Eduardo de Mello Garcia

SS7Sim: Simulador para o Sistema de Sinalização SS7

São José – SC

Março / 2012

Eduardo de Mello Garcia

SS7Sim: Simulador para o Sistema de Sinalização SS7

Orientador:
Prof. Marcos Moecke, Dr.Eng.

Co-orientador:
Prof. Emerson Ribeiro de Mello, Dr.Eng.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Março / 2012

Monografia sob o título *"Ss7Sim: Ambiente Para Simulação de Sinalização SS7"*, defendida por Eduardo de Mello Garcia e aprovada em 7 de março de 2012, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Marcos Moecke, Dr. Eng.
Orientador

Prof. Emerson Ribeiro de Mello, Dr. Eng.
Coorientador

Prof. Marcelo Maia Sobral, Dr. Eng.
IF-SC

Prof. Fabio Alexandre de Souza, Msc.
IF-SC

Os conhecimentos mais proveitosos são os que mais tardiamente se adquirem.

Friedrich Nietzsche

Agradecimentos

Dedico meus sinceros agradecimentos àqueles que contribuíram para a realização deste trabalho. Agradeço especialmente a meus familiares e amigos, que estiveram ao meu lado em todos os momentos me apoiando e me motivando a seguir em frente. Agradeço também ao meu orientador e ao meu co-orientador por todas as suas contribuições para a realização deste trabalho.

Resumo

O SS7Sim é um projeto de código aberto que cria um ambiente de simulação para o sistema de sinalização SS7 para fins didáticos. No SS7Sim é possível simular a troca de mensagens SS7 envolvidas em uma chamada telefônica comum, constituída pelas etapas de estabelecimento, atendimento (opcional) e desligamento. O usuário interage com o sistema iniciando e encerrando chamadas, tornando enlaces inoperantes e visualizando informações carregadas pelas mensagens trocadas, assim como informações contidas nos nós e enlaces da rede. No SS7Sim foram implementadas as seguintes funções SS7: Sequênciamento de mensagens (MTP2), *Signaling Message Handling* (MTP3) e estabelecimento e desligamento de chamadas (ISUP).

No SS7Sim, toda a interação do usuário com o sistema é feita via interface gráfica. Para cada elemento da rede, existem menus que indicam ao usuário todas as ações possíveis. A visualização das informações contidas em mensagens, nós e enlaces é feita através de janelas *popup*. O SS7Sim foi desenvolvido utilizando a linguagem de programação java e a biblioteca gráfica Netbeans Visual Library, bem como o ambiente integrado de desenvolvimento NetBeans, tornando o simulador compatível com diferentes sistemas operacionais.

PALAVRAS-CHAVE: simulador de rede; sinalização SS7; ambiente didático.

Abstract

SS7Sim is an open source project wich creates a simulation environment for the SS7 signaling system networks for teaching purposes. In SS7Sim, it is possible to simulate the exchange of SS7 messages in a basic telephone call, formed by the steps: setup, answer (optional) and teardown. The user interacts with the system by stating and ending calls, setting links out of service and visualizing information carried by the exchanged messages, as well as information contained in the network's nodes and links. In SS7Sim, the following SS7 functions where implemented: message sequencing (MTP2), signaling message handling (MTP3) and call setup an teardown (ISUP).

In SS7Sim, all the user's interaction with the system is made by a graphic interface. For each network element, there are menus that indicate to the user all the possible actions. The visualiziation of information contained in messages, nodes and links is made through pupop windows. SS7Sim was developed using the java programming language and the graphic library NetBeans Visual Library, as well as the integrated development environment NetBeans, wich made the simulator compatible with different operating systems.

KEYWORDS: network simulator; SS7; learning environment.

Sumário

Lista de Figuras

1 Introdução

O Sistema de Sinalização nº 7 (SS7) é atualmente o principal protocolo para a transmissão de sinalização telefônica e informações de controle necessários aos serviços de telefonia (RUSSEL, 2006). As informações que trafegam na rede SS7 são utilizadas por diversos serviços conhecidos, como SMS, portabilidade numérica e serviços telefônicos de utilidade pública (polícia, bombeiros, ambulância, etc.), bem como gerenciamento da mobilidade e roaming em redes celulares (DRYBURGH; HEWETT, 2004)

O SS7 é dividido em níveis de protocolo de uma forma semelhante ao modelo de camadas usado nas redes de computadores, modelo OSI. Porém no SS7 temos apenas 4 níveis (ou camadas), ao invés dos 7 do modelo OSI. O SS7 pode trafegar pela rede TCP/IP utilizando as camadas 1 a 3 do modelo OSI; ou sobre a rede *Synchronous Digital Hierarchy* (SDH) utilizando para os três níveis inferiores os protocolos Message Transfer Part (MTP): MTP1, MTP2 e MTP3. Neste trabalho será considerado apenas o SS7 sobre SDH.

1.1 Motivação

Devido a sua importância o SS7 é um dos temas abordados dentro da área de conhecimento Telefonia nos cursos superiores da área de telecomunicações, especialmente no CST em Sistemas de Telecomunicações do IFSC.

Normalmente o ensino do SS7 se dá por meio de aulas expositivas, com o uso de recursos didáticos tais como textos e diagramas. A inclusão de uma abordagem prática aliada às técnicas didáticas vigentes poderia contribuir para um melhor entendimento do conteúdo por parte dos alunos. No entanto, notou-se que este conteúdo carece de uma ferramenta que proporcione tal abordagem. Este trabalho motivou-se então a propor uma solução para o seguinte problema:

Como tornar mais práticos os conceitos teóricos SS7 ensinados no curso, sem a existência dos equipamentos reais das operadoras de telefonia?

1.2 Contribuições deste trabalho

Neste trabalho foi desenvolvido um simulador didático para o Sistema de Sinalização SS7, com o objetivo de auxiliar o ensino do conteúdo SS7 em cursos da área de telecomunicações. A implementação foi realizada utilizando-se a linguagem de programação java e a biblioteca gráfica NetBeans Visual Library. O SS7Sim é capaz de simular parte das funções dos níveis de protocolo MTP2 e MTP3 e do protocolo ISUP. As funções implementadas são listadas a seguir.

- **Funções do protocolo MTP2**
  - **Sequenciamento de mensagens** - as mensagens implementadas no SS7Sim possuem os números de sequência *Forward Sequence Number* (FSN) e *Backward Sequence Number* (BSN).
- **Funções do protocolo MTP3**
  - **Discriminação** - é implementada nos nós das redes simuladas no SS7Sim a função de discriminação, que consiste em determinar se uma mensagem redebida pelo nó é destinada ao próprio nó receptor ou a outro nó.
  - **Distribuição** - a função de distribuição consiste em determinar de qual protocolo dos níveis 3 ou 4 é a informação carregada pela mensagem. Esta função é implementada de forma simplificada no SS7Sim, uma vez que apenas o as mensagens do SS7Sim carregam apenas informações do protocolo ISUP em seu campo de informação.
  - **Roteamento** - esta função é realizada sempre que a função de discriminação determina que a mensagem é destinada a outro nó. os nós STP do SS7Sim possuem tabelas de roteamento e implementam o encaminhamento de mensagens com base nestas tabelas.
- **Funções do protocolo ISUP**

  - **Estabelecimento e desligamento de chamadas** - O SS7Sim simula a principal função do protocolo ISUP, que é o estabelecimento e desligamento de uma chamada.

- **Implementação da interface gráfica para modelagem e acompanhamento do funcionamento de uma rede SS7.**

1.3 Organização do texto

O texto está organizado da seguinte forma: no Capítulo 2, é dada a fundamentação teórica necessária para a compreensão deste trabalho, o que abrange o sistema de sinalização SS7, conceitos de modelagem de um *software* orientado a objetos em UML, bem como conceitos básicos de programação orientada a objetos. No Capítulo 3 é detalhado o processo de implementação do simulador, desde a identificação das funções do SS7 a serem implementadas até a codificação das mesmas na linguagem java. No Capítulo 4 são mostrados os resultados obtidos, bem como detalhes do funcionamento do simulador e testes realizados em diferentes cenário. No Capítulo 5, são apresentadas as conclusões obtidas com a realização deste trabalho, bem como possíveis aprimoramentos no simulador SS7Sim, propostos como trabalhos futuros.

2 *Fundamentação Teórica*

Para o entendimento deste trabalho, há uma série de conceitos que devem ser antes esclarecidos. Estes conceitos englobam conceitos teóricos do SS7, como os níveis de protocolo e suas funções e a estrutura da rede, até conceitos referentes ao desenvolvimento de um *software* orientado a objetos, desde conceitos básicos de orientação a objetos até a modelagem em UML. A fundamentação teórica para o SS7Sim é apresentada a seguir.

2.1 O sistema de sinalização nº7 - SS7

O Sistema de Sinalização nº 7 é um padrão definido pela União Internacional de Telecomunicações (*International Telecommunication Union* - ITU). Apesar de ser um padrão internacional, o SS7 pode ter variantes regionais, como ocorre na América do Norte (padrão definido pela Telcordia) (DRYBURGH; HEWETT, 2004). Inicialmente o SS7 era utilizado apenas para acesso a bases de dados. Como obteve sucesso com esta aplicação, o SS7 passou por um processo de expansão e o estabelecimento e desligamento de chamadas passou a ser sua principal função. O SS7 tem, também, outros usos, como portabilidade numérica, *roaming* em redes celulares, números de emergência (polícia, bombeiros, etc.), entre outros (CISCO, 2012).

De forma semelhante ao modelo OSI, o SS7 é dividido em diferentes níveis de protocolo. No SS7 existem quatro níveis de protocolo, os três níveis inferiores são chamados de *Message Transfer Part* (MTP), estes níveis correspondem às camadas física, enlace e rede do modelo OSI. O quarto nível contém os protocolos chamados de *User Parts* e *Application Parts*, dentre estes protocolos, destacam-se o *Integrated Services Digital Network - ISDN - User Part* (ISUP), *Transaction Capabilities Application Part* (TCAP) e *Signaling Connection Control Part*(SCCP). Existe um quarto protocolo neste nível superior do SS7, o TUP. No entanto, este protocolo é pouco usado, sendo quase integralmente substituído

pelo ISUP (DRYBURGH; HEWETT, 2004). Os níveis de protocolo e outros conceitos do SS7 serão detalhados a seguir. No entanto, apenas parte destas funções foram implementadas no simulador.

2.1.1 Tipos de mensagens

As mensagens enviadas na rede SS7 são chamadas de *signal units*. Existem basicamente três tipos de mensagens no SS7: a *fill-in signal unit* (FISU), a *link status signal unit* (LSSU) e a *message signal unit* (MSU).

A mensagem FISU é enviada quando a rede está ociosa, visando manter o sincronismo, além de permitir o reconhecimento positivo ou negativo de MSU's recebidas e o controle de taxa de erros em um enlace. O reconhecimento pode se feito em blocos, ou por MSU. Entre os campos da FISU estão o *flag*, utilizado para a delimitação da mensagem, o *ck*, contendo informações para a detecção de erros, os números de sequencia *forward sequence number* (FSN) e o *backward sequence number* (BSN) e os bits indicadores *forward indicator bit* (FIB) e *backward indicator bit* (BIB), usados para a correção de erros. A Figura 2.1 mostra os campos da mensagem FISU.

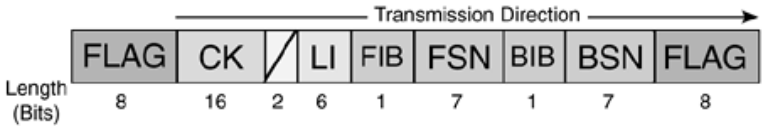


Figura 2.1: Campos da mensagem FISU

Fonte: (DRYBURGH; HEWETT, 2004)

A mensagem LSSU é enviada para indicar o estado de um link. Esta mensagem possui, além dos campos presentes na FISU, o campo *status field* (SF) para indicar o estado de um enlace. Os campos da mensagem LSSU são mostrados na Figura 2.2.

