotina, correspondente a cada processo no modelo, que descreve a história completa da sua entidade, e a forma como ela se altera através do processo correspondente.

3 Breve descrição da linguagem de simulação utilizada - OMNeT++

O OMNeT++ é uma linguagem de simulação por acontecimentos discreta com aproximação ao processo.

O OMNeT++ assim como o OPNET são linguagens de simulação de âmbito geral (de acordo com a classificação anterior do software de simulação para redes de comunicação). Possuem um editor gráfico que pode ser usado para especificar graficamente a topologia da rede.

3.1 Abordagem geral ao OMNeT++

De acordo com a filosofia de modelação que o OMNeT++ implementa, um sistema consiste em várias entidades designadas por módulos. Um modelo em OMNeT++ consiste em vários módulos, módulos esses que comunicam entre si pela passagem de mensagens. Os módulos activos são designados por módulos simples. O código dos módulos simples executa-se quase em paralelo (os módulos simples são implementados como co-rotinas – processos concorrentes). O código dos módulos simples é escrito em C++ usando a biblioteca de classes do OMNeT++. As mensagens podem conter estruturas de dados complexas.

Os módulos simples podem ser agrupados em módulos designados por módulos compostos.

Existe uma linguagem separada para definir a estrutura do modelo (linguagem NED - "**NE**twork **D**escription"). Em vez de usar a linguagem directamente pode usar-se um editor gráfico, GNED - "**G**raphical **N**etwork **ED**itor", que produz ficheiros NED a partir da representação gráfica. A descrição do modelo em linguagem NED é traduzida em C++ (usando o compilador NEDC) sendo adicionada ao código da simulação.

Num ficheiro de configuração, designado por ficheiro *ini* (por normalmente ter extensão *ini*) são inseridas opções que controlam como a simulação é executada e valores para os parâmetros do modelo. Este ficheiro é um ficheiro de texto, consistindo em entradas agrupadas em diferentes secções. Neste ficheiro podem ser seleccionadas as sementes para os geradores de números aleatórios. O OMNeT++ tem instrumentos para ajudar na escolha de boas sementes.

Para tornar uma simulação executável é necessário ligar o código produzido (código C++ correspondente à implementação dos módulos simples e à descrição do modelo) ao núcleo (*kernel*) do OMNeT++ e a um dos interfaces com o utilizador oferecido pelo OMNeT++.

Todas as corridas incluídas no ficheiro de configuração são usualmente executadas automaticamente umas após as outras. As opções que controlam como a simulação é executada e os valores para os parâmetros do modelo podem ser especificados globalmente ou individualmente para cada corrida. Os resultados da simulação são escritos em ficheiros de saída.

3.2 Linguagem utilizada para definir o modelo - Linguagem NED

A descrição que vamos passar a fazer apenas tem como objectivo servir de base para a compreensão do modelo criado. Para uma descrição completa da linguagem consultar o Manual do Utilizador do OMNeT++ [Varga00].

A topologia do modelo pode ser representada usando o editor gráfico GNED, resultando uma descrição do modelo em linguagem NED.

Um modelo OMNeT++, normalmente designado por rede, consiste em módulos embutidos hierarquicamente. O módulo no nível topo é designado por módulo de sistema que pode conter submódulos os quais por sua vez podem conter outros submódulos. Os módulos que possuem submódulos são designados por módulos compostos, e os módulos que estão no nível mais baixo de hierarquia são designados por módulos simples. Não existindo limite para a profundidade dos módulos, é permitido ao utilizador reflectir a estrutura lógica do sistema na estrutura do modelo. Os módulos comunicam com a passagem de mensagens, mensagens estas que podem conter estruturas de dados complexas. Os módulos podem enviar mensagens directamente para os destinos ou por um caminho predefinido, através de portas e ligações. As portas são os interfaces de entrada e saída dos módulos, sendo as mensagens enviadas através de uma porta de saída e recebidas através de uma porta de entrada. Podem ser criadas ligações entre portas correspondentes de dois submódulos, ou entre a porta de um submódulo e a porta correspondente do módulo composto. Os módulos podem ter parâmetros que são usados principalmente para três fins:

- Personalizar o comportamento dos módulos;

- Criar topologias flexíveis para o modelo;
- Para comunicação entre os módulos, como variáveis partilhadas.

Os parâmetros podem ser usados para construir topologias num modo flexível. Assim, dentro de um módulo composto, os parâmetros podem definir o número de submódulos, o número de portas e o modo como as ligações internas são feitas. Os módulos compostos podem passar parâmetros para os seus submódulos.

A definição da rede (ou definição do módulo de sistema) especifica o módulo sistema. O modelo fica completo quando se define o módulo de sistema. Exemplos das definições de todos os tipos de módulos podem ser obtidos no Manual do Utilizador do OMNeT++ [Varga00].

3.3 Desenvolvimento dos módulos simples – Programação dos algoritmos.

Os módulos do nível hierárquico mais baixo são implementados pelo utilizador, e contêm os algoritmos do modelo. Durante a execução de uma simulação, os módulos simples parecem correr em paralelo, pois são implementados como co-rotinas (por vezes designadas como processos). Os módulos simples são implementados em C++.

Todos os elementos da simulação (mensagens, módulos, parâmetros, etc) são representados por objectos. Alguns exemplos de classes da biblioteca das classes do OMNeT++ são: *cModule*, *cGate*, *cPar*, *cMessage*.

Cada módulo simples é implementado com uma classe C++. As classes dos módulos simples são derivadas de uma classe base (*cSimpleModule*), redefinindo uma função virtual *activity* com o algoritmo do processo. Uma outra função membro da classe *cModule* é a função *finish*() que é executada quando uma simulação termina. A função *finish*() também pode ser redefinida nas classes dos módulos simples. É possível acrescentar outras funções membro à classe ou mesmo membros de dados (variáveis internas).

Os parâmetros dos módulos simples podem ser acessíveis pelos seus algoritmos.

As funções e objectos mais importantes que podem ser usados na implementação dos módulos simples encontram-se referidos em [Varga00]. Para compreensão do código implementado é necessário o conhecimento destes componentes. Para uma descrição mais pormenorizada deve ser consultado o Apêndice D do Manual do Utilizador do OMNeT++ [Varga00].

3.4 Geração de números aleatórios

O gerador de números aleatórios utilizado no OMNeT++ é um gerador congruencial linear ("Linear Congruential Generator" - LCG) com um comprimento cíclico de $2^{31} - 2$.

Se um programa de simulação usar números aleatórios para mais do que um fim, então esses números deverão vir de geradores de números aleatórios diferentes. O OMNeT++ tem vários geradores de números aleatórios independentes.

Para evitar correlações indesejadas é importante que sejam usadas séries não sobrepostas de números aleatórios nas várias corridas de simulação e nas várias fontes de números aleatórios dentro de uma corrida de simulação. Por esta razão, os geradores devem começar com sementes bem separadas.

3.5 Compilar e executar uma simulação

Para executar uma simulação é necessário ligar o código produzido (código C++ correspondente à implementação dos módulos simples e à descrição do modelo) ao núcleo (*kernel*) do OMNeT++ e a um dos interfaces com o utilizador oferecido pelo OMNeT++. Os interfaces com o utilizador oferecidos pelo OMNeT++ são o Cmdenv, o Tvenv e o Tkenv os quais vão ser descritos na próxima secção.

Um modelo OMNeT++ consiste fisicamente nas seguintes partes:

- Na descrição da topologia na linguagem NED. Ficheiro com extensão *ned*.
- Nos módulos simples. Aos módulos simples correspondem ficheiros C++, onde se definem as operações dos componentes do modelo.

Os ficheiros NED são compilados em C++ usando o compilador NEDC que faz parte do OMNeT++.

O sistema de simulação fornece os seguintes componentes que fazem parte de qualquer simulação executável:

- O núcleo de simulação com a biblioteca de classes de simulação;
- Interface com o utilizador.

Os programas de simulação são construídos a partir dos componentes anteriores. Primeiro os ficheiros NED são compilados em código fonte C++ usando o compilador NEDC. Depois todos os ficheiros fonte C++ (implementação dos módulos simples e descrição da topologia) são compilados e ligados ao núcleo de simulação e ao interface do utilizador para construir uma simulação executável.

Como já foi referido, devem ser inseridas, num ficheiro de configuração designado por ficheiro *ini*, as opções que controlam como a simulação é executada e valores para os parâmetros do modelo. É sobre esse ficheiro que se vai falar mais em pormenor já a seguir.

3.5.1 Ficheiro de configuração

Os ficheiros de configuração (também chamados ficheiros *ini*, por terem normalmente a extensão *ini*) contêm opções que controlam como a simulação é executada e podem também conter inicializações dos parâmetros do modelo. Neste ficheiro podem ser seleccionadas as sementes para os geradores de números aleatórios (o OMNeT++ tem instrumentos para ajudar na escolha de boas sementes). O ficheiro *ini* é um ficheiro de texto que consiste em várias entradas agrupadas em várias secções. As secções que podem existir são as seguintes:

1. Se se pretender fazer inicializações comuns a todas as corridas[2]:
   - [*General*] – inicializações gerais;
   - [*Cmdenv*], [*Tvenv*], [*Tkenv*], . . . – inicializações específicas do interface com o utilizador;
   - [*Parameters*] – valores dos parâmetros dos módulos;
2. Se se pretender fazer inicializações para cada corrida individualmente:
   - [Run 1], [Run 2], [Run 3], . . . - inicializações específicas do interface com o utilizador, valores dos parâmetros dos módulos ou mapeamento das máquinas lógico-fisico.

Nestas secções, os caracteres # e ; são usados para colocar comentários. Os valores dos parâmetros dos módulos, caso não tenham sido inicializados previamente no ficheiro NED, são pesquisados no ficheiro *ini*. Caso não sejam encontrados são pedidos valores ao utilizador.

O OMNeT++ pode executar várias corridas de simulação automaticamente uma depois das outras. Se foram seleccionadas várias corridas em sucessão, as opções de inicialização e o valor dos parâmetros podem ser dados individualmente para cada corrida ou conjuntamente para todas as corridas.

3.6 Interfaces com o utilizador

Os interfaces com o utilizador do OMNeT++ são usados na execução de uma simulação. O seu principal fim é tornar o interior do modelo visível ao utilizador, iniciar ou parar a execução da simulação e, eventualmente permitir a intervenção do utilizador pela alteração de objectos e variáveis dentro do modelo. Isto é muito importante na fase de desenvolvimento e correcção do projecto de simulação.

No OMNeT++ o interface com o utilizador está separado do núcleo de simulação. Isto faz com que seja possível usar vários tipos de interfaces com o utilizador sem alterar o núcleo de simulação.

---

[2] Para além dos apresentados, existem duas secções comuns que só têm significado na execução distribuída do simulador:
- [*Slaves*] – opções referentes a cada unidade lógica de processamento distribuído;
- [*Machines*] – mapeamento lógico - fisico das máquinas, na execução distribuída, indicando quais processos vão correr em quais máquinas;

É também possível executar o mesmo modelo de simulação com diferentes interfaces com o utilizador sem qualquer mudança nos ficheiros do modelo. Desta forma, na fase de teste e correcção, pode usar-se um poderoso interface gráfico, e no final executar a simulação com um interface simples e rápido, que suporte a execução em lote.

Os interfaces com o utilizador disponíveis no OMNeT++ são:

- Cmdenv: interface com o utilizador em linha de comando, para execução em lote;
- Tvenv: interacção textual, interface com o utilizador em janelas (DOS, Turbo Vision);
- Tkenv: interacção gráfica baseada no Tk[3], interface em janelas (X-Window, Win95, WinNT, etc.).

Segue-se uma descrição mais detalhada acerca de cada um dos interfaces com o utilizador.

Cmdenv

O interface com o utilizador em linha de comando é um interface pequeno, portável e rápido que compila e executa em todas as plataformas (UNIX; DOS; ou WinNT). O interface Cmdenv foi projectado principalmente para execuções em lotes.

O Cmdenv executa todas as corridas de simulação que forem descritas no ficheiro de configuração.

Tvenv[4]

No interface com o utilizador Tvenv a interacção é textual, o interface com o utilizador em janelas suporta a execução interactiva da simulação. Tvenv é recomendado para ser usado na fase de desenvolvimento da simulação ou para apresentações, pois permite obter uma imagem detalhada do estado da simulação em qualquer ponto da execução e seguir o que acontece dentro da rede.

Tkenv

O interface Tkenv é um interface gráfico, portável que utiliza janelas. O Tkenv suporta a visualização da simulação como o Tvenv. O Tkenv também suporta a visualização dos resultados de simulação durante a execução. Os resultados podem ser mostrados como histogramas ou séries temporais. Isto pode acelerar o processo da operação de verificação e correcção do programa de simulação e fornece um bom ambiente para experiências com o modelo durante a execução.

---

[3] Tk é um toolkit de interacção com o utilizador que fornece primitivas de interacção mais avançadas do que as normalmente presentes no sistema Unix.

[4] Este interface com o utilizador tem vindo a ser depreciado no OMNeT++, já não se encontrando presente nas iterações beta da próxima versão do sistema, actualmente disponíveis.

4 *Análise do sistema a implementar*

Dada uma estrutura específica da rede de telecomunicações pretende-se analisar o grau de serviço ("Grade Of Service" - GOS) da rede em função do método de encaminhamento de tráfego utilizado.

O nosso objectivo é comparar algumas medidas de desempenho da rede, tais como bloqueios ponto a ponto e tráfego transportado, conseguidos pelos vários métodos de encaminhamento, em situações de carga e sobrecarga.

4.1 Descrição do sistema

O sistema que se pretende estudar é uma rede de comutação por circuitos, à qual é oferecido tráfego que pode ser de vários tipos e na qual o tráfego é encaminhado de acordo com um determinado método de encaminhamento, de entre vários possíveis.

Da rede de comutação por circuitos fazem parte as centrais de comutação e os feixes que as interligam. Numa rede deste tipo quando uma chamada é estabelecida num dado caminho são ocupados canais em cada arco (feixe) desse caminho, que ficam indisponíveis para outras chamadas durante todo o tempo de serviço da chamada.

Os fluxos de tráfego oferecidos à rede são descritos pela população geradora das chamadas, pela natureza das chegadas e das terminações e pelos parâmetros associados às próprias chamadas.

A população geradora das chamadas pode ser: população de tamanho infinito e/ou população de tamanho finito.

As chamadas são geradas aleatoriamente. Uma chamada recém chegada é servida imediatamente se houver canais (ou circuitos) disponíveis (necessários para o estabelecimento da chamada), caso contrário a chamada é perdida. O tempo de serviço de cada chamada tem uma distribuição exponencial negativa. No caso das chamadas serem geradas por populações de tamanho infinito o tempo entre chegadas tem uma distribuição exponencial negativa. No caso das chamadas serem geradas por populações de tamanho finito o tempo até à próxima chegada tem uma distribuição exponencial negativa que depende do número de fontes livres na população correspondente.

Para estabelecer uma chamada é necessário um determinado número de circuitos em cada feixe, do caminho que seja utilizado pela chamada. O número de circuitos necessários (em cada feixe) depende do tipo de fluxo a que a chamada pertence, pois admite-se tráfego multiclasse com chamadas multicanal.

Os métodos de encaminhamento que estão implementados na actual versão do simulador para o encaminhamento das chamadas são: o directo, o FAR, o DAR, o DCR e o RTNR.

Podemos definir o estado da rede (conjunto de variáveis necessárias para descrever o sistema) como sendo o número de chamadas entre cada par origem/destino, o número de circuitos livres entre cada par de nós, etc. Neste sistema existem apenas dois acontecimentos que podem alterar o estado do sistema, que são o estabelecimento de uma chamada e a terminação de uma chamada.

Um sistema de Teletráfego deste tipo é um sistema discreto, apenas do ponto de vista das variáveis de estado, uma vez elas são alteradas apenas quando há a geração ou terminação de uma chamada. Do ponto de vista da variável tempo (índice) é contínuo, pois as mudanças de estado podem ocorrer em qualquer instante. Os tempos de chegada e os tempos de ocupação são variáveis aleatórias, cujos valores são obtidos a partir das distribuições a elas associadas.

Vai ser desenvolvido um modelo de simulação para imitar o comportamento do sistema ao longo do tempo. O modelo de simulação vai ser usado como instrumento para predizer o desempenho do sistema num conjunto variado de circunstâncias.

O comportamento ao longo do tempo deste tipo de sistemas pode ser descrito por uma simulação por acontecimentos discretos. O modelo envolverá também componentes aleatórios (associados à geração e terminação das chamadas). O modelo a implementar será então discreto, dinâmico e estocástico.

5 *Descrição do modelo de simulação implementado*

Foi desenvolvido um modelo que pretende representar o comportamento estocástico da rede, com o objectivo de permitir o estudo comparativo do desempenho de vários métodos de encaminhamento. O modelo construído foi um modelo de simulação por acontecimentos discretos com aproximação orientada para o processo.

Do modelo fazem parte dois tipos de processos que são o processo `gerador` e o processo `central`.

As `centrais` e os `geradores` são as entidade do modelo. As chamadas vão ser modeladas por mensagens, e as ligações entre as centrais do sistema vão ser modelados por ramos que são os recursos do nosso sistema (por exemplo a duração das chamadas é um parâmetro das mensagens). Como exemplo de variáveis de estado do nosso sistema podemos considerar o número de chamadas concluídas entre quaisquer par de centrais, o número de circuitos ocupados num ramo e o número de chamadas no sistema.