Figura 2.2: Campos da mensagem LSSU

Fonte: (DRYBURGH; HEWETT, 2004)

O último tipo de mensagem, a MSU, é usada para transportar as informações dos protocolos dos níveis 3 e 4. Esta mensagem possui além dos campos da mensagem FISU, os campos *service information octet* (SIO) e *service information field* (SIF). O SIF é o campo da mensagem que carrega informação e o SIO indica que tipo de informação está sendo carregada. Os campos da MSU são mostrados na Figura 2.3.

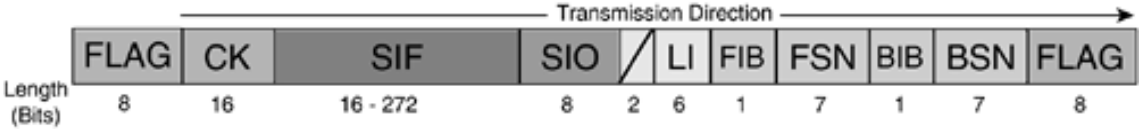


Figura 2.3: Campos da mensagem MSU

Fonte: (DRYBURGH; HEWETT, 2004)

2.1.2 Estrutura da rede SS7

Os nós da rede SS7 são chamados de pontos de sinalização (*signaling points* - SP's). Nas redes SS7 há três tipos de pontos de sinalização: o *service switching point* (SSP), que faz a interface entre switches de voz e a rede SS7, convertendo a sinalização que vem do switch de voz em sinalização SS7; o *signal transfer point* (STP), que atua como um roteador na rede SS7; e o *service control point* (SCP), que serve como interface para o acesso a bases de dados (RUSSEL, 2006).

Entre os nós de uma rede SS7, podem existir diferentes tipos de enlace, dos quais três serão considerados neste trabalho: links A, links B e links C. Os enlaces de acesso (*access links*) ou links A conectam um nó final da rede (um SSP ou um SCP) a um STP ou a um par de STP's. Os enlaces ponte (*bridge links*) ou links B conectam um STP ou um par de STP's a outro STP ou outro par de STP's desde que este, ou estes esteja (ou estejam) em um mesmo nível hierárquico. Os enlaces cruzados (*cross links*) ou links C, tem a função de formar pares de STP (DRYBURGH; HEWETT, 2004).

2.1.3 Protocolo MTP1

O MTP1 é o primeiro nível de protocolo do SS7, correspondente a camada física e tem como função a transmissão dos dados através da rede. Em uma rede E1 ou T1, a camada física é, geralmente, um *timeslot* em um quadro E1 ou T1. O MTP1 apenas especifica como as sequências de bits que representam as mensagens são transportadas de

um nó a outro na rede. Não há especificação a respeito da formação da mensagem ou do significado de cada bit dentro da mensagem (IBM, 2012).

2.1.4 Protocolo MTP2

O MTP2 é o segundo nível de do SS7, corresponde a camada de enlace, provendo detecção e correção de erros e a transmissão sequencial das mensagens SS7. A detecção de erros é feita através do método *cyclic redundancy check* (CRC) com 16 bits (CRC-16). O nó transmissor da mensagem faz o cálculo do CRC utilizando todos os campos da mensagem (exceto o campo *flag*) e gera uma sequência de verificação (*frame check sequence* - FCS). O nó receptor da mensagem faz um cálculo semelhante e compara o resultado com o campo FCS da mensagem recebida. Caso haja alguma inconsistência, a mensagem recebida é descartada e um reconhecimento negativo é enviado (RUSSEL, 2006).

Para a correção de erros, o método mais utilizado é o controle de erros básico (*basic error-control method*). Este método utiliza os bits indicadores da mensagem (FIB e BIB) para solicitar a retransmissão de mensagens. Quando a mensagem é transmitida normalmente, isto é, sem erros, o FIB e o BIB tem o mesmo valor. Quando uma retransmissão é solicitada, ou seja, uma mensagem foi recebida com erro, o BIB da mensagem enviada para pedir a retransmissão será invertido, enquanto seu FIB permanecerá com o mesmo valor.

Para o sequenciamento das mensagens são usados dois números de seqüência, o FSN e o BSN. O FSN indica o número da mensagem que está sendo enviada no momento. Este número é incrementado somente se a mensagem for uma MSU. Caso a mensagem seja uma FISU ou LSSU, ela assume o FSN da última MSU ou LSSU enviada com sucesso. O BSN é utilizado para o reconhecimento de mensagens. O BSN da mensagem que está sendo enviada no momento recebe o FSN da última mensagem que o ponto de sinalização que a envia recebeu com sucesso. Com isto, assume-se também que todas as mensagens anteriores à última recebida tenham sido recebidas com êxito (RUSSEL, 2006).

2.1.5 Protocolo MTP3

O MTP3 é o terceiro nível do protocolo SS7, correspondente a camada rede, e seu funcionamento se divide em duas funções: gerenciamento da rede de sinalização (*signa-*

ling network management - SNM) e tratamento das mensagens de sinalização (*signaling message handling* - SMH). O SNM consiste em uma série de procedimentos e mensagens cuja finalidade é tratar erros na rede para permitir que as mensagens continuem a chegar a seus destinos quando possível. O conjunto de função do SMH consiste de três funções básicas: discriminação, distribuição e roteamento (DRYBURGH; HEWETT, 2004).

A discriminação é a etapa onde se determina se as mensagens recebidas por um ponto de sinalização tem como destino aquele ponto de sinalização. Para isto, o ponto de sinalização analisa o rótulo de roteamento da MSU, que é composto por três campos: *origination point code* (OPC), *destination point code* (DPC) e *signaling-link code* (SLS), que indicam, respectivamente, o endereço (*Point Code* - PC) do ponto de sinalização que originou a mensagem, o endereço (PC) do ponto de sinalização ao qual ela se destina e o código do enlace (*linkset*) por onde a mensagem deve trafegar, isto é, a rota para a qual ela deve ser enviada. O ponto de sinalização que recebe a mensagem compara o DPC da mesma com seu próprio endereço. Se os endereços forem iguais, isto é, se a mensagem for destinada ao ponto de sinalização em questão, a mensagem é entregue para a função de distribuição. Caso contrário, a mensagem é entregue para a função de roteamento.

A função de distribuição é a função onde se determina que tipo de processamento a mensagem receberá, uma vez que o campo de informação de uma MSU, chamado de SIF, pode conter dados referentes a protocolos de nível 4, como o ISUP, ou informações referentes ao próprio MTP3. Para determinar que tipo de informação está contida no SIF, é analisado um campo da MSU chamado SIO. Este campo contém informações como o protocolo referente à informação contida no SIF, a versão deste protocolo, entre outras. Uma vez determinado qual protocolo deve processar a mensagem, este protocolo se encarrega de fazê-lo.

A função de roteamento é responsável por encaminhar a mensagem para o nó seguinte da rede, uma vez determinado que o DPC da MSU é diferente do endereço do ponto de sinalização em questão. Esta é uma função normalmente realizada pelos nós STP. É improvável que nós SSP ou SCP recebam mensagens que devem ser encaminhadas para outros nós. Primeiro, é determinado para qual ponto de sinalização a mensagem é destinada, analisando o seu DPC. Identificado o destino da mensagem, deve ser selecionada a rota para a qual a mensagem será encaminhada. As tabelas de roteamento dos pontos de sinalização apresentam mais de uma rota para cada destino. A cada rota é atribuída uma prioridade, de forma que a rota mais direta (com o menor número de saltos) é a que possui

a maior prioridade (RUSSEL, 2006).

2.1.6 Protocolo ISUP

O quarto nível consiste em uma série de diferentes protocolos chamados de *User Parts* e *Application Parts*. Como exemplo destes protocolos temos o ISUP, utilizado para o estabelecimento de chamadas, e o TCAP, utilizado para consulta a bases de dados através do SCCP. O protocolo TCAP é um protocolo baseado em transações (consultas e respostas). Sua principal função é prover informações adicionais para o estabelecimento de uma chamada sempre que necessário. As consultas podem ser feitas de um SSP para um SCP ou outro SSP. Para estas consultas, é utilizado o protocolo SCCP, que pode prover serviços orientados ou não orientados a conexão.

Nas redes SS7, o envio da sinalização para o estabelecimento e desligamento de chamadas é a principal função do ISUP. Para estabelecer uma chamada dentro da rede SS7, o SSP ligado ao assinante que inicia a chamada envia uma mensagem chamada de *initial address message* (IAM). Esta mensagem envia ao SSP destinatário informações como o número do assinante que inicia a chamada, o identificador do circuito que está sendo estabelecido (*circuit identification code* - CIC) e outras complementares. Se a IAM for enviada corretamente, o nó originário da chamada receberá em resposta a mensagem *address complete* (ACM), que indica que a chamada está sendo processada. Se a chamada for atendida, será enviada como indicação a mensagem *answer* (ANM). Após o envio da mensagem ANM, tem início o tráfego de voz. Durante este período não há tráfego de mensagens ISUP entre os nós envolvidos na chamada. Mensagens ISUP serão trocadas novamente apenas no desligamento da chamada. Quando um assinante desliga, o nó associado a este assinante envia a mensagem *release* (REL) para indicar o desligamento. Em condições normais, o nó que enviou a mensagem REL recebe em resposta a mensagem *release complete* (RLC) para confirmar o desligamento (RUSSEL, 2006).

2.2 Ferramentas para o desenvolvimento de um simulador de redes

Foram estudadas diferentes ferramentas que poderiam ser utilizadas para o desenvolvimento de um simulador de redes SS7. Cada uma destas ferramentas tem características

que a tornam mais apropriada para o uso em certas redes ou ambientes operacionais. Após o estudo dessas ferramentas decidiu-se pelo implementação do simulador em linguagem java para permitir uma maior portabilidade e também uma maior flexibilidade na programação.

2.2.1 Omnet++

O Omnet++ é um *framework* para o desenvolvimento de simulações discretas de redes na linguagem C++. É uma ferramenta de código aberto, com arquitetura modular e baseada em componentes. O Omnet++ provê componentes programados em C++ em uma interface de programação de aplicações(*application programming interface* - API) genérica para a implementação de redes de telecomunicações. Os recursos providos pela API do Omnet++ incluem nós, mensagens e funções para o envio de mensagens (OMNET++ COMMUNITY, 2012). O Omnet++ é muito utilizado para o desenvolvimento de simulação de redes de computadores, mas no caso de um simulador de redes SS7, se mostrou pouco vantajoso. Além disto, o desenvolvimento de uma interface gráfica personalizada, com todas as funcionalidades desejadas no SS7Sim, se mostrou extremamente complexo.

2.2.2 Desmo-J

O Desmo-J é um *framework* desenvolvido na Universidade de Hamburgo (Alemanha) para o desenvolvimento de simulações discretas na linguagem java. A API do Desmo-J provê a implementação de recursos como processos, filas, eventos, escalonamento, entre outros (SOURCEFORGE, 2012). Estes recursos poderiam ser úteis para o SS7Sim, mas representam, de fato, uma pequena vantagem. No entanto, o principal motivo pelo não uso deste *framework* foi o fato de não prover o recurso de montagem de modelo gráfico arrastando componentes para a tela (*drag-and-drop*), o que tornaria difícil a implementação da montagem do cenário pelo usuário.

2.2.3 Ptolemy

O Ptolemy é um *framework* de código aberto desenvolvido na Universidade da California em Berkeley. Este *framework* possibilita o desenvolvimento de modelos orien-

tados a atores. Atores são componentes de software executados de forma concorrente que comunicam-se entre sí enviando mensagem através de portas interconectadas. O Ptolemy disponibiliza uma API escrita em java e provê recursos para a o desenvolvimento de simulações discretas e contínuas (UC BERKELEY, 2012). No entanto, não apresenta muitos recursos específicos para o desenvolvimento de simulações de redes e nada foi encontrado em sua documentação a respeito da implementação de uma interface gráfica.