Os dois acontecimentos principais no sistema são o estabelecimento e a conclusão de uma chamada. No modelo temos vários acontecimentos. A cada acontecimento no modelo está sempre associada uma mensagem. Existem vários tipos de mensagens que podem ser divididos em dois grupos: mensagens que correspondem a chamadas e mensagens de controlo. As

mensagens que correspondem a chamadas podem ser de quatro tipos: mensagem que corresponde a uma chamada para ser estabelecida (tipo "chamada a estabelecer"), mensagem que corresponde a uma chamada que foi concluída (tipo "chamada concluída"), mensagem que corresponde a uma chamada que não foi possível estabelecer (tipo "chamada impossível") e mensagem que corresponde ao *crankback* de uma chamada (tipo "chamada crankback"). As mensagens de controlo podem ser de três tipos: mensagem de inicialização, mensagem de terminação da simulação ou mensagem para actualização da informação usada no encaminhamento das chamadas (tipo "mensagem de encaminhamento"), por exemplo actualização das tabelas de encaminhamento.

A lista de acontecimentos usada para determinar o próximo acontecimento é processada internamente pela linguagem de simulação. Essa lista contém os instantes em que os acontecimentos, de cada tipo, ocorrem, ordenada pelos valores desses instantes.

5.1 Representação do modelo na linguagem NED

A rede inter-centrais pode ser modelada em OMNeT++ como uma rede com vários módulos `central` (centrais de comutação) e vários módulos `gerador` (geradores de chamadas), tantos `geradores` como `centrais`. Estando cada `central` ligada a todas as outras `centrais` e cada `gerador` ligado à sua `central`. Os módulos comunicam pela troca de mensagens. Os módulos `geradores` geram chamadas para as `centrais` correspondentes, as `centrais` recebem chamadas que pretendem ser estabelecidas.

Usando a linguagem NED (**NE**twork **D**escription) foi definida a estrutura do modelo da rede. A partir do modelo em linguagem NED, resulta a topologia da rede de telecomunicações que vai ser estudada. Na figura 1 apresenta-se um exemplo de uma topologia resultante, tendo sido 6 centrais.

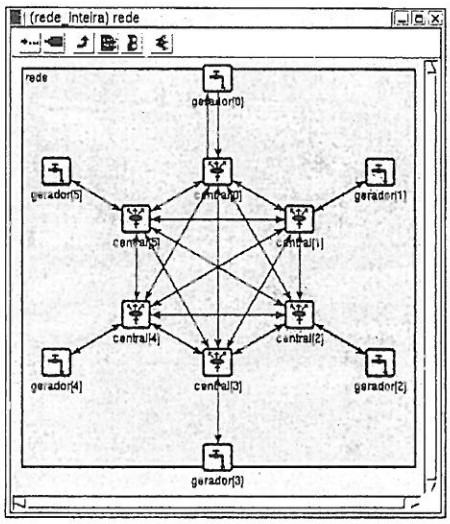

Fig. 1 – Topologia do modelo implementado (exemplo com 6 centrais).

5.2 Módulos simples implementados

O funcionamento dos módulos simples (`gerador` e `central`), foi implementado na linguagem C++ ([Stroustroup97]) utilizando a biblioteca de classes de simulação do OMNeT++. Todos os diagramas que descrevem o funcionamento do sistema, apresentados neste relatório foram desenvolvidos usando a notação UML ([Larman98]) e foram na sua maior parte criados usando a ferramenta Rational Rose ([Rational00])

Cada módulo simples é definido através de uma classe C++, derivada de uma classe chamada `cSimpleModule`, onde se redefine a função membro `activity()`. As funções `activity()` dos módulos simples na rede são executadas como co-rotinas, parecendo que são executadas em paralelo.

Vamos começar esta subsecção com uma descrição da interacção entre os processos `gerador` e os processos `central`. Lembrar que o processo `gerador` é o responsável pela criação das chamadas, chamadas essas que são trocadas entre processos `central` desde a origem até esta atingir o destino. Estas chamadas quando são enviadas causam uma inactivação da fonte que a gerou no caso de população finita, e esta fonte só volta a actividade quando recebe de volta a mensagem que enviou. No caso de populações infinitas, este mecanismo de inactivação não é necessário uma vez que se assume a existência de um número infinito de fontes independentes.

Para simplificação da descrição que se segue, sempre que pretenda referir um processo `central` que recebeu uma chamada do processo `gerador` para ser encaminhada, vamos utilizar simplesmente a designação `central` origem. Pela mesma razão vamos também usar as designações `central` intermédia e `central` destino nas situações correspondentes a um processo `central` que acabou de receber uma chamada vinda de outro processo `central` e o qual ainda não é o processo `central` para onde a chamada se destina; e a um processo `central` que acabou de receber uma chamada que lhe era destinada vinda de outro processo `central`, respectivamente.

Os processos do tipo `gerador` e `central` (a executar durante toda a simulação) são os responsáveis pela criação e encaminhamento das chamadas na rede.

A função do `gerador` é gerar mensagens, mensagens essas que correspondem a chamadas a ser estabelecidas. Quando o `gerador` gera uma mensagem, correspondente a uma chamada do tipo "chamada a estabelecer" ela é enviada para a `central` à qual está ligado. A `central` (onde a chamada surgiu - `central` origem) ao receber uma chamada determina se a capacidade livre do ramo que a liga à `central` para onde a chamada pretende ser encaminhada (`central` destino) permite o estabelecimento dessa chamada. Se a capacidade disponível o permitir a chamada a estabelecer é enviada, por esse ramo, para a `central` destino. Quando a `central` destino recebe uma chamada do tipo "chamada a estabelecer" cria uma mensagem do tipo "chamada

concluída" (para enviar a si própria) que receberá quando o tempo de ocupação da chamada terminar. Quando isso ocorrer a `central` destino envia essa mensagem para a `central` origem. A `central` origem ao receber esta mensagem envia-a para o `gerador` (apenas no caso de ela corresponder a um fluxo de tráfego gerado por uma população finita, de forma a indicar o fim da inactivação de uma dada fonte). Tudo isto caso o mecanismo de encaminhamento tenha indicado, à `central` origem, o caminho directo (caminho formado apenas pelo ramo que liga a `central` origem à `central` destino) como caminho a usar no encaminhamento da chamada. Este processo pode ser observado nos dois diagramas seguintes.

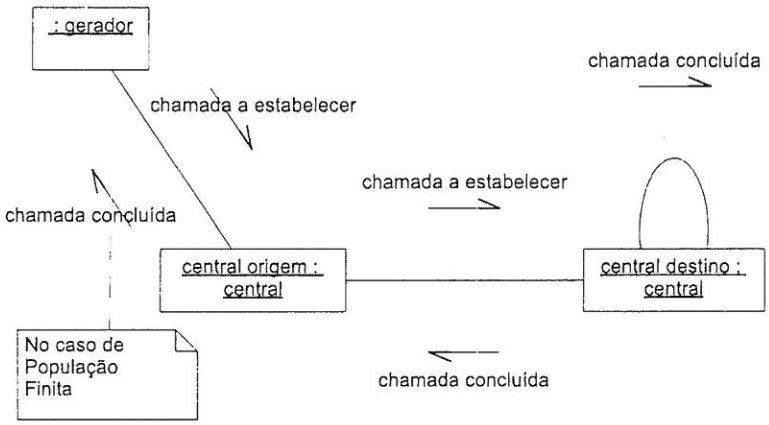

Fig. 2 - Diagrama de colaboração (encaminhamento directo).

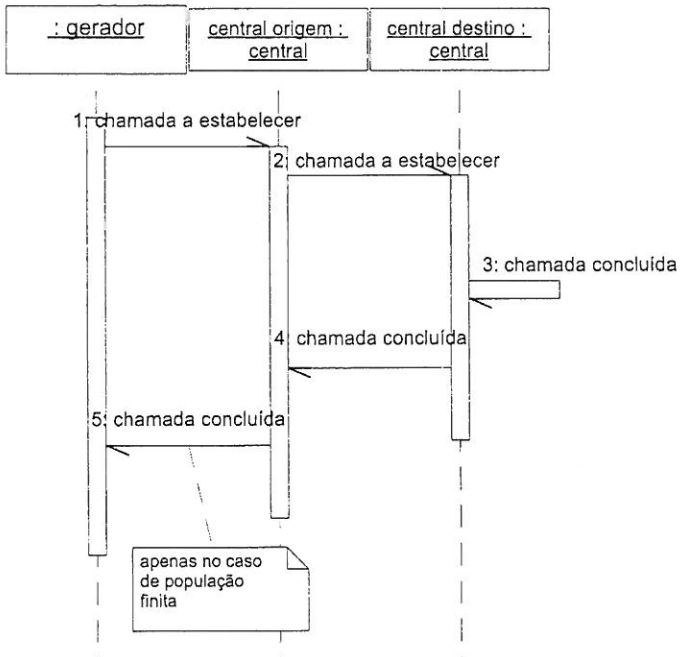

Fig. 3 – Diagrama de sequência (encaminhamento directo).

No segundo diagrama pode ver-se também a sequência temporal da passagem das mensagens.

Vamos agora analisar o caso em que o mecanismo de encaminhamento fornece à `central` origem, depois dela ter recebido do `gerador` uma chamada do tipo "chamada a estabelecer", um caminho com uma central de trânsito. Neste caso a `central` origem determina se existe capacidade disponível no ramo que a liga à `central` intermédia e se assim for envia para a `central` intermédia a chamada do tipo "chamada a estabelecer". Por sua vez a `central` intermédia, após ter recebido uma chamada do tipo "chamada a estabelecer", determina se existe capacidade disponível no ramo que a liga à `central` destino e se assim for envia para a `central` destino a chamada do tipo "chamada a estabelecer". Quando a `central` destino recebe uma chamada do tipo "chamada a estabelecer" cria uma mensagem do tipo "chamada concluída" que vai receber quando o tempo de ocupação da chamada terminar. Quando isso ocorrer a `central` destino envia essa mensagem para a `central` intermédia. A `central` intermédia ao receber esta mensagem envia-a para a `central` origem. A `central` origem ao receber esta mensagem envia-a para o `gerador`, mas, apenas no caso de ela corresponder a um fluxo de tráfego gerado por uma população finita. Os diagramas seguintes ilustram este processo.

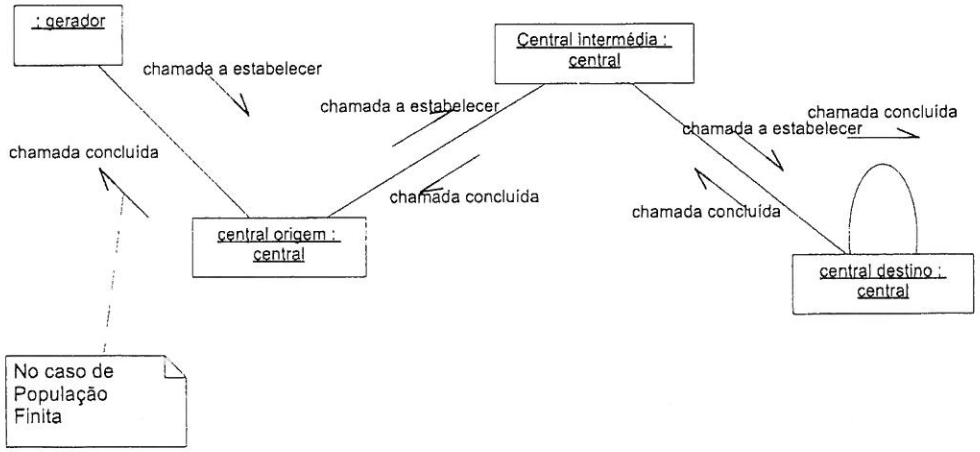

Fig. 4 - Diagrama de colaboração (encaminhamento alternativo).

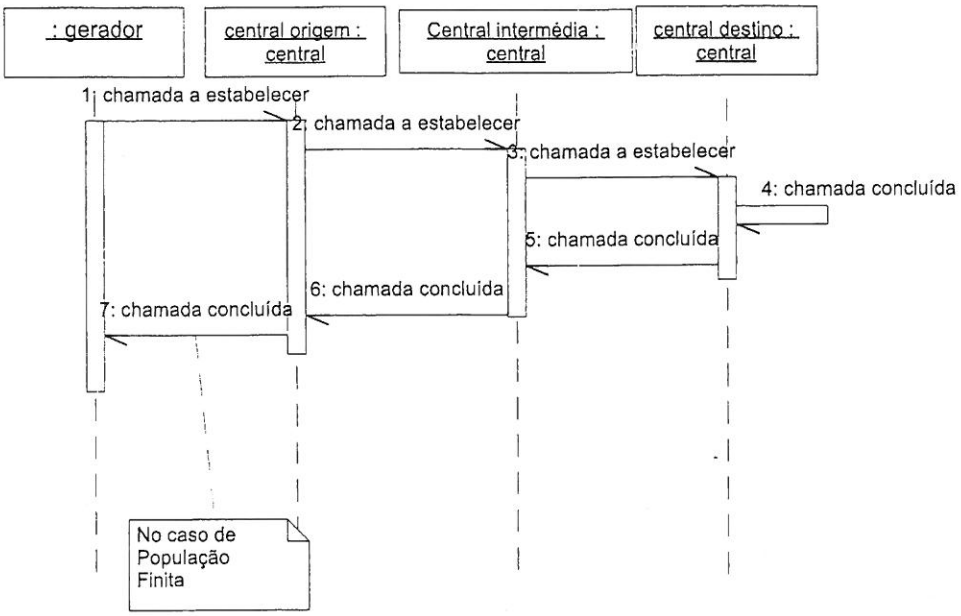

Fig. 5 – Diagrama de sequência (encaminhamento alternativo).

No segundo diagrama pode ver-se também a sequência temporal da passagem das mensagens.

Uma outra situação que pode ocorrer no caso anterior é, quando a chamada do tipo "chamada a estabelecer" chegar à `central` intermédia, esta determinar que não existe capacidade disponível no ramo que a liga à `central` destino. Neste caso duas coisas podem acontecer dependendo da `central` intermédia permitir ou não *crankback*. No caso da `central` intermédia não permitir *crankback*, ela envia uma mensagem do tipo "chamada impossível" para a `central` origem. A `central` origem ao receber esta mensagem envia-a para o `gerador`, mas, apenas no caso de ela corresponder a um fluxo de tráfego gerado por uma população finita.

No caso da `central` intermédia permitir *crankback*, ela envia uma mensagem do tipo "chamada crankback" para a `central` origem. A `central` origem ao receber uma mensagem desse tipo solicita ao mecanismo de encaminhamento que lhe indique um caminho alternativo para estabelecer a chamada. O caminho indicado, à `central` origem, pelo mecanismo de encaminhamento é normalmente um caminho que tem uma central de trânsito (situação mais frequente). Neste caso a `central` origem determina se existe capacidade disponível no ramo que a liga à nova `central` intermédia e se assim for envia para a nova `central` intermédia a chamada do tipo "chamada a estabelecer". Por sua vez esta `central`, após ter recebido uma chamada do tipo "chamada a estabelecer", determina se existe capacidade disponível no ramo que a liga à `central` destino e se assim for envia para a `central` destino a chamada do tipo "chamada a estabelecer". Quando a `central` destino recebe uma chamada do tipo "chamada a estabelecer" cria uma mensagem do tipo "chamada concluída" que vai receber quando o tempo

de ocupação da chamada terminar. Quando isso ocorrer a `central` destino envia essa mensagem para a `central` intermédia (`central` que foi utilizada no encaminhamento da chamada). Esta `central` intermédia ao receber essa mensagem envia-a para a `central` origem. A `central` origem ao receber a mensagem envia-a para o `gerador`, mas, apenas no caso de ela corresponder a um fluxo de tráfego gerado por uma população finita. Este caso pode ser observado nos dois diagramas seguintes.

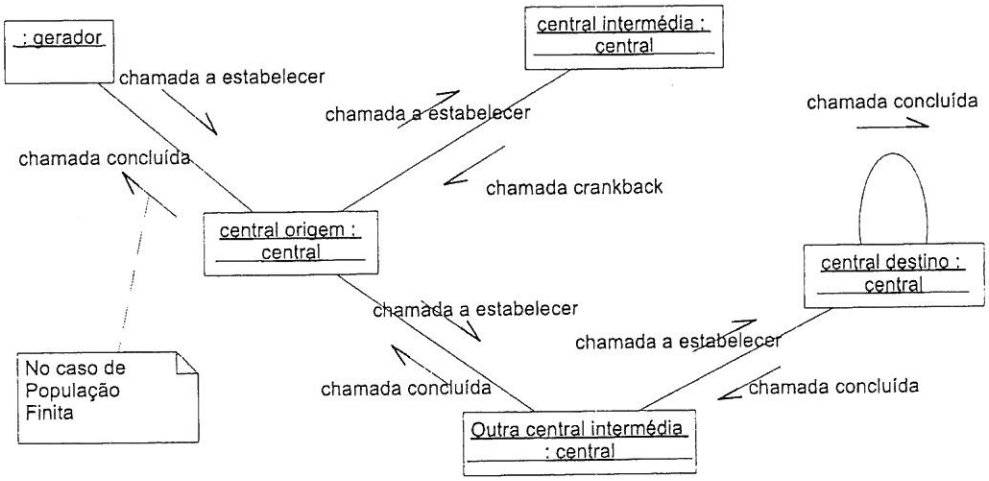

Fig. 6 - Diagrama de colaboração (encaminhamento alternativo após *crankback*).

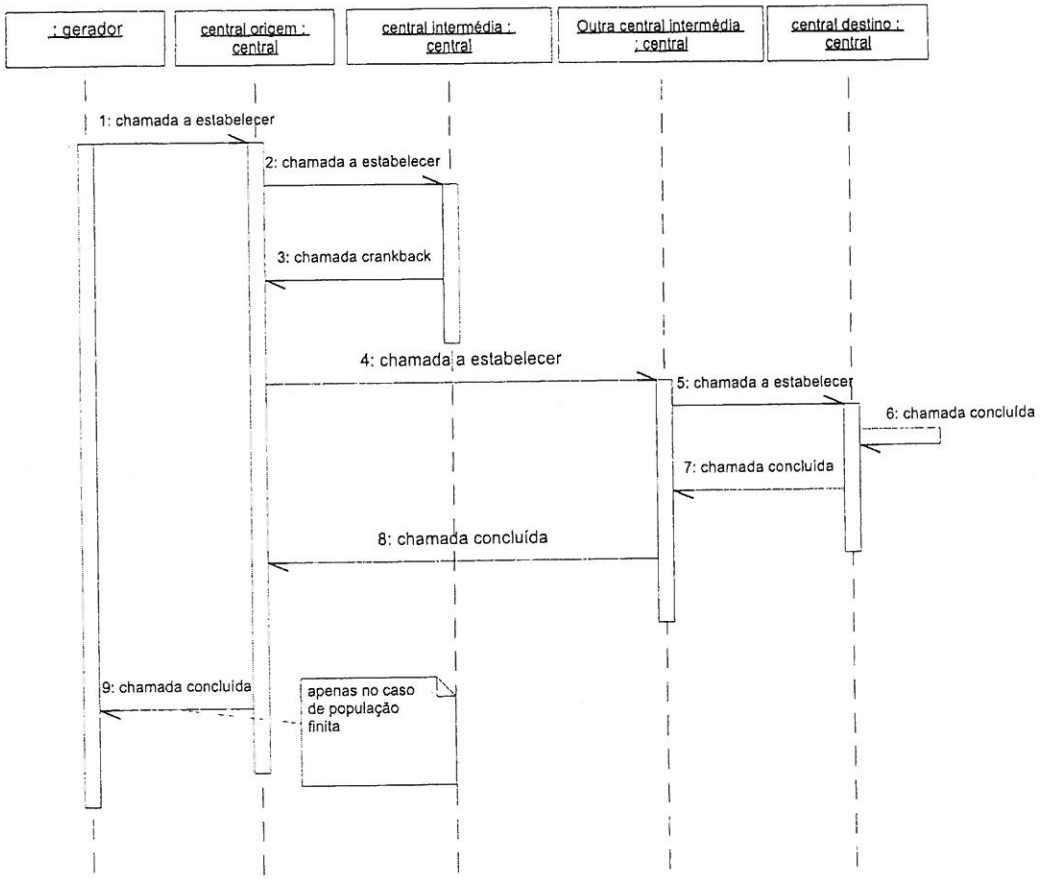

Fig. 7 – Diagrama de sequência (encaminhamento alternativo após *crankback*).

Tal como nos casos anteriores, apresenta-se no segundo diagrama a sequência temporal da passagem das mensagens, neste caso bastante útil dado o número de mensagens utilizadas.

Note-se que as mensagens que são passadas entre os vários processos, em todos os diagramas, são assíncronas o que significa que o processo que envia uma mensagem não fica inactivo, continua o processamento independentemente do tratamento que seja dado à mensagem pelo processo destino.

Os casos aqui apresentados embora englobem as situações principais do comportamento do processo `central` não são exaustivos.

Antes de entrar nos pormenores referentes ao funcionamento do processo `central` e `gerador` vão ser apresentadas sucintamente as características de cada um deles. Para isso lembra-se que existem tantos processos centrais no modelo como centrais na rede e que a cada processo `central` esta associado um processo `gerador`.

As características principais de cada `gerador` são:

- Modelar a geração de vários tipos de fluxos de tráfego, podendo coexistir vários no mesmo modelo; os fluxos de tráfego gerados, de diferentes tipos, podem usar parâmetros diferentes para os processos estocásticos de chegada e/ou terminação;
- Obter no inicio da simulação, a partir de um ficheiro, valores para vários parâmetros; no caso de tráfego exógeno de Poisson esses valores são as intensidades de tráfego entre quaisquer dois pares de centrais; no caso de tráfego exógeno de população finita (tipo Engset) esses valores são o número de fontes de tráfego em cada origem, a intensidade de chegadas por fonte livre e a proporção de tráfego gerado nessa `central` para cada destino;
- Para ambos os tipos de tráfego exógeno haverá ainda que lêr (a partir do ficheiro referido anteriormente) o número de canais (de 64 kbps) necessários pelas chamadas de cada tipo de fluxo;
- Criar, para a `central` à qual está ligado, ao longo de toda a simulação, tráfego que respeite os parâmetros referidos anteriormente.

As características principais de cada `central` são:

- Chamar as funções necessárias para determinar o caminho a seguir por cada uma das mensagens (chamadas) recebidas do `gerador` (ou de outras centrais) durante toda a simulação;
- Chamar as funções necessárias para armazenar as estatísticas respeitantes a esse tráfego;
- Enviar as chamadas da origem até ao destino passando por toda as centrais intermédias (mesmo que o caminho completo - caminho que a chamada deve seguir até ao destino - tenha sido conhecido logo na `central` origem). Decrementar a capacidade disponível dos ramos sempre que uma chamada é enviada por eles em direcção ao destino;
- Aquando do término da chamada, reenviar uma mensagem do tipo "chamada concluída" do destino em direcção à origem, pelo percurso inverso ao utilizado pela chamada a estabelecer. Incrementar as capacidades disponíveis à medida que essa mensagem passa nos várias ramos em direcção à origem;
- Chamar as funções necessárias para actualizar a informação a ser usada na decisão de encaminhamento das chamadas.

5.2.1 Processo Central

Segue-se a descrição em pormenor do funcionamento do processo `central`, indo para tal usar-se o diagrama seguinte.

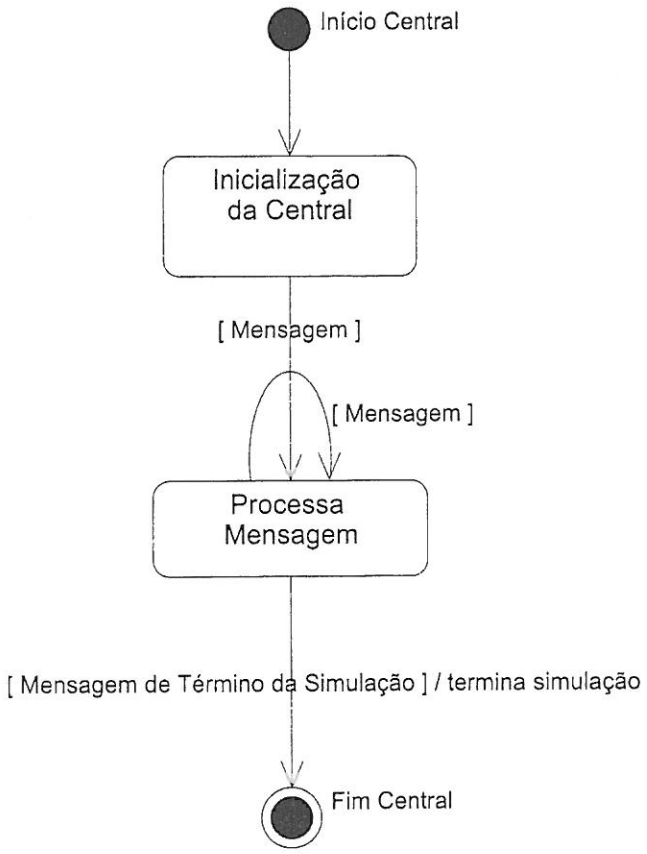

Fig. 8 - Diagrama de estados do processo `central`.

O funcionamento de cada `central` pode ser descrito por dois estados principais. Quando se inicia a simulação cada processo `central` entra no primeiro estado, designado no diagrama anterior por Inicialização da Central. No diagrama de actividade seguinte, correspondente a este estado, podemos observar as várias actividades desencadeadas neste estado bem como os objectos intervenientes.

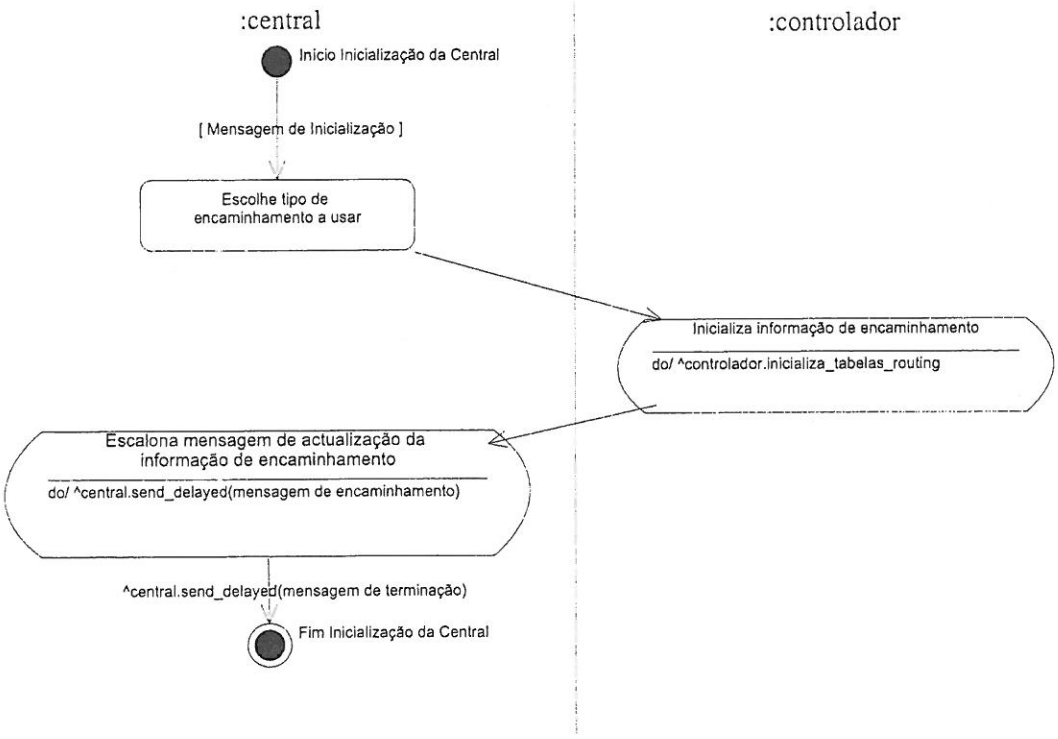

Fig. 9 – Diagrama de actividade do estado Inicialização da Central.

Cada `central` antes de iniciar o processamento necessita de receber do processo `gerador` uma mensagem de inicialização para obter dela o seu número na rede. Depois desta recepção, passa ao estado em que determina qual o encaminhamento a usar, bem como o tipo de controlo (modo de actualização da informação a usar pelo método de encaminhamento). Existe uma função que permite escolher explicitamente quais dos componentes de encaminhamento construídos que serão usados numa simulação particular. A função chama-se `escolhe_encaminhamento` e é invocada por cada processo `central` como indicado no diagrama da figura 9. Nessa função são criadas instâncias das classes derivadas da classe `stats`, da classe `controlador` e da classe `router`, correspondentes ao componente de encaminhamento que se pretende simular. As classes `stats`, `controlador` e `router` e as classes derivadas delás são referidas adiante.

Depois disso chama um método do objecto `controlador`, ou de uma sua especialização, que faz a inicialização da informação que é usada no encaminhamento das chamadas (por exemplo a inicialização das tabelas de encaminhamento). A próxima actividade executada por cada `central` é Escalona Mensagem de Actualização da Informação (o que corresponde a enviar uma mensagem para si própria), para ser feita a actualização da informação usada no encaminhamento das chamadas. Por último, cada `central` escalona uma mensagem de terminação para a altura em que a simulação deva terminar, o que corresponde a enviar uma mensagem de término da simulação para si própria e que vai ser recebida por si quando for o instante em que a simulação deva terminar.

Terminado o estado de Inicialização da Central cada uma das centrais inicia o estado Processa Mensagem. As centrais só deixam de estar neste estado quando ocorrer a mensagem de término da simulação. Quando esta mensagem ocorrer a simulação termina.

No diagrama de actividade seguinte, correspondente à decomposição do estado Processa Mensagem, podemos observar as várias actividades desencadeadas neste estado bem como os objectos intervenientes.

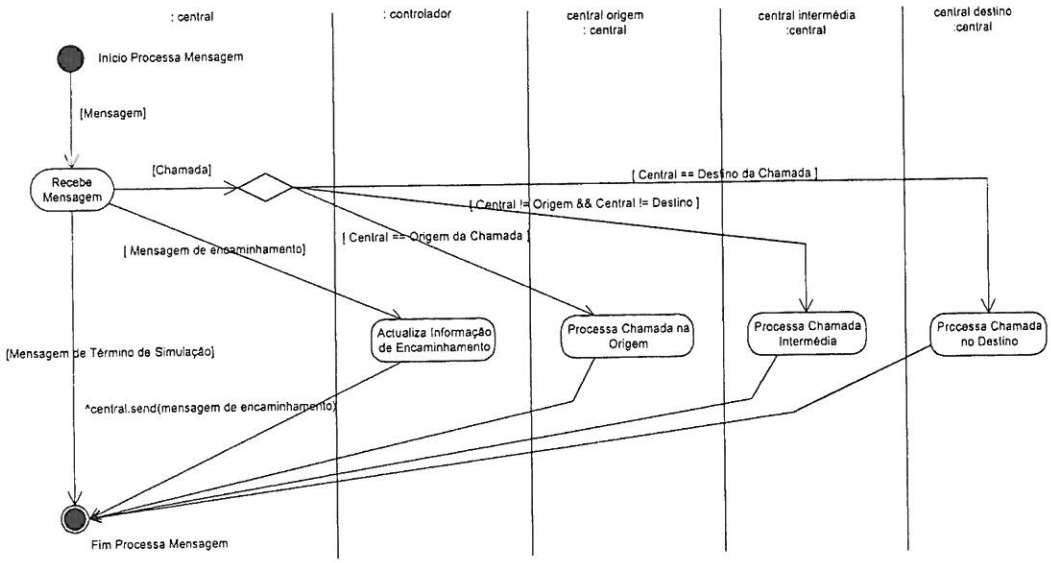

Fig. 10 – Diagrama de actividade do estado Processa Mensagem.

No estado Processa Mensagem cada `central` espera por uma mensagem, podendo a mensagem ter sido enviada pela própria `central`, por qualquer outra `central` ou pelo `gerador`. A mensagem foi gerada pela própria `central` no caso da mensagem ser do tipo "mensagem de encaminhamento", ou ser do tipo "chamada concluída", ou ainda de outros. A mensagem foi gerada pelo `gerador` no caso da mensagem ser do tipo "chamada a estabelecer". A mensagem foi gerada por outra `central` no caso da mensagem ser do tipo "chamada impossível", ou do tipo "chamada concluída", ou ainda de outros. Quando chegar um mensagem à `central` ela vai processá-la. O tratamento a dar a cada mensagem depende do tipo da mensagem. Caso a mensagem recebida corresponda a uma chamada a `central` vai determinar se se trata de uma chamada para a qual ela é a `central` origem (`central` onde a chamada surgiu do `gerador`), ou se se trata de uma chamada para a qual ela é a `central` destino (`central` para onde a chamada pretende ser estabelecida) ou se se trata de um chamada para a qual ela é uma `central` intermédia (`central` de trânsito usado no estabelecimento da chamada). Depois disso chama o método do objecto `central` correspondente à condição que for verificada. A `central` fica assim no estado Processa Chamada na Origem, ou no estado Processa Chamada Intermédia ou no estado Processa Chamada no Destino.

Caso a mensagem corresponda a uma mensagem de actualização da informação de encaminhamento, a `central` passa neste caso ao estado Actualiza Informação de Encaminhamento. Neste caso a `central` chama um método do objecto `controlador`, ou de uma sua especialização, que faz a actualização da informação a usar no encaminhamento das chamadas. Depois do estado Actualiza Informação de Encaminhamento a `central` vai escalonar (o que corresponde a enviar a mensagem para si própria) uma mensagem do tipo "mensagem de encaminhamento". Esta mensagem de encaminhamento será a próxima mensagem, deste tipo, que a `central` irá receber.

Caso a mensagem seja uma mensagem de término da simulação a `central` desencadeia as operações que devem ser realizadas imediatamente antes da simulação terminar.

No diagrama de actividade da Figura 11, correspondente ao estado Processa Chamada na Origem, podemos observar as várias actividades desencadeadas neste estado.

A primeira coisa que a `central` faz quando entra no estado Processa Chamada na Origem é verificar qual o tipo da mensagem que lá chegou. Caso a mensagem seja do tipo "chamada a estabelecer" a `central` vai tratar uma chamada que acabou de chegar do `gerador`. Para isso começar por chamar um método do objecto `stats`, ou de uma sua especialização, que faz a contabilização do número de chamadas novas. De seguida chamar um método do objecto `router`, ou de uma sua especialização, que determina a próxima `central` e/ou o caminho previsto para atingir a `central` destino. Depois disto a `central` verifica a capacidade disponível do ramo que liga esta `central` à próxima `central`. Se a capacidade disponível permitir estabelecer a chamada nesse ramo a actividade seguinte será Envia Chamada para Próxima Central.

Se a actividade a realizar for Envia Chamada para Próxima Central, então a chamada irá ocupar os circuitos necessários nesse ramo. É assim necessário efectuar as acções seguintes: decrementar a capacidade disponível desse ramo, chamar um método do objecto `router`, ou de uma sua especialização, para o informar de que a capacidade disponível do ramo foi alterada, chamar um método do objecto `stats`, ou de uma sua especialização, para ser contabilizada a ocupação de circuitos no ramo e enviar a mensagem para a próxima `central`. Após a realização desta actividade o processamento da mensagem termina.