2.2.4 Desenvolvimento de um simulador em java

O projeto descrito neste documento teve início com o título de SS7Omnet, onde seria utilizado para o seu desenvolvimento o framework Omnet++, uma ferramenta amplamente utilizada para o desenvolvimento de simulações de redes de telecomunicações. No entanto, ao longo do desenvolvimento do projeto, foram encontradas dificuldades envolvendo o uso do Omnet++. O desenvolvimento de uma interface gráfica de usuário(*graphic user interface* - GUI) utilizando o Omnet++ se mostrou extremamente complexo, com um alto risco de não ser concluído dentro do prazo para o término do projeto. Além disto, conforme mencionado anteriormente, os recursos providos pelo Omnet++ se mostraram pouco vantajosos para o desenvolvimento do simulador proposto neste trabalho.

Como não foi encontrada uma ferramenta que oferecesse vantagens significativas para a implementação do simulador, optou-se por desenvolver o simulador na linguagem java. No simulador a ser implementado, a simulação teria início e fim, respectivamente, com o início e fim de uma chamada. O estado da simulação em andamento poderia ser alterado conforme o estado da chamada (atendida ou não atendida).

2.3 Ambientes para desenvolvimento

Para o desenvolvimento na linguagem java, existem diferentes Ambientes Integrados de Desenvolvimento (*Integrated Development Environments* - IDE) que podem ser utilizados, como Eclipse, JCreator e NetBeans. Neste caso, a opção foi pelo IDE NetBeans, por ser de fácil uso e por prover uma biblioteca gráfica que supria as necessidades do simulador: a NetBeans Visual Library.

2.3.1 NetBeans Visual Library

A NetBeans Visual Library é uma biblioteca para a programação de aplicativos gráficos escrita na linguagem de programação Java. A interface de programação de aplicações(*application programming interface* - API) é parte da plataforma NetBeans e é utilizada em diversos módulos e áreas. A programação com a NetBeans Visual Library é similar à programação com Swing, uma das principais bibliotecas gráficas utilizadas na linguagem Java. Os componentes gráficos são estruturados e gerenciados como uma árvore, onde todos os componentes gráficos descendem (isto é, são subclasses) de uma classe de componentes principal, a classe `Widget` (BöCK, 2009). A raíz desta árvore de componentes é uma classe de cenário, chamada `Scene`. Esta classe conterá todos os dados gráficos do cenário (NETBEANS COMMUNITY, 2011). A classe `Scene` pode se especializar em uma classe chamada de `GraphScene`. A `GraphScene` representa um modelo de cenário orientado a grafos, que contém nós (*nodes*) e vértices (*edges*). A conexão entre os nós está associada aos vértices (*edges*) da `GraphScene`. Cada vértice pode ser conectado a apenas um nó fonte e a apenas um nó destino (NETBEANS COMMUNITY, 2011). A cada vértice, pode ser associada uma classe específica de `Widgets` chamada `ConnectionWidget` (no simulador, esta classe se especializa na classe `Linkset`).

A preferência pela NetBeans Visual Library em relação a outras bibliotecas gráficas conhecidas da linguagem java (como swing, por exemplo) se deu por esta biblioteca possibilitar fácil implementação para a funcionalidade *drag-and-drop*, possibilitando que o usuário crie fácilmente o cenário, e também por prover recursos de animação 2D de fácil implementação, fundamentais para a simulação da troca de mensagens.

2.4 Modelagem de classes

Em um software orientado a objetos, um objeto representa um elemento do mundo real que é relevante para a solução do problema em questão. A representação de um grupo de objetos com as mesmas características é chamada de classe (BORATTI, 2007). Para a modelagem de classes, a linguagem de modelagem unificada (*unified modelling language* - UML) especifica o diagrama de classes, onde se modela as classes que compõe o software e o relacionamento entre as mesmas. Uma classe possui atributos, que são características comuns a todos os objetos da classe, e métodos, que são operações realizadas por estes objetos. Para cada classe, todos os seus atributos e métodos deverão constar no

diagrama de classes. No diagrama de classes, uma classe é representada por uma forma retangular com no máximo 3 divisões: uma para o nome da classe, uma para os atributos e outra para os métodos. A Figura 2.4 mostra um exemplo de diagrama de classe, no qual podem ser vistos todos os detalhes descritos a seguir.

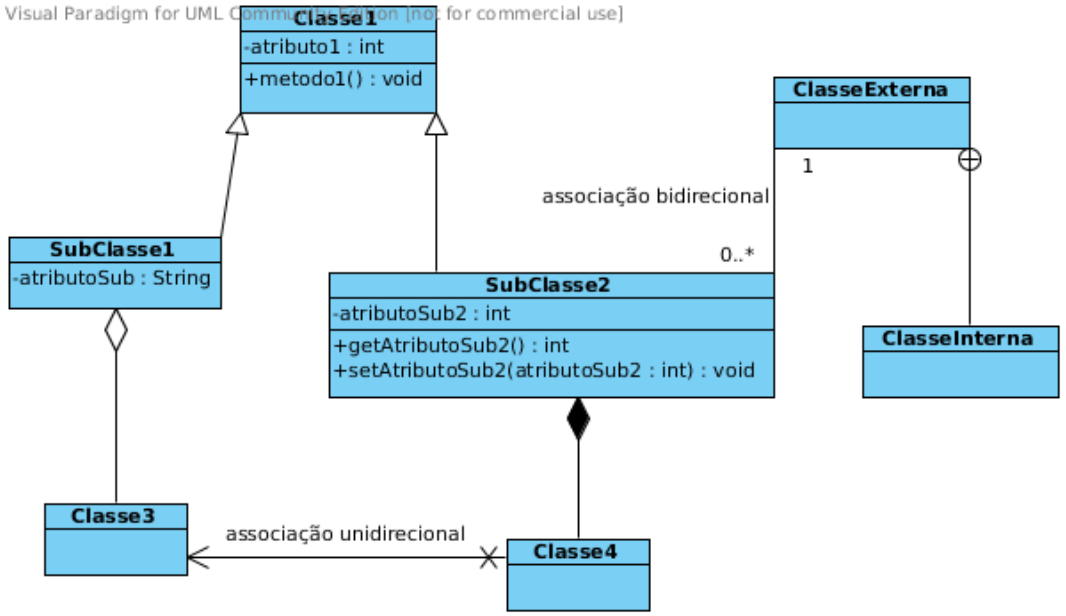


Figura 2.4: Exemplo de diagrama de classes em UML

A UML também prevê diferentes tipos de relacionamento entre classes, chamados de associações. Uma associação simples é representada por um segmento de reta ligando duas classes. Nas pontas destes segmentos, podem estar indicadas as multiplicidades do relacionamento, isto é, quantos objetos da primeira classe estão associados a quantos objetos da segunda classe. Na associação também pode haver outras indicações como, por exemplo, o papel de cada objeto neste relacionamento.

Existem casos especiais de associação. Um deles é a agregação, que é empregada em um relacionamento onde um objeto está contido em outro. Para representar graficamente uma agregação, utiliza-se um segmento de reta com um losango branco na ponta ligada à classe que representa o todo, isto é, a classe cujos objetos agregam outros objetos. Outro caso especial de associação é a composição, que é semelhante à agregação, porém, no caso da composição, o objeto que está contido não existe sem o objeto que o contém. A representação gráfica da composição também é semelhante à da agregação, mas ao invés do losango branco, há um losango preto na ponta do segmento de reta ligado ao objeto que representa o todo. As associações também podem ser unidirecionais ou bidirecionais. Em uma associação bidirecional, os objetos associados se conhecem mutuamente,

isto é, ambos referenciam o objeto associado em algum momento, neste caso, se diz que a associação é navegável em ambos os sentidos. Já em uma associação unidirecional, apenas um dos objetos referencia o objeto associado, neste caso, se diz que a associação é navegável em apenas um sentido. Quando uma associação é navegável em apenas um sentido, o segmento de reta que a representa é direcionado e aponta para a classe cujos objetos não tem conhecimento dos objetos da classe associada.

Outro conceito importante da programação orientada a objetos a ser tratado no diagrama de classes é o conceito de herança. Este conceito é empregado quando uma classe representa a especialização de outra classe, ou seja, contem todas as características desta outra classe e outras características próprias. A classe mais genérica neste relacionamento é chamada de superclasse e a mais especializada, de subclasse. Para representar graficamente este relacionamento, utiliza-se um segmento de reta com uma seta apontando para a superclasse.

Existe também a possibilidade de uma classe ser declarada dentro de outra classe. Uma classe declarada desta forma é chamada de classe interna. A notação gráfica usada para representar a associação entre classes internas e externas é um segmento de reta com um círculo, o qual contém dois segmentos de reta perpendiculares em seu interior, na extremidade ligada à classe externa. (BEZERRA, 2002).

2.5 Modelagem de casos de uso

A modelagem de casos de uso visa representar as funcionalidades externamente observáveis do sistema e os elementos externos ao sistema que interagem com o mesmo. Para esta modelagem, na UML, é utilizado o diagrama de casos de uso. Conforme mostrado na Figura 2.5, na modelagem de casos de uso estão presentes os casos de uso, os atores e os relacionamentos entre eles. Os casos de uso representam as funcionalidades do sistema mencionadas anteriormente e os atores representam os agentes externos que interagem com o sistema. Um caso de uso define o uso de parte da funcionalidade de um sistema, abstraindo a estrutura e o comportamento interno deste sistema. Na modelagem de casos de uso ainda estão previstos três tipos de relacionamento: relacionamento de comunicação, relacionamento de inclusão e relacionamento de extensão. O relacionamento de comunicação indica quais atores estão associados a quais casos de uso. O fato de um ator estar associado a um ou mais casos de uso significa que este ator troca

informações com o sistema. O relacionamento de inclusão é análogo a uma rotina em linguagens de programação. Este relacionamento denota uma sequëncia de interações que está inclusa no comportamento de um ou mais casos de uso, ou seja, será executada sempre que os casos de uso que a incluem forem executados. Esta sequëncia de interações pode ser vista como um outro caso de uso. Neste relacionamento, o comportamento contido no caso de uso incluso é obrigatóriamente executado quando o caso de uso que o inclui é executado. O relacionamento de extensão é usado quando um ou mais cenários de um determinado caso de uso podem incluir o comportamento especificado em outro caso de uso. Este outro caso de uso, portanto, estende o primeiro caso de uso. A inclusão deste comportamento, neste caso, é opcional (BEZERRA, 2002).

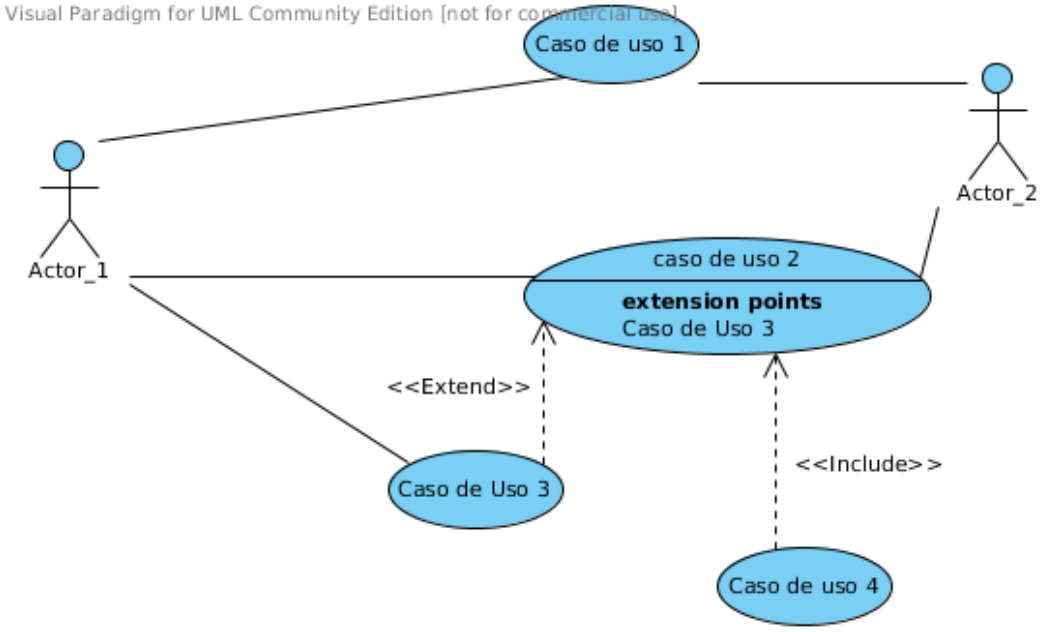


Figura 2.5: Exemplo de diagrama de casos de uso em UML

3 *Implementação do simulador de sistema de sinalização SS7 - SS7Sim*

Para a implementação do protótipo do simulador, foi necessario escolher um subconjunto das funções e protocolos do SS7. Portanto, foram identificadas dentro das funções do SS7, em cada nível de protocolo, aquelas que seriam prioritárias para que fosse possível uma simulação básica. Foi definindo, então, que o SS7Sim deveria ser capaz de simular o procedimento de uma chamada básica. Para uma simulação deste tipo, seria necessária a implementação das funções de estabelecimento e desligamento de chamadas do protocolo ISUP. As informações do protocolo ISUP são carregadas por mensagens do tipo MSU. Para que essas MSU's possam trafegar pela rede, é necessária a implementação das funções de SMH do nível MTP3 (discriminação, distribuição e roteamento). Como as informações referentes às funções do MTP3 e do ISUP anteriormente referidas estão presentes em poucos campos da mensagem, seria importante introduzir na simulação informações referentes ao nível MTP2.

3.1 Escolha dos eventos SS7 a serem simulados

Uma das estapas do desenvolvimento deste projeto consistia na identificação dos principais eventos que ocorrem dentro de uma rede SS7 e que seriam relevantes para a simulação. Conforme definido anteriormente, o SS7Sim deveria ser capaz de simular uma chamada básica. Esta chamada. Esta chamada consiste em estabelecimento, possível atendimento e desligamento. Conforme também mencionado anteriormente, o estabelecimento e desligamento de uma chamada é função do protocolo ISUP, portanto é fundamental que o SS7Sim implemente esta função. Dado que a simulação no SS7Sim consiste em uma chamada básica, o SS7Sim implementa a troca de mensagens ISUP para

o estabelecimento (IAM E ACM), atendimento (ANM) e desligamento (REL e RLC).

Para que seja possível simular o tráfego de mensagens na rede, cada nó no SS7Sim deve se capaz de determinar se uma mensagem é destinada a ele mesmo, processar a mensagem case ela o seja e, se o nó for um STP, encaminhar a mensagem caso esta seja destinada a outro nó da rede. Nas redes SS7 este papel é do SMH (nível de protocolo MTP3). O SS7Sim, portanto, implementa as funções de discriminação, distribuição e roteamento.

Nos campos das mensagens da rede simulada, estão presentes uma série de informações referentes ao nível de protocolo MTP2, no entanto, a maioria dos valores destes campos é definida estaticamente. Porém, as mensagens trocadas no SS7Sim são mensagens do tipo MSU, e mensagens deste tipo têm seus números de sequência (FSN e BSN) incrementados á medida que são trocadas na rede. Por este motivo, o SS7Sim implementa a função de sequenciamento de mensagens do nível MTP2.

Como o nível MTP1 especificica apenas como ocorre o transporte de mensagens de um nó a outro, tratando-as puramente como sequências de bits, o SS7Sim não implementa nada referente a este nível de protocolo.

3.2 Implementação do simulador em java

Uma vez identificados os eventos do SS7 a serem implementados no SS7Sim, teve início o processo de implementação do simulador. Este processo foi dividido nas etapas de modelagem, onde foram produzidos os diagramas UML de classes e casos de uso, e codificação, onde os eventos identificados foram traduzidos para a linguagem java. A presente seção descreve o processo de implementação do simulador.

A Figura 3.1 mostra o diagrama de casos de uso do SS7Sim. Observa-se neste diagrama a existência de apenas um ator, o `Ator Usuário`. As elipses representam os casos de uso, com seu nome no centro. As linhas contínuas representam os relacionamentos de comunicação e as linhas tracejadas com o rótulo *extend* representam relacionamentos de extensão. No caso, os casos de uso `Estabelecimento de chamada`, `Indução de Defeito em Enlace`, `Desligamento de Chamada` e `Visualização de Mensagem` estendem o caso de uso `Simulação`. O detalhamento dos casos de uso do SS7Sim é mostrado no Apêndice A.

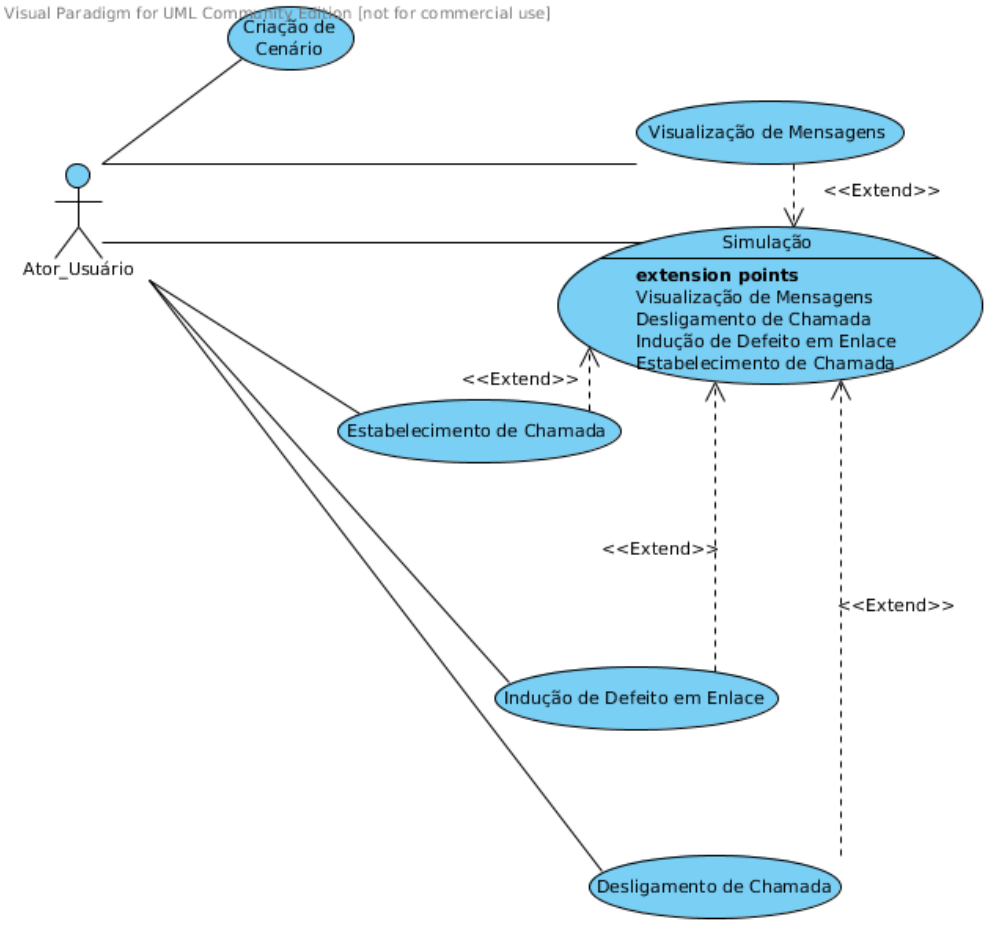


Figura 3.1: Diagrama de casos de uso do SS7Sim

Para o SS7Sim foram produzidos três diagramas de classes. O primeiro, mostrado na Figura C.1 do Apêndice C contém as classes que compõe a estrutura principal do simulador, o segundo, mostrado na Figura C.2 do Apêndice C contém classes internas à classe SimScene, ou seja, classes declaradas dentro da classe SimScene, que implementam recursos gráficos utilizados no simulador, como menus,conexão entre nós, etc. O terceiro diagrama, mostrado na Figura C.3 do Apêndice C contém as classes que representam as janelas exibidas ao usuário ao inspecionar um elemento da rede no simulador.

3.2.1 Funções implementadas do protocolo MTP2

Dentre as funções do protocolo MTP2 foi implementado apenas o sequenciamento de mensagens. Esta função pôde ser implementada de forma bastante simples. Na classe que representa as MSU's (a classe `MsuWidget`), há atributos do tipo inteiro (`int`), isto é, números inteiros, para representar os números de seqüência da mensagem (FSN e BSN). Há também atributos do tipo booleano (`boolean`) para representar os bits indicadores FIB e BIB. Os valores destes atributos são modificados conforme a progressão da troca de mensagens na simulação.

3.2.2 Funções implementadas do protocolo MTP3

Dentre as funções do protocolo MTP3, foram implementas apenas as funções do SHM: discriminação, distribuição e roteamento. A discriminação determina para qual das outras funções do SMH (distribuição ou roteamento) a mensagem recebida deve ser enviada. Um teste simples pode cumprir este papel. Como os objetos que representam os nós SS7 tem como atributos os seus PC's e as MSU's terão como atributos todos os campos referentes a seus rótulos de roteamento (OPC, DPC e SLS). Logo, a função de discriminação é implementada através de um teste que verifica se o atributo PC do nó SS7 que recebe a mensagem corresponde ao DPC da mensagem recebida. De acordo com o resultado deste teste, o simulador executará procedimentos referentes à função do SMH aplicável a esta mensagem. Para realizar esta função, as classes que representam os nós SS7 no SS7Sim possuem um método chamado de `processMessage`. Este método é chamado sempre que o nó tem uma mensagem a processar. A implementação deste método nos nós SSP (`SspWidget`) e STP (`StpWidget`) é mostrada nas seções B.1.1 e B.1.2 do Apêndice B.

A função de distribuição também foi implementada de forma relativamente simples. Como o SS7Sim está apenas simulando a troca de mensagens ISUP, esta função foi simplificada. Na implementação do método `processMessage` da classe `SspWidget`, pode-se observar que, uma vez determinado que a mensagem é destinada ao nó que a recebeu, este nó verifica diretamente que tipo de mensagem ISUP esta MSU carrega e a processa de acordo com esta informação. Uma vez que já se sabe que cada mensagem recebida carrega informações do protocolo ISUP, é possível, após o recebimento da mensagem, visualizar as informações referentes a este protocolo via interface gráfica, como mostra a Figura 3.2. O processamento de mensagem dará sequëncia à simulação, determinando como o nó que recebeu a mensagem processada deve proceder a partir desta mensagem.

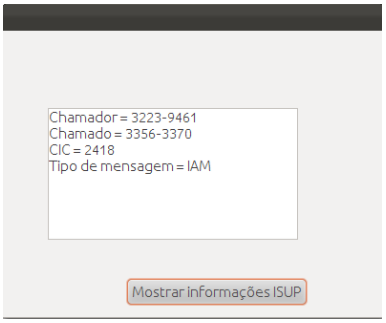


Figura 3.2: Visualização das informações do protocolo ISUP no simulador SS7Sim

A função de roteamento teve uma implementação bastante complexa, uma vez que esta envolve procedimentos prévios como a criação e manutenção de tabelas de roteamento e o estabelecimento de rotas, bem como o cálculo de prioridades para as mesmas. Para que a implementação desta função fosse possível e próxima da situação real, foi necessário que os objetos que representam os nós STP (`StpWidgets`) mantivessem tabelas de roteamento simuladas. Para representar estas tabelas de roteamento, foi criada uma classe java chamada de `RoutingTable`, que contém uma lista de entradas. As entradas da tabela de roteamento são representadas por uma classe interna à `RoutingTable` chamada `Entry`. Cada entrada da tabela de roteamento possui o identificador de um determinado nó da rede (campo `label`), seu PC (campo `dpc`, pois este nó é um possível destino para as mensagens encaminhadas), o identificador do linkset pelo qual a mensagem deve ser enviada (campo `lsid`) e a prioridade daquela rota (campo `routePriority`). Este modelo de tabela de roteamento foi implementado com base no modelo apresentado na Figura 3.3.