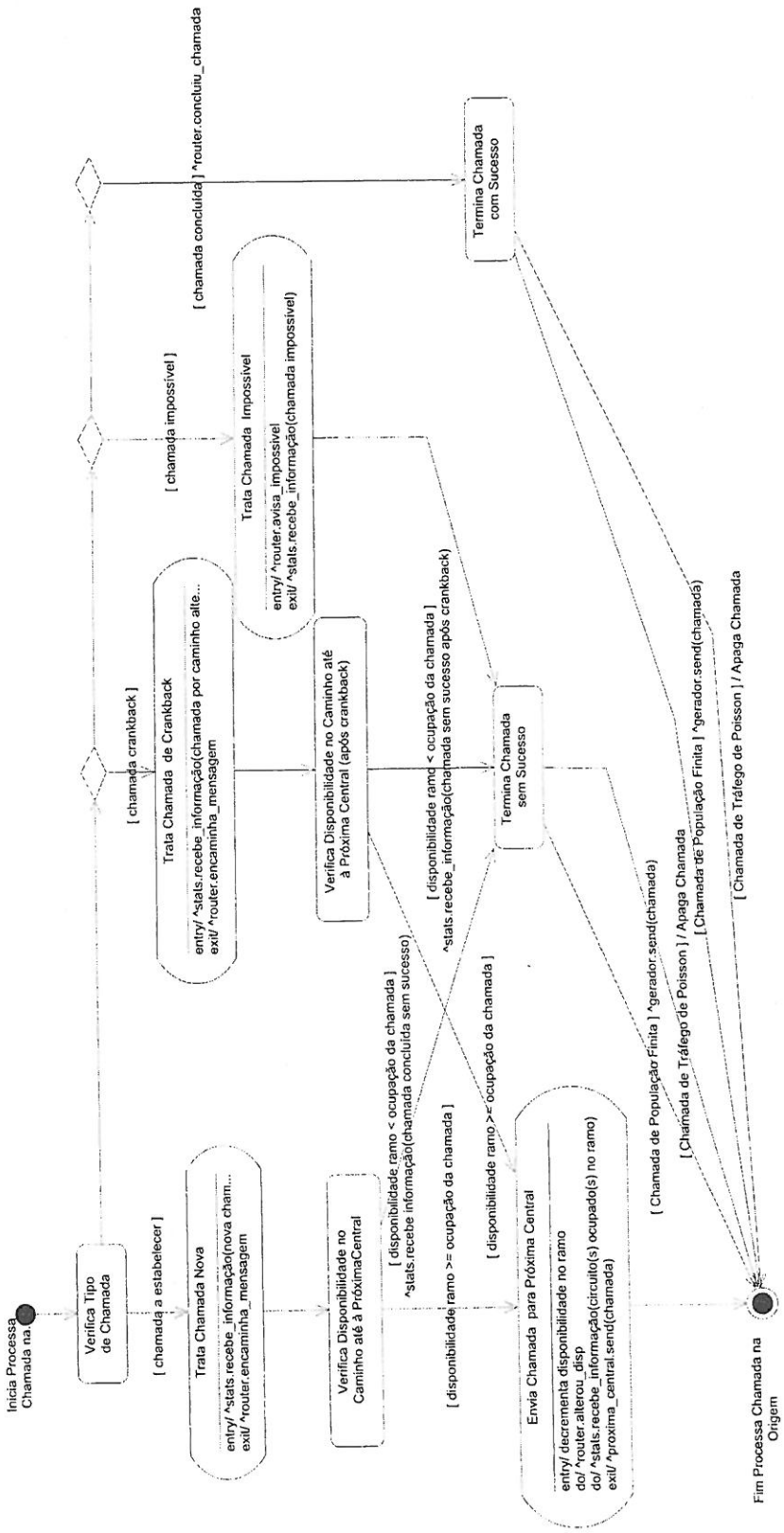

Fig. 11 – Diagrama de actividade do estado Processa Chamada na Origem.

Se, ainda no caso da mensagem ser do tipo "chamada a estabelecer", a capacidade disponível não permitir estabelecer a chamada, no ramo que liga à próxima `central`, é necessário chamar um método do objecto `stats`, ou de uma sua especialização, para ser contabilizada uma chamada terminada sem sucesso. Neste caso a `central` passa ao estado de escolha Termina Chamada sem Sucesso. Se a chamada corresponder a um fluxo de tráfego de população finita a `central` envia a chamada para o `gerador`. Caso contrário, isto é, no caso da chamada corresponder a um fluxo de tráfego de Poisson a `central` apaga simplesmente a mensagem. Em ambas as situações, termina o processamento da mensagem.

Caso a mensagem seja do tipo "chamada crankback" a `central` realiza a actividade Trata Chamada de Crankback. Para isso é necessário executar as duas acções seguintes: chamar um método do objecto `stats`, ou de uma sua especialização, para ser contabilizada uma chamada por caminho alternativo e chamar um método do objecto `router`, ou de uma sua especialização, para determinar a próxima `central` e/ou o caminho previsto para atingir a `central` destino. Depois disto a `central` verifica a capacidade disponível do ramo que liga esta `central` à próxima `central`. Se a capacidade disponível permitir estabelecer a chamada nesse ramo a actividade seguinte será Envia Chamada para Próxima Central. Esta actividade é a mesma que a realizada no caso da mensagem ser do tipo "chamada a estabelecer", que já foi descrita. Depois desta actividade ser realizada, termina o processamento da mensagem.

Se, ainda no caso da mensagem ser do tipo "chamada crankback", a capacidade disponível não permitir estabelecer a chamada, no ramo que liga à próxima `central`, é necessário chamar um método do objecto `stats`, ou de uma sua especialização, para ser contabilizada uma chamada terminada sem sucesso depois de ter ocorrido *crankback*. Neste caso a `central` passa ao estado de escolha Termina Chamada sem Sucesso, já anteriormente descrito.

Caso a mensagem seja do tipo "chamada impossível" a `central` chama um método do objecto `router`, ou de uma sua especialização. Depois chama um método do objecto `stats`, ou de uma sua especialização, para contabilizar uma chamada concluída sem sucesso. E de seguida passa ao estado de escolha Termina Chamada sem Sucesso. A partir daqui tudo se passa como nos casos anteriores.

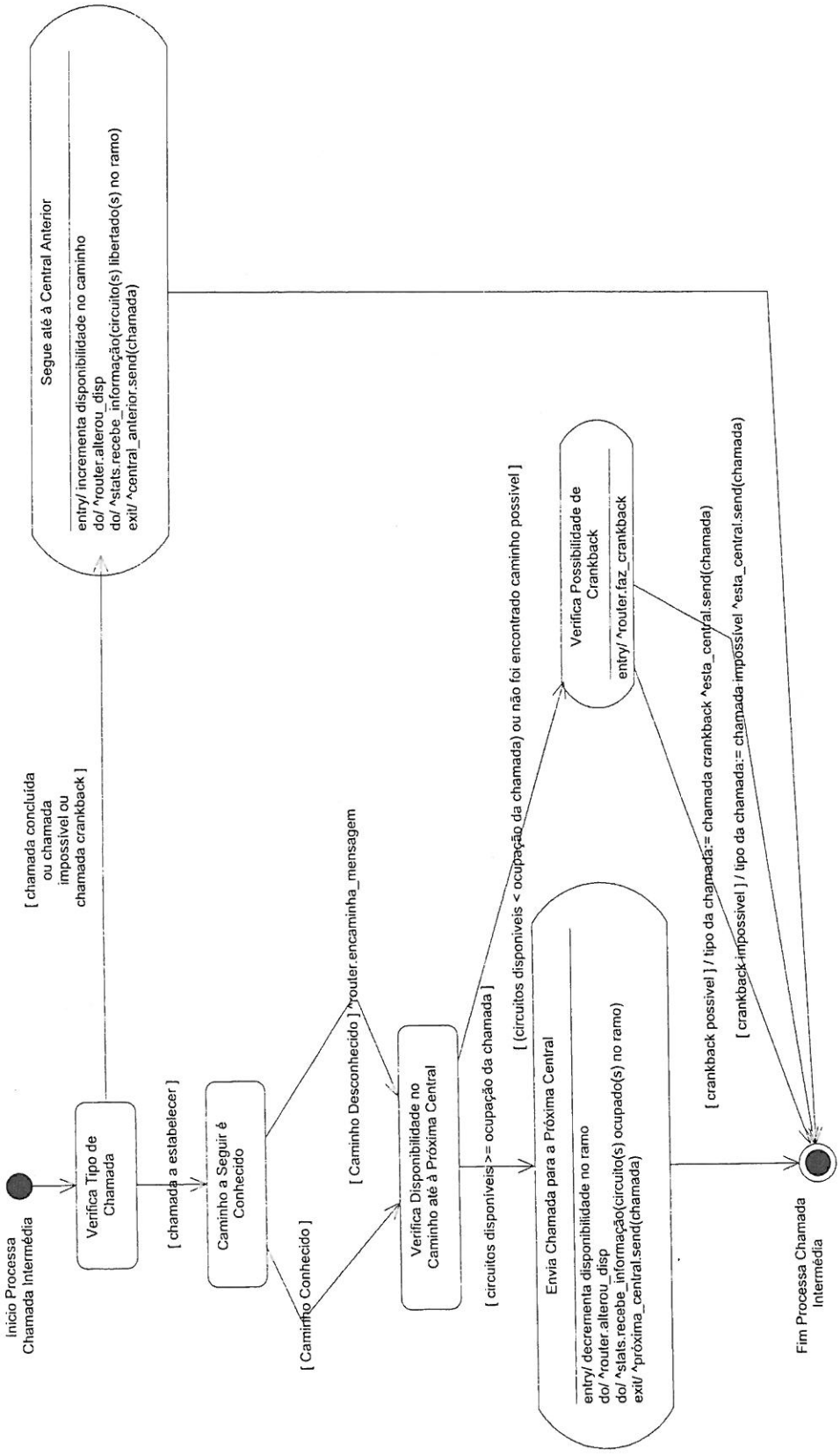

Fig. 12 – Diagrama de actividade do estado Processa Chamada Intermédia.

Caso a mensagem seja do tipo "chamada concluída" a `central` chama um método do objecto `router`, ou de uma sua especialização, para o informar que foi concluída uma chamada. Neste caso a `central` passa ao estado de escolha Termina Chamada com Sucesso. Se a chamada corresponder a um fluxo de tráfego de população finita a `central` envia a chamada para o `gerador`. Caso contrário, isto é, no caso da chamada corresponder a um fluxo de tráfego de Poisson a `central` apaga simplesmente a mensagem. Em ambas as situações, termina o processamento da mensagem. Acaba assim o estado Processa Chamada na Origem.

No diagrama de actividade da Figura 12, correspondente ao estado Processa Chamada Intermédia, podemos observar as várias actividades desencadeadas neste estado.

A primeira coisa que a `central` faz quando entra no estado Processa Chamada Intermédia é verificar qual o tipo da mensagem que lá chegou. Caso a mensagem seja do tipo "chamada a estabelecer" a `central` vai tratar uma chamada que acabou de chegar a uma `central` de trânsito. Quando uma chamada chega a uma `central` de trânsito uma das duas situações pode acontecer: a chamada já conhece o caminho que deve seguir ou a chamada ainda não conhece o caminho a seguir. Caso o caminho a seguir já seja conhecido, a `central` vai verificar a capacidade disponível do ramo que liga esta `central` (`central` onde a chamada se encontra) à próxima `central` no caminho. Se a capacidade disponível do ramo permitir estabelecer a chamada a `central` passa a realizar a actividade Envia Chamada para a Próxima Central. As acções a executar para realizar esta actividade são as seguintes: decrementar a capacidade disponível desse ramo, chamar um método do objecto `router`, ou de uma sua especialização, para o informar de que a capacidade disponível do ramo foi alterada, chamar um método do objecto `stats`, ou de uma sua especialização, para ser contabilizada a ocupação de circuitos no ramo e enviar a mensagem para a próxima `central`. Após a realização desta actividade o processamento da mensagem termina.

Considere-se ainda o caso da mensagem ser do tipo "chamada a estabelecer", mas em que se verificou uma das duas situações seguintes: não havia capacidade disponível no ramo que liga à próxima `central` ou não foi encontrado caminho possível. Neste caso a `central` determina se é possível fazer *crankback*, para isso recorrendo ao objecto `router`, ou uma sua especialização. Se for possível fazer *crankback* envia para si própria uma mensagem, do tipo "chamada crankback". Se não for possível fazer *crankback* envia para si própria uma mensagem, do tipo "chamada impossível".

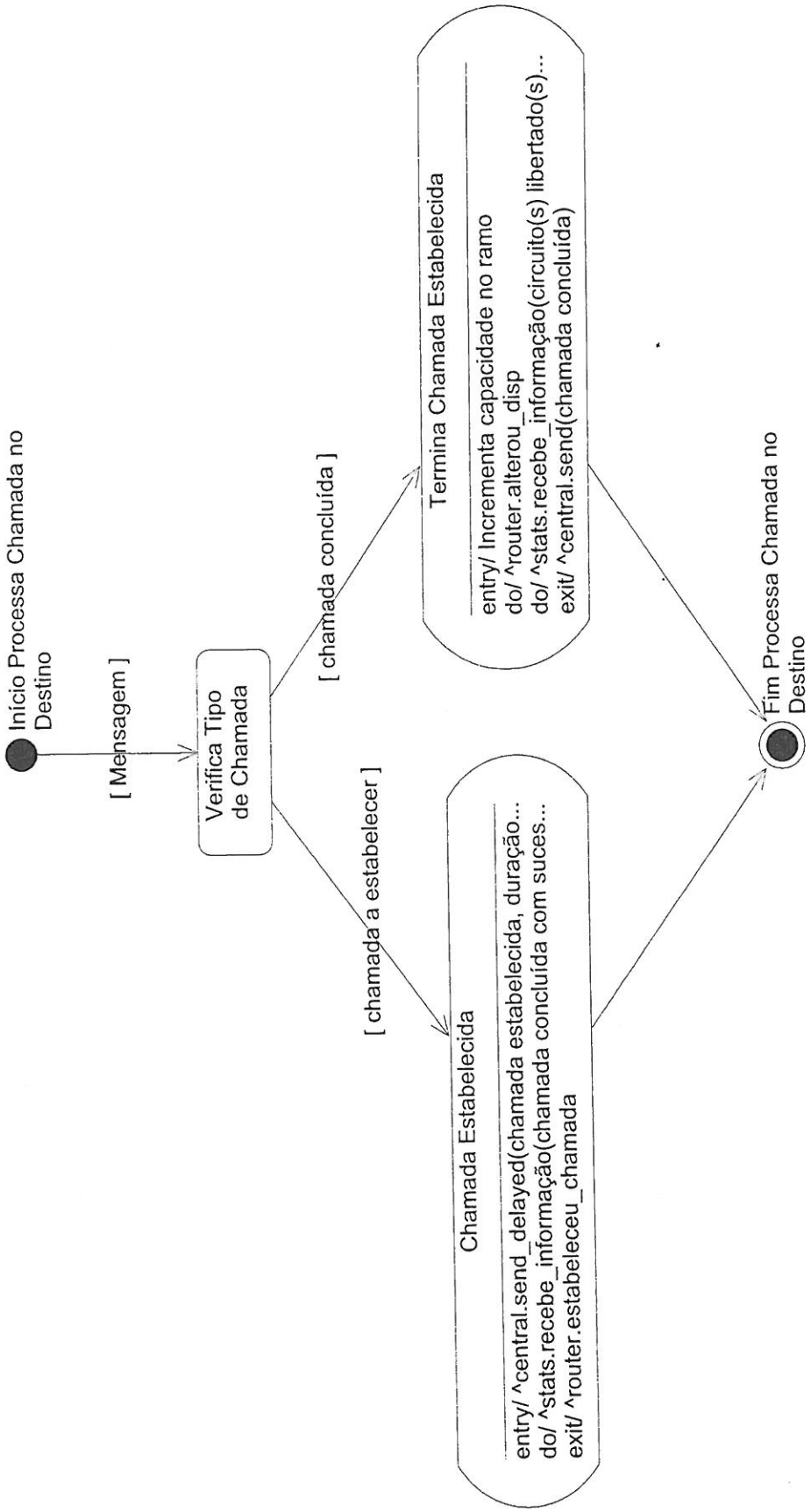

Fig. 13 – Diagrama de actividade do estado Processa Chamada no Destino.

No caso ainda da mensagem ser do tipo "chamada a estabelecer", mas na situação do caminho a seguir não ser conhecido, a `central` chama um método do objecto `router`, ou de uma sua especialização, para determinar a próxima `central` e/ou caminho previsto para atingir a `central` destino. Tudo o que se passa a seguir é o mesmo que no caso anterior a partir do estado de escolha Verifica Disponibilidade no Caminho até à Próxima Central.

No caso da mensagem ser do tipo "chamada concluída", "chamada impossível" ou "chamada crankback" a única actividade a realizar é Determina Caminho a Seguir até à Origem. Para realizar esta actividade é necessário executar as seguintes acções: incrementar a capacidade disponível do ramo que liga esta `central` à `central` anterior, chamar um método do objecto `router`, ou de uma sua especialização, para o informar de que a capacidade disponível do ramo foi alterada, chamar um método do objecto `stats`, ou de uma sua especialização, para ser contabilizada a libertação de circuitos no ramo e por último enviar a chamada para a `central` anterior. Após a realização destas acções o processamento da mensagem termina. Note que, neste parágrafo utilizou-se a designação `central` anterior quando se queria referir a `central`, que foi utilizada no caminho usado no estabelecimento da chamada, por onde a chamada tinha passado, no seu estabelecimento, imediatamente antes de chegar à `central` onde a chamada se encontra.

No diagrama de actividade da Figura 13, correspondente à decomposição do estado Processa Chamada no Destino, podemos observar as várias acções desencadeadas neste estado.

A primeira coisa que a `central` faz quando entra no estado Processa Chamada no Destino é verificar qual o tipo da mensagem que lá chegou. Caso a mensagem seja do tipo "chamada a estabelecer" a `central` vai tratar uma chamada que acabou de chegar ao destino. Para isso começa por escalonar uma mensagem do tipo "chamada concluída" (o que corresponde a enviar uma mensagem para si própria). Esta mensagem vai ser recebida pela própria `central` com um atraso igual ao tempo de duração da chamada. Depois chama um método do objecto `stats`, ou de uma sua especialização, que faz a contabilização do número de chamadas concluídas com sucesso. E por fim chama um método do objecto `router`, ou de uma sua especialização, para o informar que foi estabelecida um chamada.