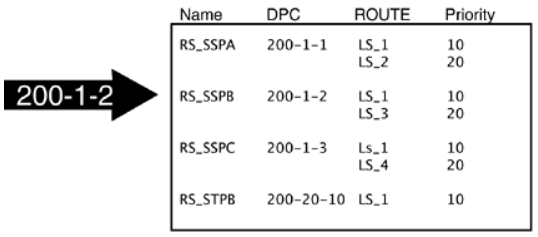


Figura 3.3: Exemplo de uma tabela de roteamento

Fonte: (DRYBURGH; HEWETT, 2004)

O código da classe `RoutingTable` e de sua classe interna `Entry` são apresentados na seção B.1.3 do Apêndice B. As operações realizadas pelos objetos da classe `RoutingTable` são, basicamente, a adição e remoção de entradas na tabela e a obtenção das entradas contidas na mesma. Os objetos `Entry` apenas fornecem os valores de seus atributos e permitem a alteração do valor da prioridade de uma rota (`routePriority`).

Uma vez criadas as tabelas de roteamento, é necessário que estas tabelas sejam atualizadas à medida que são inseridos nós na rede. No SS7Sim, cada `StpWidget` é capaz de atualizar sua própria tabela de roteamento. O processo de atualização das tabelas de roteamento ocorre, em cada nó STP da rede simulada, sempre que o usuário adiciona um novo `linkset` na rede, remove um nó da rede ou induz defeito em algum `linkset`. O processo de atualização das tabelas de roteamento, no caso da inserção de um `linkset`

entre dois nós, é baseado nos nós que foram conectados. Cada vez que dois nós são conectados, dá-se início a um procedimento para a atualização das tabelas de roteamento de todos os `StpWidgets` na rede. Os nós conectados pelo novo `linkset` são tratados como novos destinos por cada `StpWidget` da rede e o novo *linkset* é tratado como uma nova possível rota por qualquer um dos nós conectados que seja um `StpWidget`. Quando um `StpWidget` é conectado a outro `StpWidget`, cada um destes nós adiciona todas as entradas da tabela de roteamento do outro na sua própria tabela de roteamento, definindo como rota o novo `linkset` criado. Para realizar esta função, existe um método na classe `StpWidget` chamado de `updateRoutingTable`. Este método tem duas implementações, uma delas recebe como parâmetro um objeto `Ss7Widget` e a outra, um objeto `Linkset`, sendo que a primeira implementação trata os casos de inserção e remoção de um nó na rede e a segunda, de atualização de prioridades e remoção de um `linkset`. Além do objeto `Ss7Widget` ou do objeto `Linkset`, cada uma destas implementações recebe como parâmetro a ação a ser realizada (inserção, remoção ou atualização de prioridades). Dada esta ação, o método `updateRoutingTable` chamará métodos específicos para realizá-la. Para as ações de inserção e remoção de nós SS7 na rede, estes métodos são, respectivamente, `addToRoutingTable` e `removeFromRoutingTable`. Para as ações de atualização de prioridades de rotas e remoção de um `linkset`, são chamados, respectivamente, o método `updateRoutePriorities` e uma outra implementação do método `removeFromRoutingTable`, esta recebendo um objeto `Linkset` como parâmetro. O código destes quatro métodos mencionados é mostrado a seguir é mostrado, respectivamente, nas seções B.1.4, B.1.5, B.1.6 e B.1.7 do Apêndice B.

Sendo os nós STP da rede simulada capazes de criar e atualizar suas tabelas de roteamento, eles são, portanto, capazes de encaminhar as mensagens recebidas durante a simulação. No SS7Sim, cada nó monitora periodicamente a chegada de mensagens, mantendo as mensagens recebidas em uma fila (`messageQueue`). Quando o nó detecta uma mensagem na fila aguardando processamento, o nó se encarrega de processá-la, chamando o método `processMessage`. No caso dos `StpWidgets`, este método determina se a mensagem recebida deve seguir para a função de distribuição ou de roteamento. Como se trata de uma MSU carregando informações do protocolo ISUP, a mensagem acaba sempre seguindo para a função de roteamento. Para encaminhar a mensagem, o STP deve selecionar uma nova rota (isto é, um novo `linkset`) pela qual ela deve ser enviada para o próximo nó. Nos `StpWidgets`, este processo é realizado dentro do método `forwardMsu`. O método `forwardMsu` é chamado pelo método `forwardMessage`, que verifica o tipo

de mensagem SS7 recebida (no caso, uma MSU) e chama o método específico para encaminhar aquela mensagem. O método `forwardMessage`, por sua vez, é chamado pelo método `processMessage`. O código do método `forwardMsu` é mostrado a na seção B.2.1 do Apêndice B.

3.2.3 Funções implementadas do protocolo ISUP

O SS7Sim implementa as principais funções do protocolo ISUP: estabelecimento e desligamento de chamadas. No SS7Sim, a simulação consiste na troca de mensagens SS7 em uma chamada básica, onde um assinante inicia uma chamada, que pode ou não ser atendida, e um dos assinantes desliga. As mensagens ISUP envolvidas em uma chamada básica são: IAM e ACM (início da chamada), ANM (atendimento) e REL e RLC (encerramento da chamada). O usuário inicia a simulação iniciando uma chamada entre dois SSP's, o que faz com que o SSP que inicia a chamada envie uma mensagem IAM ao SSP destinatário. O destinatário, por sua vez, responde com uma mensagem ACM (o simulador assume que a IAM foi recebida corretamente). Caso a chamada seja atendida, o SSP destinatário envia a mensagem ANM ao originário da chamada. Por fim, quando o usuário decidir encerrar a chamada, o SSP associado a esta ação envia a mensagem REL, que é respondida com a mensagem RLC.

O envio das mensagens IAM, ANM e REL é oriundo de ações do usuário, já as mensagens enviadas em resposta (ACM e REL), resultam do processamento das mensagens ISUP recebidas nos nós SSP. A classe `SspWidget` possui dois métodos responsáveis pelo envio das mensagens ACM e REL, os métodos `processIam` (envia a ACM) e `processRel` (envia a RLC). O algoritmo destes métodos consiste basicamente em criar as mensagens (isto é, instanciar as `MsuWidgets`), alterar campos da mensagem que assim necessitem e e enviá-las. Estes métodos são chamados dentro do método `processMessage`. Para a criação das demais mensagens, a classe `SspWidget` possui o método `createMessage`. O código destes três métodos é mostrado nas seções B.2.2, B.2.3 e B.2.4 do Apêndice B.

3.3 A interface gráfica de usuário

O SS7Sim foi desenvolvido como um módulo do IDE NetBeans e utiliza, portanto, o mesmo modelo de interface gráfica deste IDE. A GUI do SS7Sim é muito semelhante à do NetBeans, as diferenças são o espaço para a montagem do cenário, situado na parte

central da janela, e uma paleta contendo os nós SS7, localizada à direita da janela. A tela inicial do SS7Sim é mostrada na Figura 3.4.

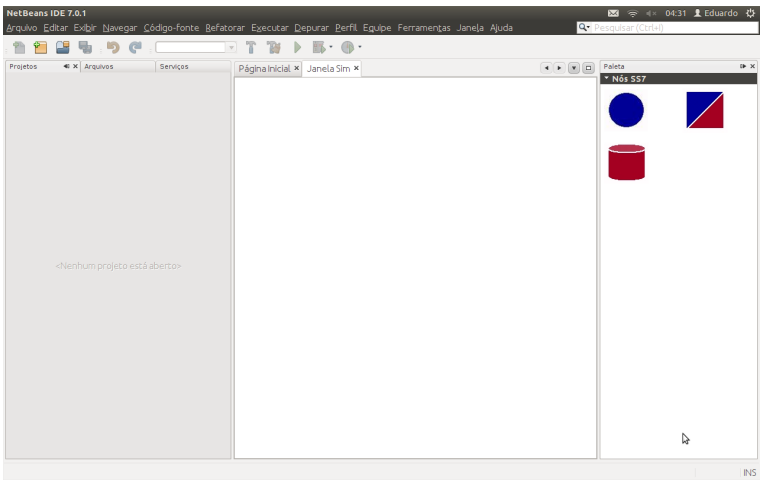


Figura 3.4: Tela inicial do SS7Sim

3.3.1 Criação do cenário pelo usuário

Para criar uma rede no SS7Sim, o usuário arrasta os nós SS7 para o centro da janela. Esta parte central da tela é a classe de cenário (`Scene`) da NetBeans Visual Library. No caso do SS7Sim foi utilizada uma especialização da `GraphScene`: a `SimScene` .

Quando o usuário arrasta os nós SS7 para o cenário, são criados `Widgets` para representar cada um deles. No SS7Sim, este é o papel cumprido pela classe `Ss7Widget`, que se especializa em classes específicas para cada um dos nós SS7 (`SspWidget`, `StpWidget` e `ScpWidget`). Estas classes, portanto, além de implementar suas funções específicas dentro da rede SS7, implementam também um componente gráfico da NetBeans Visual Library. A Figura 3.5 mostra a aparência destes nós no SS7Sim.

(a) SSP (b) STP (c) SCP

Figura 3.5: Representação gráfica dos nós SS7 no simulador no simulador SS7Sim

Outra etapa da criação do cenário no é a conexão entre os nós. A princípio, é permitido que os nós sejam conectados livremente na `GraphScene`, ficando a cargo do desenvolvedor implementar quaisquer restrições. No SS7Sim, foi necessária a imposição de

restrições, afim de aproximar a rede simulada de uma rede real, respeitando as especificidades de cada nó e de cada tipo de `linkset` que pode ser criado (links A, B e C). Os STP's, são adicionados ao pares, isto é, a cada vez que o usuário insere um nó STP no cenário, um outro STP, que formará um par com o nó inserido, é adicionado automaticamente. Todo o nó que for conectado a um STP deve estar conectado também ao par deste STP, sendo que qualquer nó que não seja um STP pode estar conectado a apenas um par de STP's. Além disto, não são permitidas conexões diretas entre nós SSP e SCP, uma vez que este tipo de conexão caracteriza um tipo de enlace que não está sendo tratado dentro do simulador (o *fully associated link* ou link F). Um cenário corretamente montado teria aparência semelhante à do cenário mostrado na Figura 3.6.

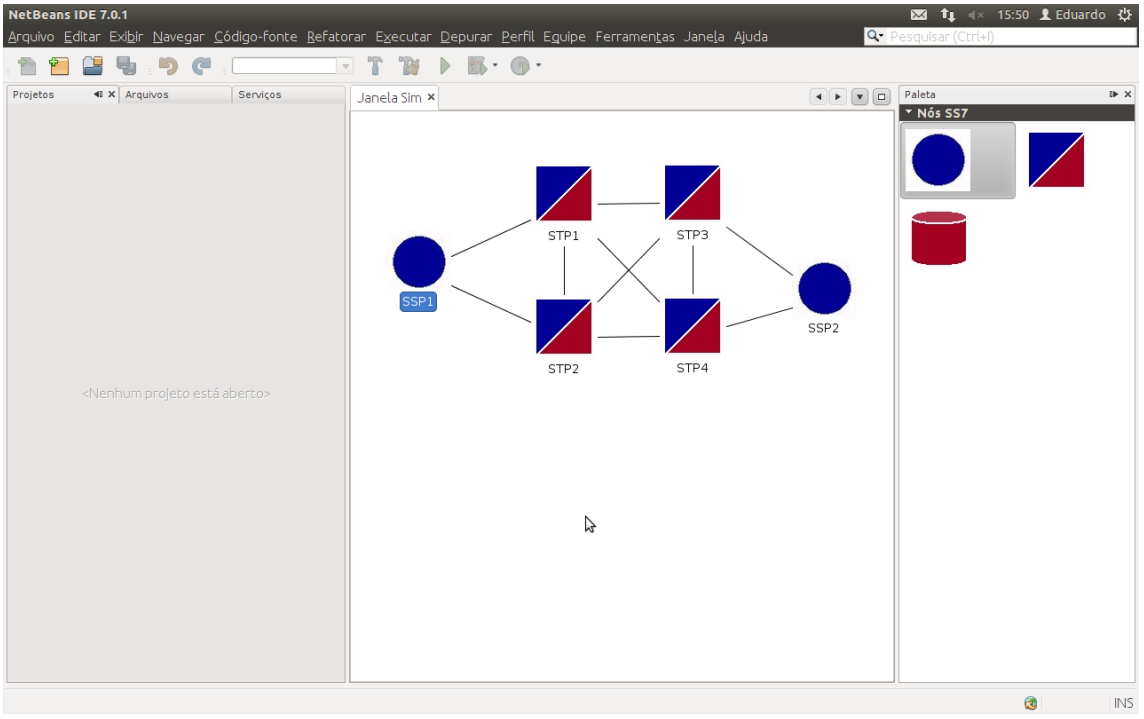


Figura 3.6: Um cenário do SS7Sim

3.3.2 Interação do usuário com o simulador

A interação do usuário com o SS7Sim é feita por meio de menus que aparecem na forma de janelas *popup*. Estes menus aparecem quando o usuário com o botão direito do *mouse* em um dos componentes do cenário. Cada elemento do cenário tem um menu específico com um conjunto de ações que podem ser realizadas por aquele elemento. Todos os componentes podem ser inspecionados para a visualização de informações relevantes. Alguns componentes podem realizar ações específicas, por exemplo, um SSP pode iniciar

uma chamada, dando início à simulação, e pode também atender ou encerrar a chamada. As ações disponíveis neste nó dependem do estado da chamada (se a chamada foi atendida ou não). Nos *linksets* é possível induzir defeito no link, fazendo com que mensagens não trafeguem mais pelo *linkset* em questão.