Caso a mensagem seja do tipo "chamada concluída" a `central` vai tratar uma chamada que ela própria escalonou. Esta chamada indica que o tempo de duração de uma chamada que tinha sido estabelecida chegou ao fim. Por isso a `central` vai realizar a actividade Termina Chamada Estabelecida. Nesta actividade são executadas as seguintes acções: incrementar a capacidade disponível do ramo que liga esta `central` à `central` anterior, chamar um método do objecto `router`, ou de uma sua especialização, para o informar de que a capacidade disponível do ramo foi alterada, chamar um método do objecto `stats`, ou de uma sua especialização, para ser

contabilizada a libertação de circuitos no ramo e por último enviar a chamada para a `central` anterior. Após a realização destas acções o processamento da mensagem termina. Neste parágrafo utilizou-se a designação `central` anterior com o mesmo sentido utilizado na descrição do estado Processa Chamada Intermédia.

5.2.2 Processo Gerador

Esta subsecção vai ser dedicada ao modo de funcionamento do processo `gerador`. Para ilustrar o seu funcionamento vai usar-se o diagrama seguinte:

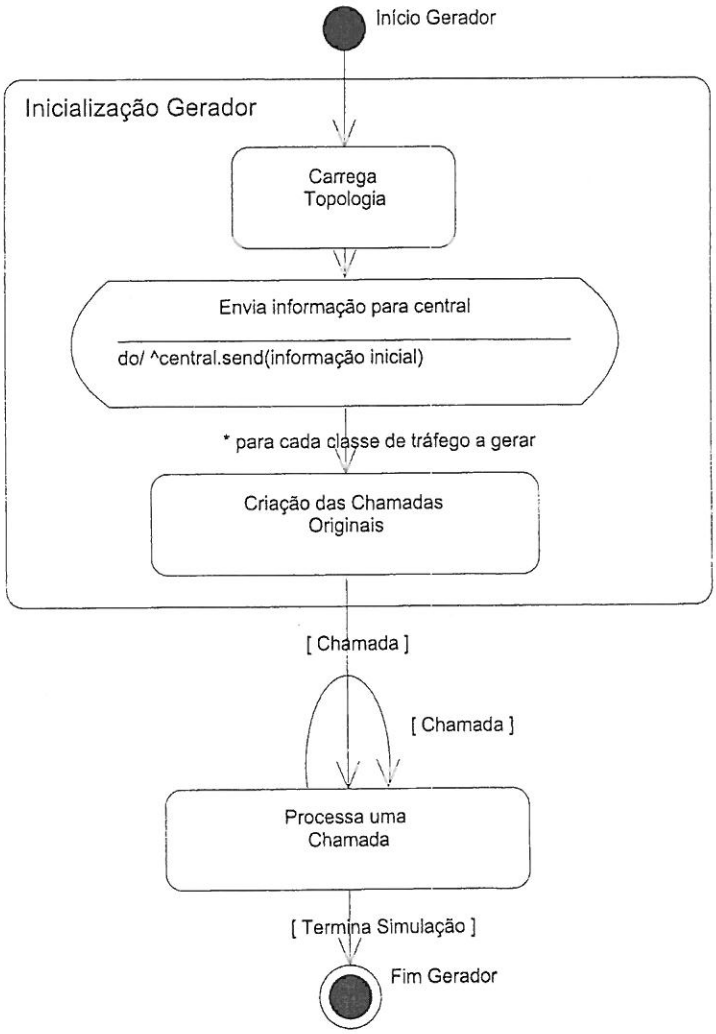

Fig. 14 – Diagrama do processo `gerador`.

O funcionamento de cada `gerador` pode ser também descrito por dois estados principais, como aconteceu para o processo `central`. Quando se inicia a simulação cada processo `gerador` começa no estado designado por Inicialização Gerador. Este estado pode ser decomposto em vários subestados. No primeiro subestado designado por Carrega Topologia, o `gerador` vai ler a configuração da rede de um ficheiro, que contém a informação sobre a topologia da rede, e ler

também desse ficheiro os parâmetros do modelo, por exemplo os parâmetros dos tráfegos. Depois do subestado Carrega Topologia estar completo, a próxima acção efectuada pelo `gerador` é enviar para a `central`, à qual está ligado, uma mensagem com informação necessária à inicialização da `central`. Após isto, e para completar o processo de inicialização, o `gerador` tem ainda que passar no subestado designado por Criação das Chamadas Originais, tantas vezes quanto o número de classes de tráfego a gerar.

No diagrama de actividades seguinte, correspondente ao subestado Criação das Chamadas Originais, podemos observar as várias acções desencadeadas neste subestado.

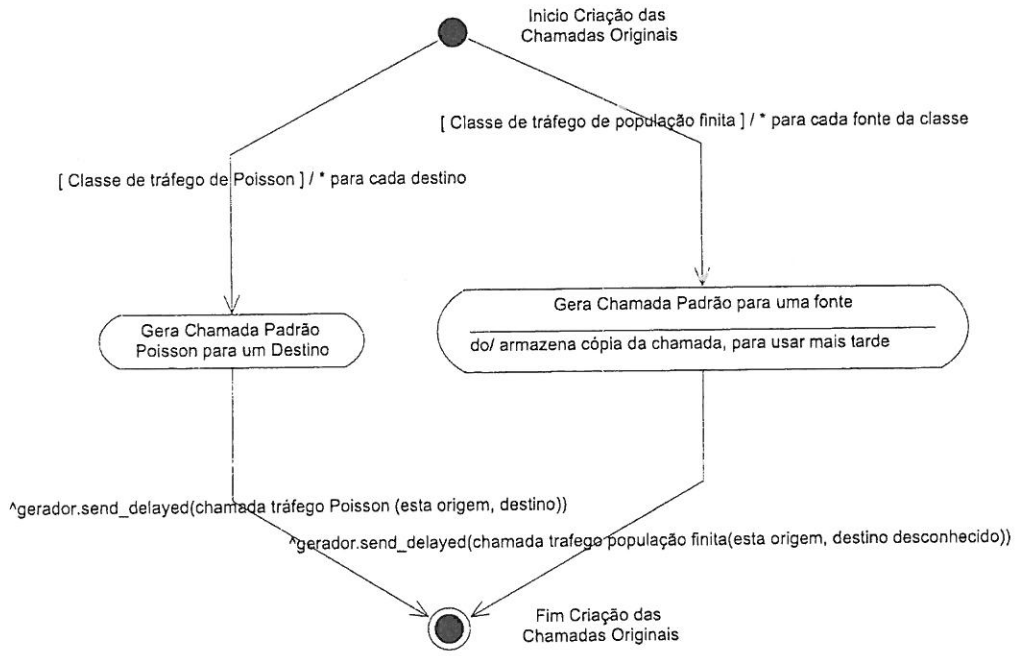

Fig. 15 – Diagrama de actividade do subestado Criação das Chamadas Originais.

As acções a serem realizadas dependem da classe de tráfego a gerar (tipo de fluxo de tráfego a gerar). Se o tráfego a gerar corresponde a uma classe de tráfego de Poisson (o que corresponde ao caso de, no sistema, existir um tipo de fluxo de tráfego no qual o tráfego exógeno é de Poisson) é criada, para cada uma das centrais destino (centrais destino são todas as centrais para as quais exista tráfego com origem na `central` que se encontra ligada ao `gerador`), uma mensagem do tipo "chamada a estabelecer" e é escalonada essa mensagem (o que corresponde a enviá-la para si próprio). Cada uma dessas mensagens vai ser recebida pelo próprio `gerador` com um atraso igual ao tempo aleatório que demora a ocorrer uma nova chamada para o destino correspondente.

Se o tráfego a gerar corresponde a uma classe de tráfego de população finita (o que corresponde ao caso de, no sistema, existir um tipo de fluxo de tráfego no qual o tráfego exógeno é de População Finita) é criada, para cada indivíduo dessa população, uma mensagem do tipo

"chamada a estabelecer" e é escalonada essa mensagem. Esta mensagem vai ser recebida pelo próprio `gerador` com um atraso igual ao tempo, aleatório, que demora a ocorrer uma nova chamada desse indivíduo. Além de criar, para cada fonte, uma mensagem do tipo "chamada a estabelecer" é também guardada uma cópia de cada uma dessas mensagens, antes de serem escalonadas. Cada uma dessas cópias vai ser usada quando for necessário escalonar uma nova chamada com as mesma características (ver o estado Coloca uma Chamada em Espera no diagrama de estados Processa uma Chamada, apresentado na figura 16).

A cada mensagem, que corresponda a uma chamada, vão ser associados, quando a mensagem é criada, os seguintes parâmetros:

- Origem – número da `central` origem da chamada;
- Destino – número da `central` destino da chamada;
- Duração – duração da chamada;
- Ocupação - número de circuitos (canais a 64 kbps) necessários às chamadas dessa classe em cada ramo, ou seja é permitido a uma chamada ocupar mais do que um canal simultaneamente (tráfego multicanal);
- Caminho desde a origem – caminho seguido pela mensagem da origem até à `central` corrente (inclusive);
- Caminho previsto – caminho previsto para a mensagem seguir, desde a próxima `central` até ao destino (inclusive);
- Caminho percorrido – caminho colocado nas mensagens que voltam para trás quer por *crankback*, impossibilidade ou conclusão de ligação. Este caminho indica as centrais por onde a mensagem passou até à última `central` a que a mensagem chegou;
- Classe – tipo de fluxo de tráfego;
- População Finita – indica se a mensagem é ou não de tráfego de população finita;
- Número da fonte – número da fonte que gerou tráfego, apenas com significado no caso de mensagens de tráfego de população finita.

A cada chamada, que corresponda a uma classe de tráfego de População Finita vai ser associado também o seguinte parâmetro:

- Espera – que indica se uma fonte está inactiva (em espera) ou não.

Quando as mensagens são criadas, são atribuídos valores iniciais a todos os parâmetros. Alguns dos parâmetros são inicializados com valores que se mantêm durante toda a duração da simulação, entre os quais a classe, a informação sobre se é ou não tráfego de população finita e a

ocupação (número de circuitos a ocupar pela chamada, o que é necessário para suportar tráfego multicanal); outros são inicializados a nulo ou vazio. Alguns destes valores serão depois substituídos antes da mensagem ser enviada para a `central` correspondente.

É criado na fase de inicialização tráfego da `central` à qual cada `gerador` está ligado para todas as outras. Todo o tráfego restante é derivado deste, como veremos em seguida.

Terminado o estado Inicialização Gerador cada um dos `geradores` inicia o estado Processa uma Chamada. Os `geradores` permanecem neste estado até ao instante em que ocorra a terminação da simulação. Para auxiliar a descrição deste estado vamos utilizar o diagrama seguinte:

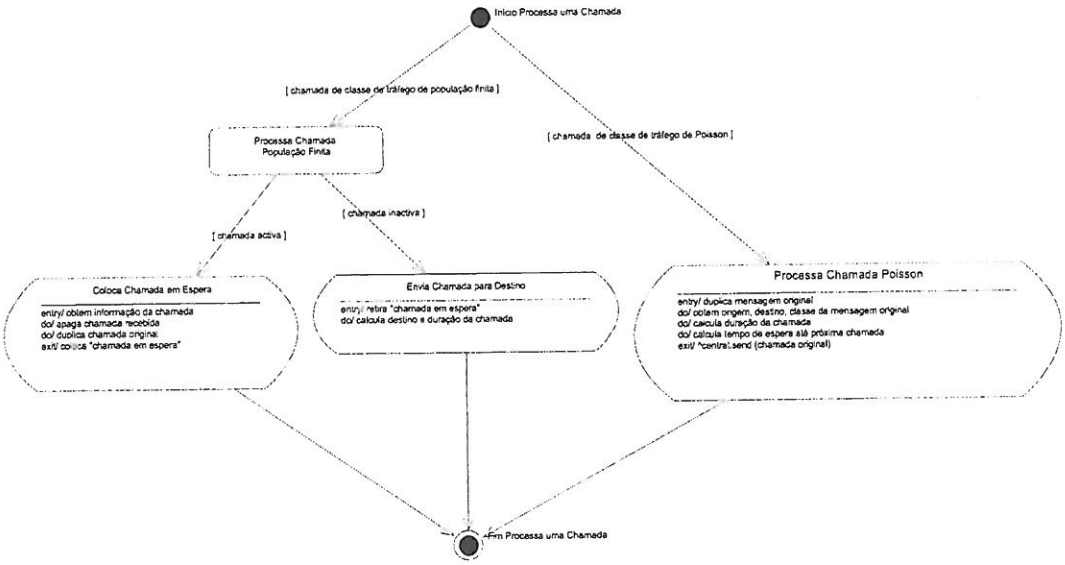

Fig. 16 – Diagrama de actividade do estado Processa uma Chamada.

No estado Processa uma Chamada cada `gerador` espera por uma mensagem (mensagem que corresponde sempre a uma chamada). Quando chegar uma chamada ao `gerador` este testa a mensagem para determinar se se trata de uma chamada de uma classe de tráfego de população finita ou se se trata de uma chamada de uma classe de tráfego de Poisson. Caso a chamada seja de uma classe de tráfego de população finita ela pode ter sido enviada pelo próprio ou pela `central` que está ligada ao `gerador`. A mensagem ter sido enviada pelo próprio `gerador` ocorre quando uma das fontes que estava inactiva esgotou o seu tempo de inactividade. A mensagem ter sido enviada pela `central` que está ligada ao `gerador` ocorre quando uma das fontes que estava activa acabou o seu tempo de serviço. Se a chamada que o `gerador` recebeu tinha sido enviada pela `central` (chamada activa) a próxima actividade, do `gerador`, é a actividade Coloca Chamada em Espera. Nesta situação as acções que é necessário realizar são as seguintes: determinar qual a chamada (fonte) que acabou o tempo de serviço; apagar a mensagem que foi recebida (porque a mensagem pode conter informação que não seja válida

para a mensagem que vai ser escalonada); obter uma copia da mensagem, correspondente à chamada que se recebeu, a partir das mensagens originais que temos guardadas; colocar o parâmetro espera com o valor verdadeiro (isto indica que a chamada passa a estar inactiva). De seguida, a mensagem que se criou, é escalonada, isto é, o `gerador` envia a mensagem para si próprio. Esta mensagem vai ser recebida pelo próprio `gerador` com um atraso igual ao tempo que foi determinado, aleatoriamente, como sendo o tempo de inactividade da fonte. Se a chamada que o `gerador` recebeu tinha sido enviada por si (chamada inactiva) a próxima actividade, do `gerador`, é a actividade Envia Chamada para Destino. Neste caso as acções que é necessário realizar são as seguintes: colocar o parâmetro espera com o valor falso (isto indica que a chamada passa a estar activa); determinar o destino, (aleatoriamente, de acordo com as probabilidades comulativas, que representam as proporções de tráfego da central origem para cada uma das centrais na rede); colocar o valor do destino determinado no parâmetro correspondente da mensagem; determinar a duração da chamada (utilizando um gerador de números aleatórios) e colocar esse valor no parâmetro correspondente da mensagem. De seguida o `gerador` vai enviar a mensagem para a `central` que esta ligada a si e que corresponde à `central` origem para a chamada.

Caso a chamada seja de uma classe de tráfego de Poisson o `gerador` passa à realização da actividade Processa Chamada Poisson. Nesta situação o `gerador` tem que realizar as acções seguintes: criar uma mensagem igual à que recebeu; determinar quais os geradores de números aleatórios a usar, para isto é necessário saber qual é a origem, o destino e a classe da chamada; determinar o valor do tempo de duração para a chamada (utilizando um gerador de números aleatórios); actualizar o parâmetro duração da mensagem; determinar o tempo até ocorrer uma nova chamada (utilizando um gerador de números aleatórios diferente e independente do anterior); e enviar para a `central` que está ligada a si a chamada original. Para terminar o estado Processa uma Chamada, para o caso apresentado, falta apenas escalonar a mensagem que se criou no estado Processa Chamada Poisson, isto é, o `gerador` envia a mensagem para si próprio. Esta mensagem vai ser recebida pelo próprio `gerador` com um atraso igual ao tempo que foi determinado, aleatoriamente, como sendo o tempo até ocorrer uma nova chamada.

Pela análise da descrição anterior constata-se que para modelizar as várias classes de tráfego foi necessário recorrer a geradores de números aleatórios.

Para simular as chamadas de populações de tamanho infinito foi utilizada uma distribuição exponencial negativa tanto para gerar o instante da próxima chamada como para determinar o tempo de serviço de cada chamada.

Para simular as chamadas de populações de tamanho finito foi utilizada uma distribuição exponencial negativa para determinar o tempo de serviço de cada chamada. Para gerar o instante da próxima chamada podia ser seguida uma de duas abordagem seguintes:

1. Considerar uma "fonte" que gerasse tráfego equivalente ao gerado por todas as diferentes fontes. As chamadas geradas por essa "fonte" equivalente não seguem nenhuma distribuição fixa, pois a intensidade de chamadas depende do número de fontes livres;
2. Considerar as diversas fontes separadamente. Neste caso para determinar o instante da próxima chamada (após a anterior ter concluído) pode ser usada a distribuição exponencial negativa. É necessário utilizar um número de variáveis aleatórias, associadas a essa distribuição, igual ao número de fontes (indivíduos da população).

Optou-se pela segunda por ser mais simples de implementar. Os resultados obtidos por simulação, considerando este princípio de geração de chamadas, são semelhantes aos obtidos pelo método analítico, como era esperado.

O processo `gerador` recorre a números aleatórios obtidos directamente de geradores de números aleatórios e também a números aleatórios obtidos de distribuições. O OMNeT++ possuía, já implementados, modos de gerar amostras para as distribuições que precisamos de usar.