4 Resultados obtidos

A simulação implementada no SS7Sim consiste na troca de mensagens ISUP em uma chamada básica. Este processo tem início quando o usuário seleciona a opção `Iniciar chamada` no menu de um SSP e seleciona o número que será chamado (e, portanto, o SSP destinatário da chamada). Após o início da chamada, o menu do SSP destinatário apresentará a opção `atender` e ambos os SSP's envolvidos na chamada apresentarão em seus menus a opção `desligar`.

Para fim de testes, foram montados alguns cenários, nos quais foram iniciadas algumas chamadas e foi induzido defeito em enlaces. Cada um deste cenário é mostrado em uma figura que ilustra todo o processo desde o início da chamada ou indução de defeito em enlace pelo usuário até o envio das mensagens.

4.1 Interação do usuário com o sistema via interface gráfica

A implementação de uma interface gráfica amigável para o usuário foi um dos principais objetivos deste projeto, uma vez que o SS7Sim deveria ser de fácil uso e interativo. Com o auxílio dos recursos providos pela NetBeans Visual Library, foi possível a implementação de uma série de funcionalidades que facilitam a interação do usuário com o simulador.

Para criar um cenário, o usuário adiciona os nós desejados na rede e estabelece as conexões (*linksets*) entre eles. A adição de nós na rede é feita através da funcionalidade *drag-and-drop*, onde o usuário clica em dos nós que aparece na paleta de componentes e o arrasta para a parte central da tela. Nesta parte da criação do cenário, existe uma peculiaridade na adição de nós STP. Cada vez que o usuário arrasta um nó STP para o cenário, outro nó STP é adicionado automaticamente ao cenário para formar um par com

o primeiro. A Figura 4.1 mostra a adição de nós ao cenário através da funcionalidade *drag-and-drop*.

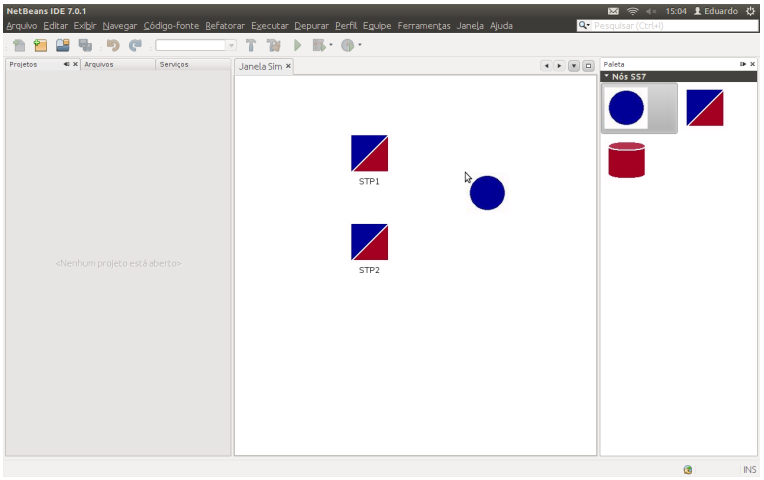


Figura 4.1: Adicionando nós SS7 ao cenário no SS7Sim

A criação de enlaces é feita de forma semelhante. Para conectar dois nós, o usuário pressiona a tecla `Ctrl`, clica duas vezes no nó origem da conexão e arrasta o *mouse* até o nó destino. Se esta conexão for permitida conforme as especificações do simulador, ela será estabelecida. A Figura 4.2 mostra a criação de um enlace no SS7Sim.

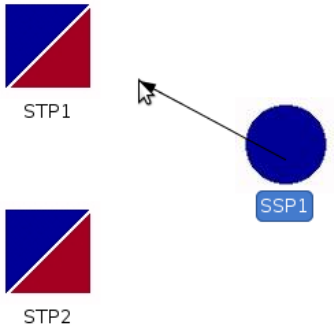


Figura 4.2: Criando um enlace no SS7Sim

Uma vez criado o cenário, o usuário interage com o sistema através menus, que surgem na forma de janelas *popup* cada vez que o usuário clica com o botão direito do *mouse* sobre um componente do cenário ou sobre o próprio cenário. Cada um destes menus apresenta ações que podem ser realizadas no componente ao qual ele está associado. Na Figura 4.3 são mostrados alguns exemplos destes menus. Na Figura 4.3(a) é mostrado o menu inicial de um nó SSP, com as opções `iniciar chamada` e `inspecionar`. A opção `inspecionar` é comum a todos os componentes do cenário, enquanto a opção `iniciar chamada` é específica do nó SSP antes do início da simulação. Durante a simulação, o menu de um nó SSP pode apresentar as opções `atender` e `desligar`. Na Figura 4.3(b) é mostrado o menu de um *linkset*. A opção `induzir defeito` presente neste menu fará

com que aquele *linkset* se torne defeituoso. Na Figura 4.3(c), é mostrado o menu do cenário. O usuário tem as opções de salvar o cenário, carregar um cenário de um arquivo ou criar um novo cenário desde o início. É possível também que o usuário remova nós do cenário selecionando o nó e pressionando o botão `Delete`, no entanto, a remoção de nós desta maneira pode interferir em simulações futuras, causando um funcionamento inadequado do simulador. Remover um nó da rede fará com que todos os *linksets* conectados a ele sejam removidos e atualizará as tabelas de roteamento de todos os nós STP.

(a) Menu de um SSP (b) Menu de um *linkset* (c) Menu do cenário

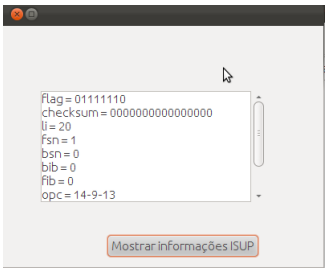
Figura 4.3: Menus do SS7Sim

4.2 Inspeção dos campos do protocolo MTP2

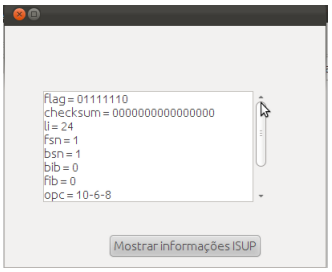
Todos os componentes da rede simulada, inclusive mensagens, podem ser inspecionados, mostrando ao usuário informações relevantes Ao selecionar a opção `inspecionar` no menu da mensagem, uma janela mostrará ao usuário os campos daquela mensagem referentes aos protocolos MTP2 e MTP3. Cada linha da área de texto da janela de inspeção corresponde a um campo da mensagem e seu valor. As linhas de 1 a 7 mostram os campos: *flag, checksum*, LI, FSN, BSN, BIB e FIB. As linhas 8, 9 e 10 mostram, respectivamente, os campos OPC, DPC e SLS. Os sete primeiros campos mostrados são campos relativos ao protocolo MTP2, no entanto, apenas os números de sequência (FSN e BSN) são relevantes no SS7Sim, uma vez que, entre as funções do protocolo MTP2, apenas a função de sequenciamento de mensagens foi implementada. O campo *checksum* receberá sempre o mesmo valor, pois sem a implementação da função de detecção de erros, não há cálculo de CRC. O valor dos bits indicadores BIB e FIB permanecerá inalterado devido ao fato de não ter sido implementada a função de correção de erros. O valor do campo LI indicará sempre que a mensagem é do tipo MSU, uma vez que mensagens FISU e LSSU não são trocadas no SS7Sim.

Inspecionando mensagens trocadas ao longo de uma chamada, pode se observar o incremento dos números de sequência conforme a progressão das mensagens. Como o SS7Sim não trabalha com erros na transmissão de uma mensagem, o valor do FSN da

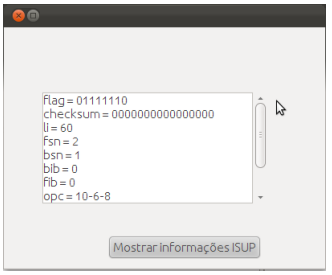
ultima mensagem recebida por um nó será sempre o valor do BSN da próxima mensagem enviada por aquele nó. A Figura 4.4 mostra as janelas de inspeção de mensagens IAM, ACM e ANM enviadas durante uma mesma chamada realizada no Cenário de testes 1.



(a) Inspeção de uma mensagem IAM - campos do protocolo MTP2



(b) Inspeção de uma mensagem ACM - campos do protocolo MTP2

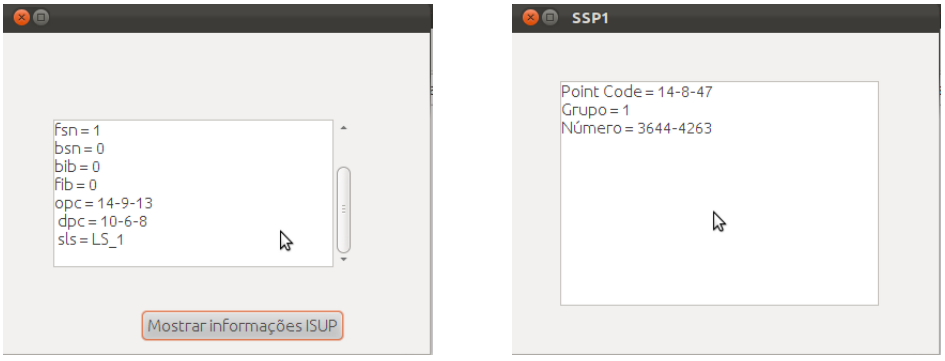


(c) Inspeção de uma mensagem ANM - campos do protocolo MTP2

Figura 4.4: Inspeção dos campos do protocolo MTP2 no SS7Sim

4.3 Inspeção dos campos do protocolo MTP3

As funções do protocolo MTP3 implementadas no SS7Sim também podem ser observadas por meio de inspeção. Conforme mencionado anteriormente, as funções do protocolo MTP3 implementadas no SS7Sim correspondem às funções do SMH (discriminação, distribuição e roteamento). Os campos da mensagem relevantes para estas funções são os campos referentes ao rótulo de roteamento (OPC, DPC e SLS). A função de discriminação foi implementada através da comparação entre o DPC da mensagem recebida e o PC do nó que a recebeu. Esta função pode ser visualizada no simulador através da inspeção de uma mensagem e do nó que a recebe, como mostra a Figura 4.5. Nesta figura, são mostradas as janelas de inspeção de uma mensagem IAM e de um nó SSP que a recebe. Conforme mencionado na seção anterior, as três últimas linhas da janela de inspeção de uma mensagem correspondem aos campos do rótulo de roteamento. No SS7Sim, os PC's dos nós são gerados aleatoriamente, seguindo o formato `4 bits - 4 bits - 6 bits`. O SLS mostra por qual *linkset* o nó que originou a mensagem a enviou, no entanto, não é selecionado um *link* dentro deste *linkset* para o envio da mensagem. Na janela de inspeção do nó SSP, a primeira linha mostra seu PC, a segunda linha mostra o seu grupo e a terceira, o número de telefone associado a este SSP. No SS7Sim, um grupo corresponde a um par de STP's e os SSP's ou SCP's conectados a ele. No SS7Sim, os nós são agrupados desta forma para restringir a criação de *linksets*, uma vez que nem todos os tipos de enlaces do SS7 podem ser criados no simulador. Cada nó SSP possui também um número de telefone associado.



(a) Inspeção de uma mensagem IAM - campos do protocolo MTP3

(b) Inspeção de um nó SSP

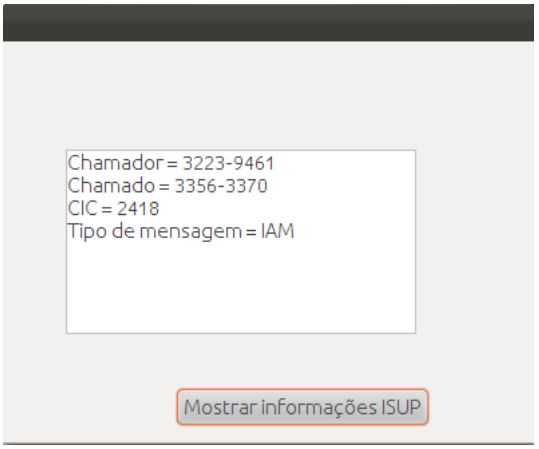
Figura 4.5: Inspeção dos campos do protocolo MTP2 no SS7Sim

Caso a mensagem seja destinada ao nó que a recebeu, como ocorre no exemplo mostrado na Figura 4.5, a mensagem será entregue à função de distribuição. Como no SS7Sim apenas as mensagens ISUP envolvidas em uma chamada básica são trocadas, esta função trata as mensagens recebidas diretamente como mensagens ISUP e estas mensagens são imediatamente processadas. Caso a mensagem seja destinada a outro nó da rede, ela será entregue à função de roteamento. No SS7Sim, apenas os nós STP realizam esta função e, como as mensagem trocadas no SS7Sim são mensagens ISUP, elas serão sempre encaminhadas quando recebidas por um nó STP. Os nós STP encaminham mensagens com base em suas tabelas de roteamento, que são contruídas à medida que são estabelecidos os *linksets* na rede. Quando se inspeciona um nó STP, é possível ver sua tabela de roteamento através do botão `Mostrar tabela de roteamento` que aparece em sua janela de inspeção. A Figura 4.6 mostra a inspeção de uma tabela de roteamento em um nó STP (dados apresentados no formato: `DPC` (PC do nó destino) - `label` (identificador do nó destino) - `rota` (identificador do *linkset* de saída para esta rota)). As prioridades para rotas não são mostradas na inspeção pois estas são estabelecidas apenas antes do início de uma simulação, quando se assume que o cenário não sofrerá mais alterações. As prioridades devem ser estabelecidas desta forma pois são calculadas com base no número de saltos entre o STP em questão e um determinado destino na rede. No SS7Sim as prioridades para rotas são números inteiros entre 0 (inclusive) e 100 (exclusive) contados em intervalos de 10. Uma prioridade para rota igual a 0 indica enlace defeituoso.

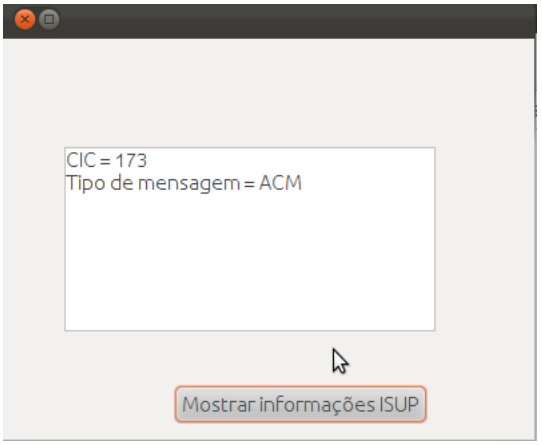
Figura 4.6: Inspeção de uma tabela de roteamento

4.4 Inspeção dos campos do protocolo ISUP

Ao inspecionar as mensagens na simulação, também é possível visualizar informações do protocolo ISUP. Cada janela de inspeção de mensagem possui o botão `Mostrar informações ISUP`, que dá ao usuário a opção de visualizar em cada mensagem informações específicas deste protocolo. A visualização de informações ISUP é mostrada na Figura 4.7, onde é mostrada a inspeção de uma mensagem IAM e de uma mensagem ACM. Os campos ISUP mostrados são: CIC, tipo de mensagem ISUP e no caso de uma mensagem IAM, são mostrados os números dos assinantes chamador e chamado.



(a) Inspeção de uma mensagem IAM - campos do protocolo ISUP



(b) Inspeção de uma mensagem ACM - campos do protocolo ISUP

Figura 4.7: Inspeção dos campos do protocolo ISUP no SS7Sim

4.5 Inspeção de *linksets*

No SS7Sim, são criados *linksets* para a conexão entre os nós. No entanto, conforme mencionado anteriormente, não são específicados *links* dentro de cada *linkset*, portanto,

qualquer ação realizada em um enlace no SS7Sim envolve todo um *linkset*. Uma mensagem enviada por um determinado nó pode trafegar por qualquer *linkset* operacional na rede e, como não foram implementadas regras para a seleção de *links*, uma mensagem enviada em resposta para este mesmo nó pode trafegar pelo mesmo caminho. Além disto, induzir defeito em um enlace significa tirar de operação todo um *linkset*, fazendo com que nenhuma mensagem trafegue pelo mesmo. Como não foi implementada a função de correção de erros, o *linkset* defeituoso permanecerá defeituoso, não havendo como recuperá-lo. A Figura 4.8 mostra a inspeção de dois *linksets*, um normal e um defeituoso. Os campos mostrados na janela de inspeção do *linkset* são: o identificador do *linkset* (`LSID`), o tipo de enlace (A, B ou C) e o *status* do *linkset*, que pode ser normal (*linkset* operacional) ou *out of service* (OOS - *linkset* defeituoso).

(a) Inspeção de um *linkset* operacional

(b) Inspeção de um *linkset* defeituoso

Figura 4.8: Inspeção de *linksets* no SS7Sim

4.6 Cenário 1 - Dois pares de STP's e dois SSP's

O primeiro cenário montado é o mesmo cenário mostrado na Figura 3.6. Um cenário simples com apenas dois SSP's e dois pares de STP's. Cada SSP está conectado a um par de STP's. Apesar de simples, este cenário possibilita que se observem mensagens percorrendo a rede de uma extremidade a outra, uma vez que cada nó SSP está em uma extremidade da rede. Com isto, tem-se uma boa visualização da função de roteamento, uma vez que a mensagem terá de passar por dois pares de STP's até atingir seu destino. Este cenário também requer que um grande número de *linksets* sejam criados, possibilitando explorar uma variedade de rotas alternativas induzindo defeito em enlaces.

Na simulação realizada neste cenário, a chamada é iniciada no nó SSP1, atendida no nó SSP2 e desligada no nó SSP1. As figuras D.1, D.2 e D.3 do Apêndice D ilustram, respectivamente, o início, atendimento e desligamento da chamada simulada. Estas figuras mostram, também, a tragetória percorrida pelas mensagens, onde as setas pretas indicam a tragetória das mensagens enviadas pelo nó SSP1 (IAM e REL) e as setas brancas, a tragetória das mensagens enviadas pelo nó SSP2 (ACM, ANM e RLC).

A chamada foi repetida neste cenário após a indução de defeito em um dos enlaces, o que fez com que as mensagens trafegassem por rotas alternativas, conforme ilustrado nas figuras D.4, D.5 e D.6 do Apêndice D.

4.7 Cenário 2 - Dois pares de STP's e quatro SSP's

Neste cenário, foi criada uma rede semelhante à anterior, adicionando-se a ela dois nós SSP, um conectado a cada par. Neste teste é possível observar como o simulador se comporta quando é inserido um maior número de nós SSP na rede. Neste cenário, o principal ponto a se observar é o aumento do número de entradas nas tabelas de roteamento de cada nó STP. A chamada neste cenário foi realizada do nó SSP1 para o nó SSP4 e, assim como no cenário anterior, foi atendida e desligada. As figuras D.7, D.8 e D.9 mostram o andamento desta chamada.

Da mesma forma que o teste anterior, este teste também foi realizado com enlaces defeituosos,como mostram as figuras D.10, D.11 e D.12 do Apêndice D.

4.8 Cenário 3 - Três pares de STP's e dois SSP's

O último cenário de testes montado é um cenário também semelhante ao primeiro, porém com um par de nós STP a mais, inserido entre os outros dois pares. Neste cenário é possível ter uma visualização ainda melhor da função de roteamento com a inserção de mais um par de nós STP entre os dois conectados aos nós SSP. Também pode se explorar mais rotas alternativas em situações de defeito em enlaces. A chamada neste cenário foi apenas iniciada e desligada pelo nó SSP1, não havendo atendimento. As figuras D.13 e D.14 do Apêndice D mostram a chamada realizada.

Estes testes também foram realizados após a indução de defeito em enlaces, conforme

mostrado nas figuras D.15 e D.16 do Apêndice D.

5 *Conclusões*

A proposta deste trabalho foi o desenvolvimento de um simulador com o objetivo de auxiliar o ensino do conteúdo de SS7 em cursos da área de telecomunicações. Optou-se por simular um subconjunto das funções do SS7, o que resultou em um simulador com funcionamento simples, mas que possibilita a visualização dos eventos ocorridos em uma rede SS7 associados às funções implementadas.

A função de sequenciamento de mensagens, do protocolo MTP2, foi implementada por completo, uma vez que as mensagens possuem os números de sequência FSN e BSN e estes são incrementados conforme a progressão da troca de mensagens no simulador. No entanto, estes números de sequência não estão servindo para fins de reconhecimento de mensagens e correção de erros, funções não implementadas no SS7Sim. Também devido à não-implementação das funções de detecção e correção de erros, um *linkset* defeituoso não é recuperado no SS7Sim.

Entre as funções do SMH, do protocolo MTP3, a função de discriminação foi implementada através da comparação entre os o campo DPC de uma mensagem e o PC do nó que a recebe pode cumprir este papel. A função de distribuição foi implementada parcialmente, pois apenas mensagens ISUP são trocadas no SS7Sim. Para que esta função seja implementada por completo, é necessária a implementação das funções de outros protocolos de nível 4, como TCAP e SCCP, além das demais mensagens ISUP. Para permitir o encaminhamento das mensagens nos nós STP, foi implementada a função de roteamento, através de tabelas de roteamento. Estas tabelas são contruidas a medida que são estabelecidas as conexões entre os nós da rede. Cada rota contida na tabela de roteamento possui uma prioridade, sendo a rota de mais alta prioridade escolhida para o encaminhamento da mensagem. As prioridades para rotas são calculadas com base no número de saltos até o nó destino. Para evitar a ocorrência de *loops* (mensagens sendo trocadas infinitamente entre nós STP) as prioridades para rotas são calculadas logo antes do início de uma simulação, quando se assume que não será feita nenhuma outra alteração

no cenário. O algoritmo de roteamento implementado funciona corretamente para redes pequenas, mas não foi testado para redes maiores pois o SS7Sim não prevê simulações em redes de grande porte, uma vez que não é possível a criação de uma estrutura hierárquica utilizando *links* D, E e F. Também por este motivo, os nós SSP e SCP não realizam a função de roteamento e, portanto, não possuem tabelas de roteamento.