Devido ao elevado número de geradores necessários na geração de chamadas de população finita o número de geradores de números aleatórios independentes disponíveis na linguagem de simulação (32 por omissão) foi aumentado para 4096, alterando o código fonte do OMNeT++ (nas definições usadas pelo núcleo do simulador). O código do OMNeT++ possui uma ferramenta geradora de sementes que permite escolher sementes iniciais para os geradores, separadas entre si de um número fixo (escolhido pelo utilizador) de forma a que os ciclos dos geradores não coincidam. Como o ciclo do gerador é de 2^{31}-2, mesmo com mais geradores podemos separar as sementes entre si de 100000 valores sem problemas. Embora tal não fosse estritamente necessário, procedeu-se também a uma alteração no código da ferramenta geradora de sementes, de forma a tornar mais rápida a geração de sementes em certos casos.

5.3 Componentes gerais para todos os encaminhamentos

O simulador foi construído de modo a permitir com facilidade simular sistemas com uma grande diversidade de características, para isto contribuiu a forma modular estruturada que foi usada na implementação, tirando partido da orientação a objectos permitida pela linguagem C++.

Na figura seguinte apresenta-se um diagrama com as principais classes do modelo de simulação que foi construído, designadamente classes `controlador`, `router`, `stats`, `gerador` e `central`. As classes que correspondem ao processo `gerador` e ao processo `central` são respectivamente as classes `gerador` e `central`. As restantes classes, que correspondem aos componentes controlador, encaminhador e estatísticas são respectivamente as classes `controlador`, `router` e `stats`. Estas últimas são substituídas por suas especializações para implementar cada método de encaminhamento em particular, enquanto as primeiras fornecem suporte a todos os tipos de métodos de encaminhamento, como referido anteriormente.

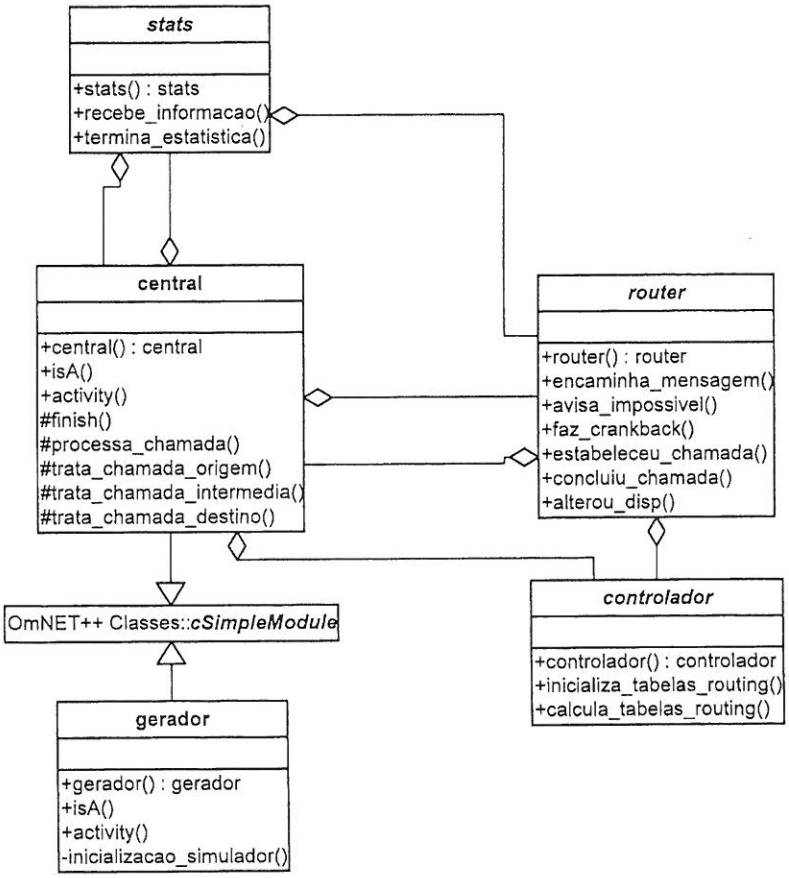

Fig. 17 – Diagrama das classes principais do modelo.

Tanto a classe `gerador` como a classe `central` são classes que correspondem a módulos simples definidos no modelo, derivadas por isso de uma classe (`cSimpleModule`) da biblioteca de classes de simulação do OMNeT++. A classe `gerador` e a classe `central` herdam deste algumas propriedades, de entre as quais as funções `activity` e `finish`, que redefinem.

O código das funções `activity` dos vários processos (`central` e `gerador`) do modelo são executados em paralelo, desde o início da simulação até ao fim da simulação. É portanto nestas funções, de cada processo, que está definido o código que corresponde a todas as operação que cada processo deve realizar durante a simulação.

O processo `central` possui uma função (`finish`) que é executada automaticamente quando a simulação termina. Foi por esta razão o lugar escolhido para pedir o registo dos resultados da simulação em ficheiros.

Como os funcionamentos dos processos `central` e `gerador` foram descritos na secção anterior, não se vai aqui descrever a actividade que os processos realizam durante toda a simulação.

O programa de simulação (incluindo o que diz respeito aos métodos de encaminhamento) foi construído a partir de vários tipos de componentes. São eles os componentes comuns a todos os tipos de encaminhamento e os componentes específicos de cada método de encaminhamento. Os componentes comuns a todos os tipos de encaminhamento são, para além do `gerador` e da `central`, os designados de componentes base e ainda outros componentes (que são usados para funções específicas de suporte aos componentes base). Os componentes específicos de cada método de encaminhamento são os componentes derivados dos componentes base e os componentes auxiliares.

5.3.1 Componentes base

Estes componentes fornecem as funções necessárias para o encaminhamento das chamadas e para o estudo do seu comportamento, sendo estas funções chamadas pelo processo `central`. Cada um dos componentes (controlador, encaminhador e estatísticas) deve ser redefinido para cada método de encaminhamento, ou mais precisamente, "sobreposto" por outro código que redefine algumas funções destes, através do polimorfismo permitido pela orientação a objectos. Esta sobreposição é conseguida na linguagem utilizada, C++, através do processo de herança. Assim são definidos novos componentes (por exemplo `controlador_dar`) que sendo derivados da classe base (neste caso `controlador`) herdam destas um conjunto de propriedades e redefinem um conjunto de métodos, de forma a implementar o comportamento desejado.

5.3.1.1 Controlador

O controlador vai modelar o comportamento do processador (centralizado ou distribuído) nos métodos de encaminhamento que o usam. O controlador vai obter informação de variáveis de estado e de acordo com o resultado do processamento dessa informação resultará a actualização da forma de encaminhamento das chamadas. O controlador tem essas funções apenas para os métodos de encaminhamento onde haja necessidade de actualizar as tabelas de encaminhamento. Nestes casos a actualização ocorre durante toda a simulação, com intervalos mais ou menos longos, dependendo da frequência com que é necessário actualizar a informação a utilizar no encaminhamento das chamadas.

O controlador fornece também as funções necessárias para inicializar as tabelas de encaminhamento, o que ocorre apenas uma vez, no início da simulação.

5.3.1.2 Encaminhador

A função principal do encaminhador é indicar à `central` o modo como ela deve tentar encaminhar uma chamada que tenha lá chegado. Isto é, cada `central` (origem ou intermédia) espera receber a identificação da `central` (intermédia ou destino) para onde deve enviar a chamada e possivelmente também o caminho completo que a chamada deve seguir até à `central` destino; se a identificação que recebeu não corresponder a uma `central` válida então é porque o encaminhador não conseguiu determinar uma `central` para onde seja possível enviar a chamada de modo a que ela chegue ao destino.

Além desta o encaminhador deve desempenhar outras funções: quando a chamada se encontra numa `central` intermédia e se verifica que não é possível encaminhar a chamada pelo caminho que estava a ser tentado, a `central` solicita o encaminhador para que lhe seja indicado se a chamada pode tentar outro caminho alternativo; se o encaminhador indicar que ainda pode ser tentado outro caminho alternativo a chamada deve retornar por *crankback* para a `central` origem para de seguida tentar outro caminho alternativo; caso contrário, se o encaminhador indicar que não pode ser tentado outro caminho alternativo a chamada deve retornar para a `central` origem como chamada que não foi possível estabelecer.

O encaminhador pode ser solicitado pela `central` onde a chamada teve origem, como já referido, mas pode também ser chamada por uma `central` intermédia que seja usada pela chamada no seu encaminhamento. Esta segunda possibilidade ocorre quando a chamada chega a uma `central` intermédia sem conhecer o caminho que deve seguir, isto é, não sabe para que `central` deve seguir para chegar ao destino.

5.3.1.3 Estatísticas

O objectivo central do modelo de simulação implementado é a obtenção de parâmetros estatísticos que exprimem os resultados do seu comportamento ao longo de cada corrida do simulador.

O componente estatísticas fornece as funções de recolha de informação (chamadas a cada momento pelos processos `central`), e algumas funções de consolidação de informação (chamadas no fim da simulação para apresentar a informação consolidada). Outras análises aos dados da simulação podem também ser realizadas no exterior do simulador.

O componente estatísticas, solicitado pelo processo `central`, permite armazenar vários valores e com esses valores produzir e registar os resultados da simulação em ficheiros. Os valores

armazenados são actualizados pelo componente estatísticas sempre que ocorre uma das acções seguintes:

- Início de uma chamada (sendo a função chamada pela `central` origem);
- Uma chamada foi concluída com sucesso (sendo a função chamada pela `central` destino);
- Uma chamada foi concluída sem sucesso (sendo a função chamada pela `central` origem);
- Uma chamada vai tentar um caminho alternativo após ter ocorrido *crankback* (sendo a função chamada pela `central` origem);
- Uma chamada foi concluída sem sucesso depois de ter ocorrido *crankback* (sendo a função chamada pela `central` origem);
- Ocupação de circuitos de um ramo (sendo a função chamada pela `central` origem e pela `central` intermédia);
- Libertação de circuitos de um ramo (sendo a função chamada pela `central` intermédia e pela `central` destino);
- Um ramo foi tentado sem sucesso (sendo a função chamada pelo `router` associado à `central`).

5.3.2 Outros componentes

Componentes de apoio chamados pelos processos `central` e/ou pelo processo `gerador`, armazenam informação, fornecem informação e geram os dados manipulados pelo sistema.

Nos dois diagramas das figuras seguintes, figura 18 e figura 19, estão representadas as classes auxiliares que foram utilizadas no modelo. As classes que correspondem aos componentes caminhos e armazém de dados são respectivamente as classes com os nomes `caminhos` e `arm_dados` e estão representadas na figura 18. As classes que correspondem aos componentes reserva e tráfego são respectivamente as classes com os nomes `rr` e `classe_trafego` e estão representadas na figura 19. Repare que nessa figura aparece também um tipo de dados designado por `MPTR` (usado pelo processo `gerador` na declaração da variável que armazena cópias das mensagens, do tipo "chamada a estabelecer", para os tráfegos de população finita) que permite aceder uma `cMessage`.

5.3.2.1 Caminhos

Quando se pretende armazenar um caminho este é armazenado de forma codificada. O componente caminhos é quem trata de fazer a codificação e descodificação dos caminhos que estão armazenados numa chamada e na tabela de encaminhamento.

5.3.2.2 Armazém de dados

O componente armazém de dados guarda a informação comum a todos os processos sendo esta informação partilhada por todos os processos (utilização de memória partilhada).

A informação armazenada neste componente é: o número de circuitos disponíveis e o número de circuitos existentes para cada feixe da rede, o número de circuitos reservados e o número de circuitos planeados em cada feixe da rede para cada uma das classes. O número de circuitos planeados para uma classe em cada ramo ou por outras palavras a largura de banda mínima garantida para uma classe quando existe bloqueio é uma fracção dos circuitos existentes no ramo que é atribuída a essa classe quando as condições de bloqueio, das chamadas dessa classe, o justificam.

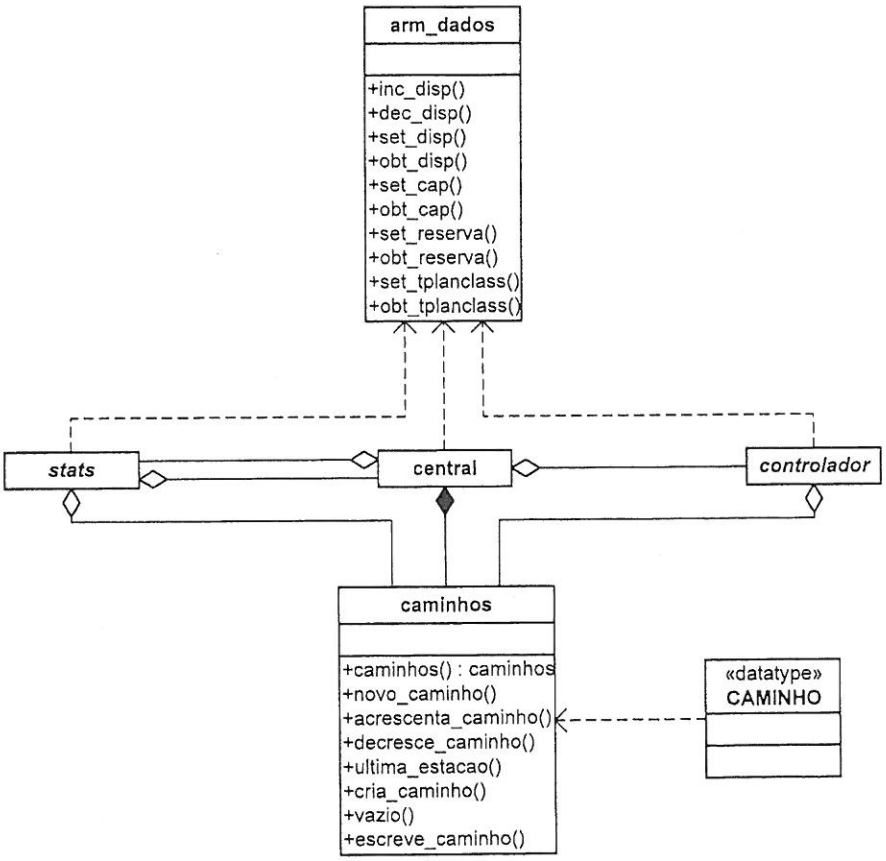

Fig. 18 – Diagrama das classes auxiliares do modelo, usadas pelo processo `central`.

5.3.2.3 Tráfego

O componente tráfego guarda a informação relativa a uma classe de tráfego.

Ao componente tráfego corresponde uma estrutura chamada `classe_trafego`. Nesta estrutura é armazenado, para cada classe, o número de circuitos necessários às chamadas, em cada ramo, a duração (tempo de serviço) das chamadas e a indicação de se o tráfego é ou não de população finita.

5.3.2.4 Reserva de números aleatórios

O componente reserva de números aleatórios é utilizado para fazer a gestão dos geradores de números aleatórios independentes. Como são usados um grande número de geradores durante a simulação, foi necessário implementar este código para facilitar a tarefa de atribuição de geradores, de forma a garantir que o mesmo gerador não seja utilizado em mais do que uma situação de geração de números, causando dependências inadvertidas entre geradores supostamente independentes. A utilização de reserva de geradores, garante assim a atribuição de geradores de números aleatórios independentes em situações que o requeiram. A classe que corresponde a este componente é a classe `rr`.

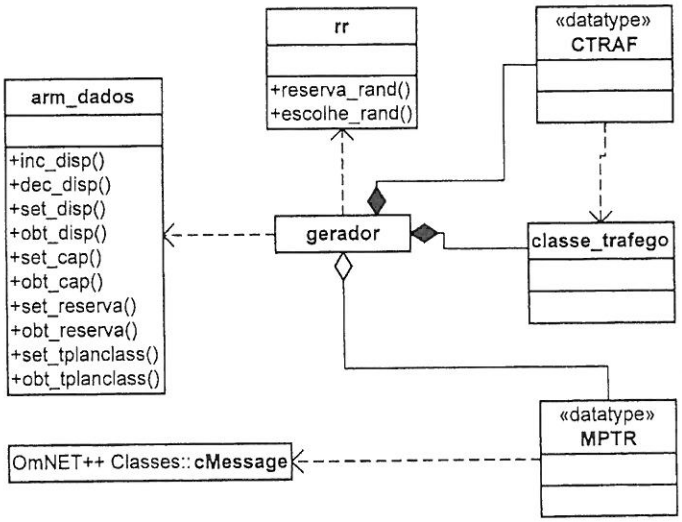

Fig. 19 – Diagrama das classes auxiliares do modelo, usadas pelo processo `gerador`.

5.4 Componentes específicos de cada encaminhamento

Foram criados componentes específicos de cada tipo de encaminhamento implementado:

- Directo
- FAR (Fixed Alternative Routing)
- DAR (Dynamic Alternative Routing)
- DCR (Dinamically Controlled Routing)
- RTNR (Real Time Network Routing)

Entretanto iremos referir aspectos particularmente importantes relativos à forma como foram implementados alguns destes métodos, nomeadamente o DAR, o DCR e o RTNR, envolvendo algumas diferenciações em relação à descrição original apresentada no relatório [Jorge00] sobre métodos de encaminhamento dinâmicos.

5.4.1 DAR

Este método foi implementado de forma a permitir o encaminhamento simultâneo de vários tipos de tráfego. Para o conseguir fez-se uma generalização do método de encaminhamento original. Passando o algoritmo de encaminhamento a ser o seguinte: quando uma chamada, de uma determinada classe chega ela é oferecida primeiro ao caminho directo, se este não tiver capacidade disponível suficiente para a chamada em questão, esta transborda para o caminho alternativo, de dois feixes, actualmente seleccionado para as chamadas dessa classe entre essa origem/destino. O caminho alternativo permanece inalterado a menos que a chamada seja bloqueada nele. Neste caso um novo caminho alternativo é seleccionado aleatoriamente para as chamadas subsequentes, da classe em questão. O caminho alternativo está sujeito a reserva de circuitos para o encaminhamento directo. As diferenças em relação ao algoritmo original são: a necessidade de verificação, sempre que uma chamada chega, da classe a que pertence para determinar a quantidade de recursos de que precisa e a necessidade de armazenar tantos caminhos alternativos, por par origem/destino, como o número de classes de tráfego que existam. A implementação desenvolvida para este algoritmo tem como caso particular o algoritmo do método de encaminhamento DAR descrito no relatório [Jorge00] (onde era considerado apenas o encaminhamento de um tipo de tráfego).

Na implementação feita, além do mecanismo de protecção de tráfego normal, descrito no método de encaminhamento original, foi implementado um segundo mecanismo. Os dois tipos de mecanismo de protecção do tráfego normal implementados foram:

- Reserva de Circuitos Fixa - um número de circuitos, fixo, é reservado, em cada feixe, para ser utilizado no encaminhamento directo, tal como descrito no método de encaminhamento original.
- Reserva de Circuitos Dinâmica - um número de circuitos, variável com o tempo, é reservado, em cada feixe, para ser utilizado apenas no encaminhamento directo de chamadas, quando o número de circuitos livre for igual ou inferior a esse número. O número de circuitos reservados em cada feixe, para cada instante, é calculado usando uma expressão simplificada, obtida a partir da expressão para o cálculo do número de circuitos reservados no método de encaminhamento RTNR, tal como descrito no relatório [Jorge00]. A expressão simplificada é:

$$RBWtraf_k = \sum_{i=VN_1}^{VN_N} R_k^i \times r_i \qquad (1)$$

onde a largura de banda total reservada $RBWtraf_k$, é calculada como o somatório dos níveis de reserva para cada classe R_k^i multiplicado pela largura de banda média necessária para essa classe, r^i. Este somatório soma desde a classe de serviço considerada até à classe de serviço que necessite de maior largura de banda.

Sempre que se pretende simular este método deve-se escolher qual dos dois mecanismos se pretende utilizar.

Se for escolhida reserva de circuitos dinâmica além do mecanismo normal de protecção do tráfego foi incorporado um mecanismo de protecção de modo a permitir uma utilização equilibrada dos recursos pelos vários serviços. O tráfego das várias classes está então sujeito a um mecanismo de controlo de acesso ao caminho directo. O mecanismo de controlo de acesso implementado foi o descrito no método de encaminhamento RTNR.

O algoritmo implementado foi por isto tornado ainda mais geral do que o indicado pela descrição inicial. Se for escolhida reserva de circuitos dinâmica, o algoritmo tem que determinar periodicamente os valores para as reservas. E além disso, para encaminhar uma chamada no caminho directo não basta verificar se possui capacidade disponível suficiente para ela, devido ao mecanismo de controlo de acesso referido anteriormente.

5.4.2 DCR

Este método de encaminhamento foi implementado tal como definido no relatório [Jorge00]. No entanto o número de circuitos gastos por uma chamada, em cada feixe do caminho, pode ser um qualquer valor que se escolha, não está portanto limitado a chamadas que ocupem apenas um circuito. Pode ser escolhido um de três tipos de reserva de circuitos, são eles:

- Reserva fixa;
- Reserva dinâmica, conseguida utilizando a expressão:

  $$m_s(t) = g \times a_s' \qquad (2)$$

  onde $m_s(t)$ corresponde à reserva de circuitos no instante t no grupo de circuitos s, g corresponde a um factor de escala e a_s' corresponde à quantidade corrente de tráfego, oferecido pela primeira vez, que transborda do ramo s.
- Reserva dinâmica, como definido no método de encaminhamento DAR, pela expressão 1.

5.4.3 RTNR

Recomenda-se que, dada a complexidade deste método de encaminhamento, antes de analisar a descrição da implementação que se segue, seja revista a apresentação feita no relatório [Jorge00] do método de encaminhamento RTNR com várias classes de tráfego.

O RTNR foi implementado de acordo essa descrição. No entanto é permitido, como já aconteceu nos métodos DCR e DAR, escolher outro tipo de mecanismo de protecção do tráfego normal. O tipo permitido, além do definido no relatório [Jorge00], é a reserva de circuitos dinâmica como a utilizada na implementação do DAR, dada pela expressão 1. Note que este tipo de reserva, como referido no DAR, é uma simplificação ao tipo de reserva dinâmica definida no método de encaminhamento RTNR original.

5.5 Estrutura da global da implementação (Diagrama de classes)

O diagrama de classes, da figura 20, serve para dar uma visão geral de todas as classes que foi necessário implementar e também algumas das ligações que foram criadas entre elas.

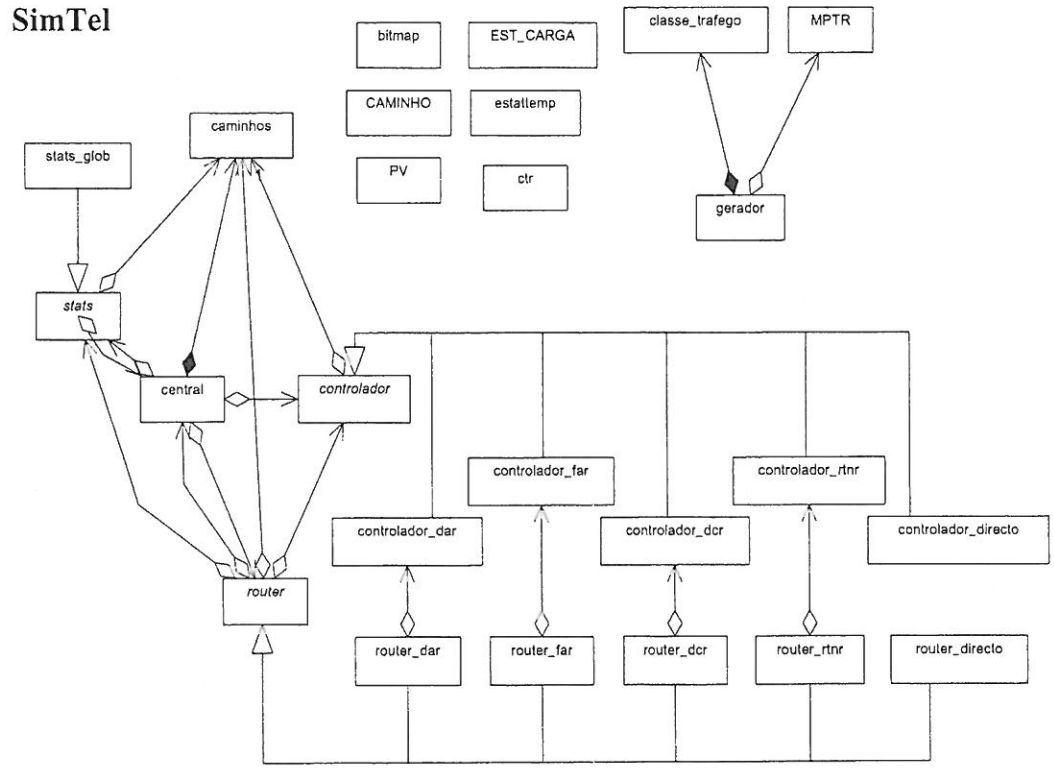

Fig. 20 - Diagrama de todas as classe definidas para a implementação.

Para concluir a descrição feita em relação à implementação dos métodos de encaminhamento vamos agora fazer uma analogia entre as classes principais implementadas no modelo com os componentes do sistema. Assim sendo, a classe controlador desempenha as mesmas funções que o processador no sistema. No caso dos métodos isolados ou distribuídos temos um controlador associado a cada `central` desempenhando as funções de processador local. No

caso dos métodos centralizados é apenas um dos controladores que possui funcionalidades desempenhando esse as funções de processador central. A classe `central` juntamente com a classe `router` desempenham o papel das centrais de comutação no sistema. A determinação do caminho para encaminhar um chamada é da competência do `router`. A função principal da classe `central` é levar as chamadas, que recebe, até ao destino se possível, para isto recorre às funções da classe `router`. Como pretendíamos ter a possibilidade de seleccionar um de vários algoritmos na determinação dos caminhos, derivamos, a partir da classe `router`, a classe `router_directo`, a classe `router_far`, a classe `router_dar`, a classe `router_dcr` e a classe `router_rtnr`. Cada uma destas classes implementa um dos algoritmos de encaminhamento. Como a informação que cada algoritmo utiliza e o modo como é determinada difere de algoritmo para algoritmo derivamos também, a partir da classe controlador, a classe `controlador_directo`, a classe `controlador_far`, a classe `controlador_dar`, a classe `controlador_dcr` e a classe `controlador_rtnr`.

No método de encaminhamento Directo não é preciso qualquer processamento de informação pois o encaminhamento é baseado apenas na capacidade disponível dos caminhos directos. Neste caso os controladores não desempenham qualquer função.

No método de encaminhamento FAR o controlador tem funções apenas na fase de inicialização (carregamento das tabelas de encaminhamento), pois a partir daí a informação a usar no encaminhamento permanece inalterada.

O método de encaminhamento DAR é isolado logo os controladores associados a cada `central` fazem a actualização da informação, a usar no encaminhamento, baseada apenas na informação armazenada nele.

No DCR, como é um método de encaminhamento centralizado, só foram atribuídas funcionalidades a um dos controlador.

Por último o RTNR é um método de encaminhamento distribuído logo temos um controlador associado a cada `central` que utiliza a informação dos outros controladores.

Por questões de simplificação ou para evitar a duplicação de informação não se manteve, por vezes, a analogia entre o local onde a informação a usar no encaminhamento é armazenada no sistema, e o local onde é armazenada no modelo; contudo os resultados não são alterados com isso. Isto passa-se tanto no RTNR como no DCR.

6 *Como executar uma simulação*

6.1 Entradas para uma simulação

A principal entrada da simulação consiste nos ficheiros que definem as variáveis descritivas dos componentes da rede (isto é, a topologia da rede). As entradas restantes consistem nos parâmetros do modelo e da simulação.

6.1.1 Configuração da rede (Ficheiro da topologia)

A configuração da rede de comutação por circuitos, que se pretende usar, isto é, o número de circuitos do feixe que liga quaisquer duas centrais é especificado num ficheiro de texto a que chamaremos Ficheiro da Topologia.

6.1.2 Ficheiro de configuração (Ficheiro *ini*)

É permitida grande flexibilidade nos vários algoritmos de encaminhamento implementados em parte pelo uso de parâmetros. O ficheiro de configuração pode descrever várias corridas de simulação, em que cada corrida pode ter os seus parâmetros. No ficheiro de configuração podem também ser seleccionadas as sementes para os geradores de números aleatórios.

Para geração do ficheiro de configuração foi desenvolvido em Perl uma ferramenta chamada `gerador_ini.pl`, que constrói um ficheiro *ini* a partir um ficheiro mais simples. Agradeço por esta implementação ao Eng. Paulo Melo.

6.2 Os vários passos para executar uma simulação

O programa de simulação foi desenvolvido em ambiente Linux. É tendo por base este ambiente que vão ser descritos os passos para executar a simulação.

Depois de termos o modelo construído, isto é, depois de termos feito a descrição da topologia do modelo (em linguagem NED) e de termos implementado em C++ os módulos simples estamos em condições de criar o programa de simulação executável. O primeiro passo para criar o programa executável é utilizar o compilador NEDC para compilar o ficheiro *ned* e então obter o ficheiro correspondente em C++. O que é feito automaticamente ao executar o comando `make` referido a seguir.

O passo seguinte é compilar todos os ficheiros C++ (ficheiro ned compilado e ficheiros correspondente à implementação dos módulos simples) e fazer a ligação deles com o núcleo de simulação, com a biblioteca de classes do OMNeT++ e com a biblioteca do interface com o utilizador. Deve começar-se por executar o comando:

```
makemake -u cmdenv
```

A utilização da opção `-u` é que torna possível a especificação do interface com o utilizador (cmdenv no exemplo) que se pretende usar. Se a opção `-u` não for usada o interface por omissão é o Tkenv.

O nome do programa executável que será produzido é o nome da directoria onde temos os códigos fontes, se pretendermos que o programa executável fique com outro nome deve utilizar-se a opção `-o`. Se o comando executado for por exemplo:

```
makemake -o simtel
```

Neste caso o nome do programa executável será `simtel`.

Ao executar o comando `makemake` foi produzido o ficheiro `makefile` após o que deve ser executado o comando `makedepend` e por último o comando `make`. Com o comando `make` é que será efectuada a compilação e ligação das bibliotecas de rotinas e então produzido o programa executável. Note que se depois de executado o comando `make` houver necessidade de alterar o código de algum ficheiro ao executar de novo o comando `make` só serão recompilados os ficheiros dependentes do ficheiro alterado e o próprio.

Para poder executar o programa de simulação é necessário ter o ficheiro de configuração (denominado de ficheiro *ini*) construído.

Para aliviar a fastidiosa tarefa de construção dos ficheiros *ini*, para muitas replicações de uma simulação, foi desenvolvido um programa, em Perl, chamado `gerador_ini.pl`, que constrói um ficheiro *ini* a partir um ficheiro mais simples. A saída do programa é o ficheiro de saída standard; para obter um ficheiro *ini* a saída desse programa deve ser redireccionada.

O ficheiro de configuração *ini* deve ter o nome `omnetpp.ini`, caso contrário é necessário indicar o nome do ficheiro de configuração, ao executar o programa de simulação, utilizando para tal a opção `-f`. Se, por exemplo, o ficheiro de configuração e o programa executável tiverem o nome `teste.ini` e `simtel`, respectivamente, para correr o programa de simulação devia-se executar o comando:

```
simtel -f teste.ini
```

Após o arranque da aplicação, é apresentada a janela inicial do interface gráfico (figura 21).

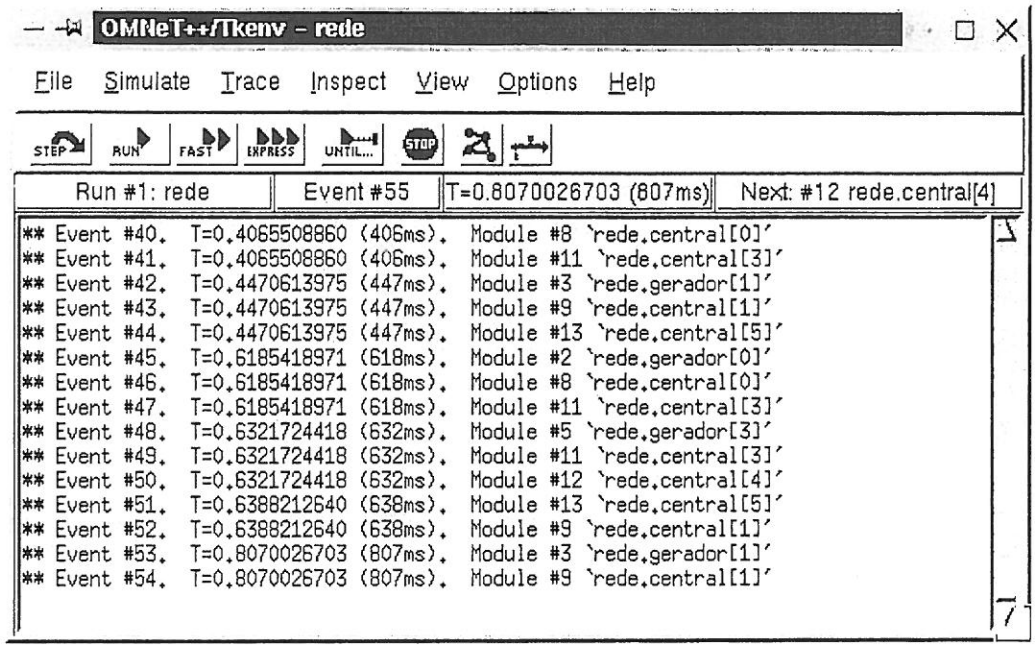

Fig. 21 - Janela principal da aplicação.

Este interface permite executar a simulação e verificar os valores presentes nos diversos componentes do modelo a simular. Para além dos menus, que permitem efectuar diversas operações, possuí também uma barra de botões, que podem ser usados para aceder mais facilmente às funções mais usuais.

Assim, nesta barra podemos encontrar as opções de controlo da simulação que permitem a execução passo a passo (*step*), a velocidade baixa com total animação gráfica (*run*), a velocidade mais rápida com actualizações de animação pouco frequentes (*fast*) ou mesmo execução sem animação (*express*). É também possível executar a simulação até à ocorrência de determinada condição (*until*). Seja qual for a velocidade de execução seleccionada, pode ser interrompida a simulação através do botão de parar (*stop*), podendo depois prosseguir-se com a simulação a partir desse ponto.

Na barra de botões estão ainda presentes opções para a visualização da rede (que permite a representação da rede graficamente, como apresentado na figura 23), e para a visualização da lista de mensagens escalonadas (que corresponde aproximadamente à lista de acontecimentos futuros). O corpo da janela principal é também o local onde as mensagens de aviso (enviadas através do interface *ev*) e as marcas de acontecimento são apresentadas por omissão, embora seja possível redireccionar as mensagens geradas por cada módulo para janelas individuais.

Aquando do início da simulação, o interface permite-nos a escolha da corrida a executar, de entre todas as que estejam predefinidas no ficheiro *ini*.

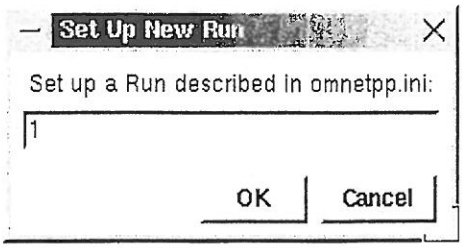

Fig. 22 - Arranque da simulação, escolha da corrida a executar.

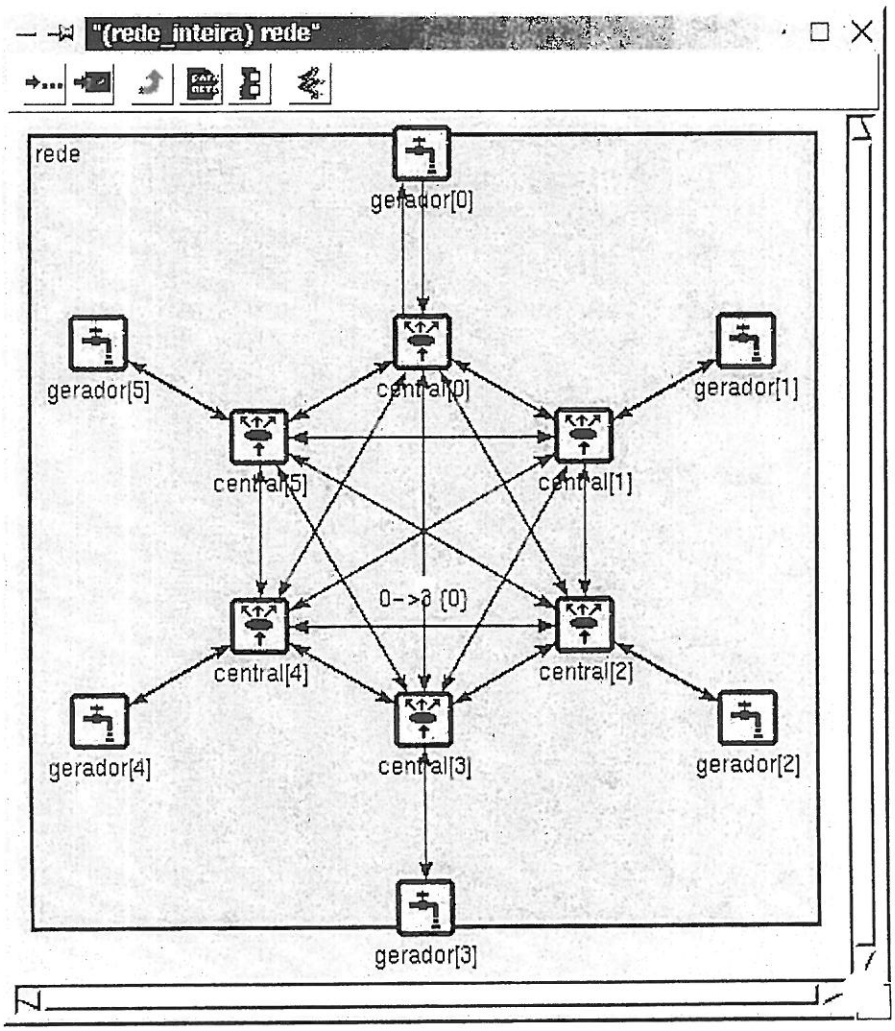

Fig. 23 - Um exemplo de rede a simular.

A apresentação gráfica da rede a simular, para além de permitir a visualização animada das trocas de mensagens entre módulos (notar na figura a mensagem a ser enviada da `central` 0 para a `central` 3) permite-nos também acesso às informações específicas sobre cada um dos módulos e componentes nela representados, bastando para tal efectuar um duplo clique sobre o elemento que se pretenda observar, quer durante o período de corrida da simulação, quer ela tenha sido interrompida (mas não antes do inicio da corrida ou após o término da mesma).

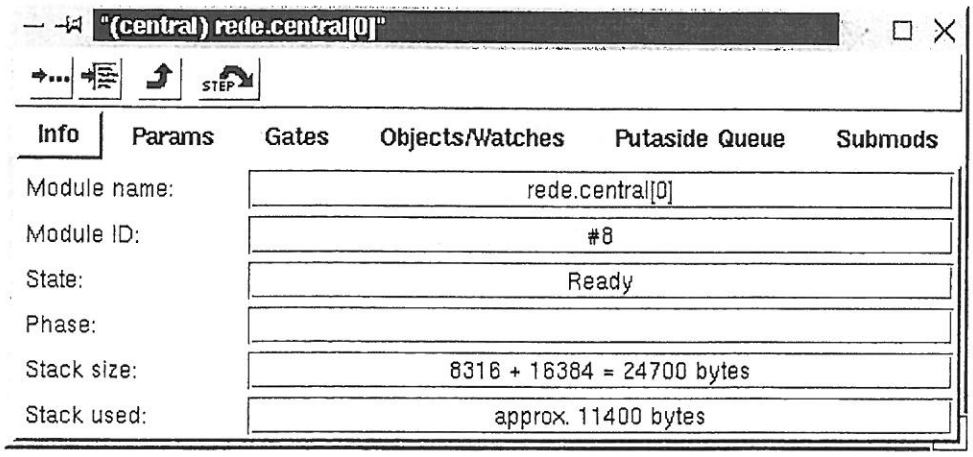

Fig. 24 - Observação de um módulo da simulação – informação genérica.

Cada módulo presente na simulação pode ser analisado durante a mesma quer em função das suas características básicas (figura 24), quer através dos valores dos seus parâmetros (que podem ser alterados durante a execução, figura 25), quer através da visualização das suas portas de saída e entrada (figura 26). Ainda outros valores podem ser analisados durante a execução (é possível definir valores a serem monitorizados (*watches*), e observar submódulos ou a fila auxiliar de mensagens do módulo, mas neste trabalho tal capacidades não foram utilizadas).

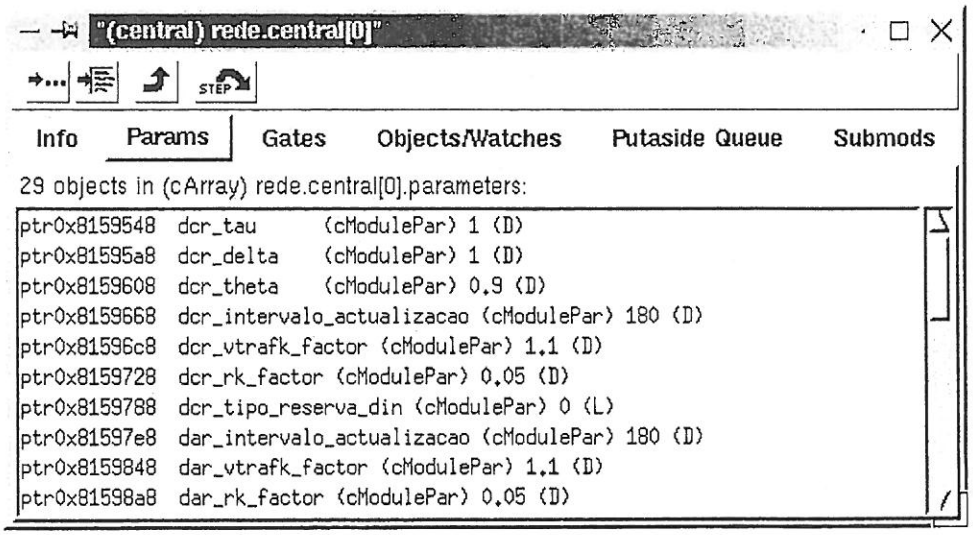

Fig. 25 - Observação de um módulo da simulação – parâmetros do módulo e seus valores.

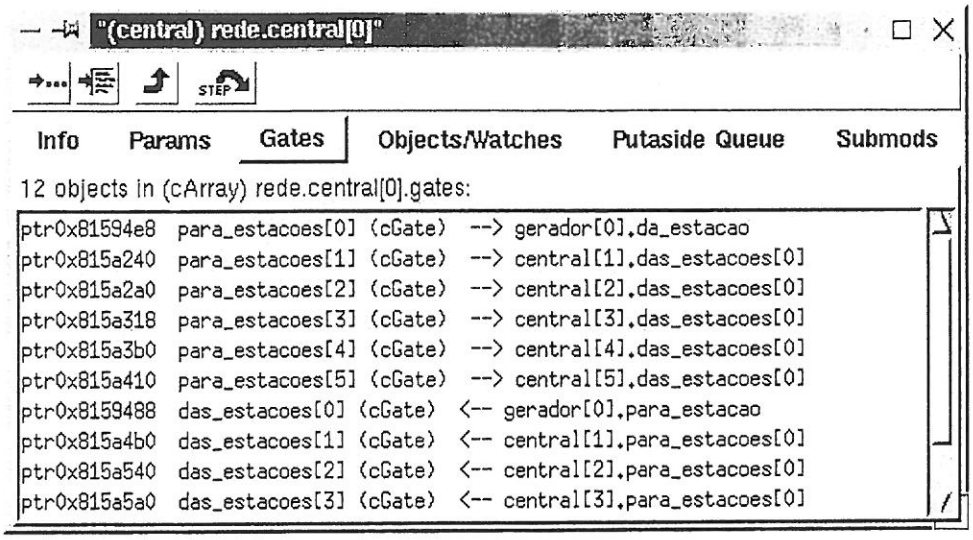

Fig. 26 - Observação de um módulo da simulação – portas de ligação entre módulos.

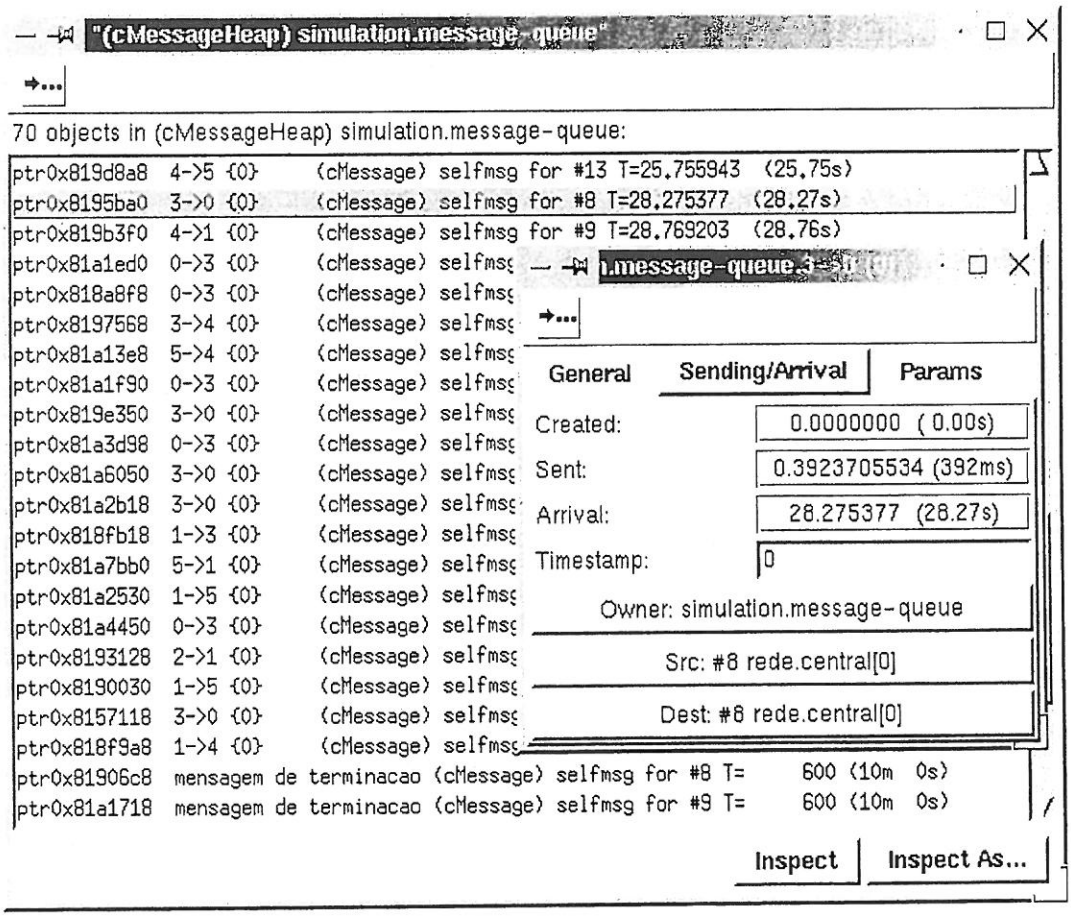

Fig. 27 - Execução da simulação – lista de mensagens, com pormenor de uma mensagem.

A fila de mensagens escalonadas (figura 27) permite-nos observar a todo o momento a totalidade das mensagens escalonadas pelo sistema. No sistema aqui descrito, as mensagens geradas podem ser mensagens correspondentes a chamadas telefónicas, ou mensagens de controlo (como as mensagens de terminação). É possível observar em maior pormenor as

mensagens trocadas pelo sistema, sendo possível observar as suas características genéricas, informação sobre as sua chegada ou partida (como é visível na figura 27) ou ainda os diversos parâmetros que tenham sido associados à mesma (como se pode ver na figura 28).

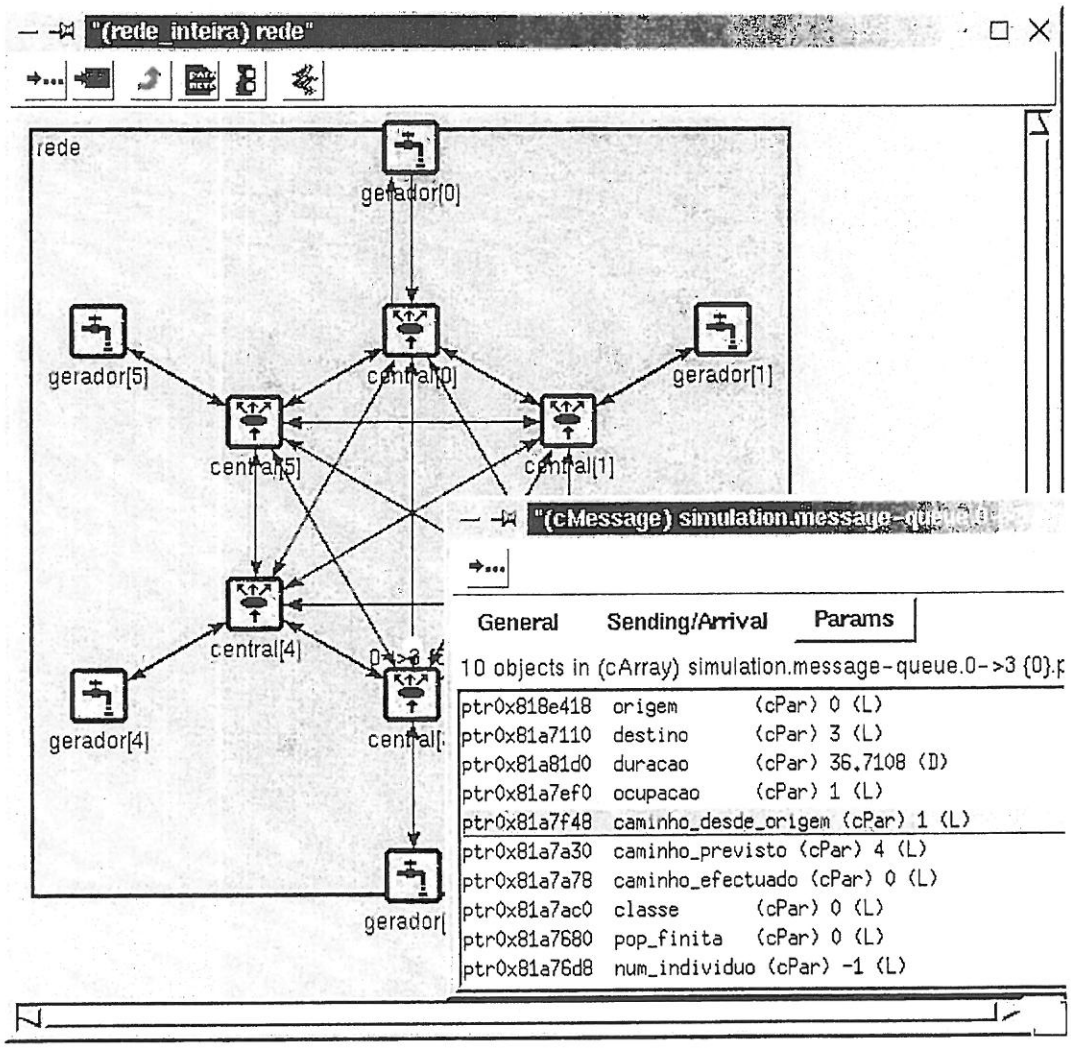

Fig. 28 - Execução de uma simulação – o envio de uma mensagem entre centrais, e os parâmetros que descrevem essa mensagem.

6.3 Como analisar os resultados produzidos

Os valores armazenados num dos ficheiros de saída (ficheiro escalar) foram-no através da função `recordscalar` (função já referida e que faz parte da biblioteca de classes de simulação do OMNeT++) e por essa razão esse ficheiro é formado por várias linhas tendo um formato adequado à importação para uma base de dados ou folha de cálculo e posterior tratamento nela. No outro ficheiro (ficheiro de tabelas) a informação é apresentada em forma de matriz o que torna a sua visualização simples.

Bibliografia

[Banks96] Jerry Banks, John S. Carson e Barry L. Nelson, "Discrete-Event System Simulation", Prentice Hall, New Jersey, 2nd Edition, 1996

[Larman98] Craig Larman, "Applying UML and patterns: an introduction to object oriented analysis and design", Prentice-Hall, USA, 1998

[Law91] Averill M. Law e W. David Kelton, "Simulation Modeling & Analysis", McGraw-Hill, Inc., U.S.A., 2nd Edition, 1991

[Law94] Averill M. Law e Michael G. McComas, "Simulation Software for Communications Networks: The State of the Art", IEEE Communications Magazine, Vol. 32 Nº 3, pp 44-50, Março de 1994

[Jorge99] Luísa M. G. Jorge e José F. Craveirinha, "Resenha sobre Métodos de Encaminhamento Dinâmicos em Redes Inter-Centrais", Research Report NT-N10, INESC-Coimbra, Outubro de 1999

[Rational00] Rational, "Rational Rose, Visual Modelling, UML, Object Oriented, Component-Based Development", recolhido em Julho de 2000 em http://www.rational.com/products/rose/, Julho de 2000

[Stroustroup97] Bjarne Stroustroup, "The C++ Programming Language", Addison Wesley, USA, 1997

[Varga00] András Varga, "OMNeT++ Discrete Event Simulation System - User Manual", recolhido em Julho de 2000 em http://www.hit.bme.hu/phd/vargaa/opp-docs/usman.htm, Julho de 2000