No caso do protocolo ISUP, foi implementada a troca de mensagens em uma chamada básica, no entanto, existem outros tipos de mensagem ISUP que podem ser enviadas em diferentes situações e que agregariam valor à simulação. Além disto, mais campos de uma mensagem ISUP podem ser mostrados na inspeção de uma mensagem, uma vez que a implementação atual mostra um número reduzido de informações ISUP.

Para o desenvolvimento de uma interface gráfica, a biblioteca NetBeans Visual Library se mostrou bastante vantajosa, uma vez que permitiu a criação de uma interface gráfica amigável e interativa. Dentre os recursos providos por esta biblioteca, destacam-se: a funcionalidade *drag-and-drop*, de fácil implementação nesta biblioteca e fundamental para que se pudesse possibilitar a criação do cenário pelo usuário; e os recursos de animação, fundamentais para que a troca de mensagens fosse claramente visível ao usuário. Além disto, a NetBeans Visual Library provê uma série de recursos para que a interface já implementada possa ser aprimorada em trabalhos futuros. Entre os possíveis aprimoramentos estão: permitir ao usuário selecionar um conjunto de nós com um arraste do *mouse*, bom como copiar e colar os nós selecionados; facilitar a criação de *linksets* possibilitando que estes sejam criados com apenas um clique no nó origem e no nó destino; e fazer com que haja persistência na tragetória das mensagens, para facilitar a visualização.

Com os resultados obtidos com o SS7sim, pode se dizer que ele já auxilia o entendimento do conteúdo SS7 por parte dos estudantes, uma vez que é capaz de simular trocas de mensagens e parte dos eventos ocorridos em uma rede SS7. Porém, o resultado obtido neste trabalho pode ser aprimorado, uma vez que muito do que ocorre em uma rede SS7 ainda não foi implementado no SS7sim. Tais aprimoramentos são propostos na seção seguinte como trabalhos futuros (para informações sobre autorização do uso do SS7Sim e de seu código fonte, ver Apêndice E).

5.1 Possibilidades de trabalhos futuros

Para um melhor entendimento do SS7, há funções importantes do mesmo que podem ser implementadas. Como trabalhos futuros, propõe-se a implementação das seguintes funcionalidades:

- Envio de mensagens FISU para manter o o sincronismo da rede e também para o reconhecimento de MSU's.
- Monitoramento de enlaces através do contador SUERM, que considera o enlace como defeituoso após uma sequência de erros.
- Detecção e correção de erros - implementar retransmissão de mensagens e envio de mensagens LSSU para indicar mudança no estado de um enlace.
- Seleção de links para o envio de mensagens - no SS7sim não há regra para esta seleção, é selecionado qualquer link que esteja operacional, de acordo com a tabela de roteamento de cada nó STP. Além disto, não é especificado qual dos links dentro do linkset foi selecionado para o envio da mensagem.
- Implementação de outras situações de chamada além de uma chamada básica, envolvendo diferentes tipos de mensagens ISUP, além do acréscimo de outros campos de uma mensagem ISUP para a visualização do usuário.
- Implementar a função de roteamento no nós SSP e SCP.
- Implementar as funções dos protocolos TCAP e SCCP.
- Implementar os demais tipos de links possíveis no SS7 (links D, E e F), para permitir a criação de estruturas hierárquicas.
- Opção de seleção de um conjunto de de nós via arraste do *mouse*, bem como a opção de copiar e colar os nós selecionados.
- Facilitar a criação de *linksets* - idealmente isto se daria através de um clique no nó origem e outro no nó destino.
- Persistência na tragetória das mensagens.

APÊNDICE A – Detalhamento dos casos de uso do SS7Sim

A.1 Criação de cenário

- O usuário arrasta os componentes desejados para o cenário.
- O usuário estabelece as conexões entre os componentes.

A.2 Simulação

- O sistema verifica se o cenário está corretamente montado e segue caso esteja.
- O usuário inicia a simulação, iniciando uma chamada.
- O sistema simula a troca das mensagens de estabelecimento de chamada e aguarda ação do usuário.
- O usuário seleciona a opção de atendimento ou desligamento de chamada em um dos nós SSP envolvidos na chamada.
- O sistema simula a troca de mensagens de acordo com o procedimento escolhido.

A.3 Estabelecimento de chamada

- O usuário seleciona a opção `estabelecer chamada` em um nó SSP.
- O sistema simula a troca das mensagens de estabelecimento de chamada.

A.4 Indução de defeito em enlace

- O usuário seleciona a opção `induzir defeito` em um *linkset*.
- O sistema torna o *linkset* defeituoso.
- O sistema atualiza as tabelas de roteamento dos nós STP alterando a prioridade das rotas envolvendo este *linkset*.

A.5 Desligamento de chamada

- O usuário seleciona a opção `desligar chamada` em um nó SSP.
- O sistema simula a troca de mensagens envolvidas no desligamento de uma chamada.

A.6 Visualização de mensagens

- O usuário seleciona a opção `inspecionar` em uma mensagem.
- O sistema mostra as informações da mensagem em uma janela.

APÊNDICE B – Metodos do SS7Sim

B.1 Métodos que implementam funções do protocolo MTP3

B.1.1 Método `processMessage` na classe `SspWidget`

```
1  @Override
2     public MessageProcessingAction processMessage(SignalUnitWidget message)
          {
3         this.messageQueue.remove(message);
4         ++this.backwardSeqNumberGen;
5         if (!(message instanceof MsuWidget)) {
6             return MessageProcessingAction.DISCARDED;
7         } else {
8             MsuWidget msu = (MsuWidget) message;
9             if (!msu.getDpc().equals(this.getPointCode())) {
10                return MessageProcessingAction.DISCARDED;
11            } else {
12
13                switch (msu.getMessageType()) {
14                    case IAM:
15                        this.processIam(msu);
16                        return MessageProcessingAction.PROCESSED;
17                    case REL:
18                        this.processRel(msu);
19                        return MessageProcessingAction.PROCESSED;
20                    default:
21                        break;
22                }
23                return MessageProcessingAction.PROCESSED;
24            }
```

```
25          }
26      }
```

B.1.2 Método `processMessage` na classe `StpWidget`

```
1   @Override
2       public MessageProcessingAction processMessage(SignalUnitWidget message)
            {
3           this.messageQueue.remove(message);
4           if (message instanceof MsuWidget) {
5               MsuWidget msu = (MsuWidget) message;
6               if (!(msu.getDpc().equals(this.getPointCode()))) {
7                   this.forwardMessage(msu);
8                   return MessageProcessingAction.FORWARDED;
9               }
10          }
11          return MessageProcessingAction.DISCARDED;
12      }
```

B.1.3 Código da classe `RoutingTable`

```
1
2   public class RoutingTable {
3
4       private ArrayList<Entry> entries;
5
6       public RoutingTable(){
7           this.entries = new ArrayList<Entry>();
8       }
9
10      public void addEntry(Entry entry){
11          this.entries.add(entry);
12      }
13
14      public boolean removeEntry(Entry entry){
15          return this.entries.remove(entry);
16      }
17
18      public ArrayList<Entry> getEntries() {
```

```
19          return entries;
20      }
21      public static class Entry {
22
23          private String label;
24          private String dpc;
25          private String route;
26          private int routePriority;
27
28          public Entry(String label, String dpc, String route) {
29              this.label = label;
30              this.dpc = dpc;
31              this.route = route;
32              this.routePriority = 0;
33          }
34
35          public Entry(String label, String dpc, String route, int
                routePriority) {
36              this.label = label;
37              this.dpc = dpc;
38              this.route = route;
39              this.routePriority = routePriority;
40          }
41          public String getDpc() {
42              return dpc;
43          }
44          public String getLabel() {
45              return label;
46          }
47          public String getRoute() {
48              return route;
49          }
50          public int getRoutePriority() {
51              return routePriority;
52          }
53          public void setRoutePriority(int routePriority) {
54              this.routePriority = routePriority;
55          }
56
57      }
58  }
```

B.1.4 Método `addToRoutingTable`

```
1  private void addToRoutingTable(Ss7Widget newDestination) throws
       MissingLinkException {
2        SimScene scene = (SimScene) this.getScene();
3        if (scene.findNodeEdges((MyNode) scene.findObject(this), true, true)
             .isEmpty() || scene.findNodeEdges((MyNode) scene.findObject(
             newDestination), true, true).isEmpty()) {
4            throw new MissingLinkException("Não há links entre o STP " +
                 this.getLabelWidget().getLabel() + "e o nó " + newDestination
                 .getLabelWidget().getLabel());
5        } else {
6            for (String edge : scene.findNodeEdges((MyNode) scene.findObject
                 (this), true, true)) {
7                Linkset ls = (Linkset) scene.findWidget(edge);
8                if (scene.getEdgeSource(edge).equals(scene.findObject(this))
                     && scene.getEdgeTarget(edge).equals(scene.findObject(
                     newDestination))
9                        || scene.getEdgeSource(edge).equals(scene.findObject(
                             newDestination)) && scene.getEdgeTarget(edge).
                             equals(scene.findObject(this))) {
10                   RoutingTable.Entry entry = new RoutingTable.Entry(
                         newDestination.getLabelWidget().getLabel(),
                         newDestination.getPointCode(), ls.getLsid());
11                   if (!this.containsRoute(entry.getLabel(), entry.getDpc(),
                          entry.getRoute())) {
12                       this.getRoutingtable().addEntry(entry);
13                   }
14                   if (newDestination instanceof StpWidget) {
15
16                       StpWidget newStp = (StpWidget) newDestination;
17                       for (RoutingTable.Entry e : newStp.getRoutingtable().
                             getEntries()) {
18                           if (!this.containsRoute(e.getLabel(), e.getDpc(),
                                 ls.getLsid()) && !e.getDpc().equals(this.
                                 getPointCode())) {
19                               this.routingtable.addEntry(new RoutingTable.
                                     Entry(e.getLabel(), e.getDpc(), ls.getLsid
                                     ()));
20                           }
21                       }
```

```
22                  }
23              }
24
25          }
26
27      }
28   }
```

B.1.5 Método `removeFromRoutingTable` para a remoção de nós SS7

```
1  private void removeFromRoutingTable(Ss7Widget dest) {
2
3    for (RoutingTable.Entry e :
4   new ArrayList<RoutingTable.Entry>(this.routingtable.getEntries())) {
5        if (e.getDpc().equals(dest.getPointCode()) &&
6   e.getLabel().equals(dest.getLabelWidget().getLabel())) {
7            this.routingtable.removeEntry(e);
8        }
9    }
10
11 }
```

B.1.6 Método `removeFromRoutingTable` para a remoção de linksets

```
1  private void removeFromRoutingTable(Linkset route){
2    for(RoutingTable.Entry e :
3   new ArrayList<RoutingTable.Entry>(this.routingtable.getEntries()))
4        if(e.getRoute().equals(route.getLsid()))
5            this.routingtable.removeEntry(e);
6  }
```

B.1.7 Método `updateRoutePriorities`

```
1   private void uptadeRoutePriorities(Linkset route, int newPriority) {
2       for (RoutingTable.Entry e : this.routingtable.getEntries()) {
3           if (e.getRoute().equals(route.getLsid())) {
4               e.setRoutePriority(newPriority);
5           }
6       }
7    }
```

B.2 Métodos que implementam funções do protocolo ISUP

B.2.1 Método `forwardMsu`

```
1   private void forwardMsu(MsuWidget msu) {
2         SimScene scene = (SimScene) this.getScene();
3         Ss7Widget w;
4         Ss7Widget destination = null;
5         Linkset link = null;
6         for (MyNode n : scene.getNodes()) {
7             w = (Ss7Widget) scene.findWidget(n);
8             if (w.getPointCode().equals(msu.getDpc())) {
9                 destination = w;
10                int priority = 0;
11                for (String edge :
12  scene.findNodeEdges((MyNode) scene.findObject(this), true, true)) {
13                    Linkset l = (Linkset) scene.findWidget(edge);
14                    if (this.containsRoute(w.getLabelWidget().getLabel(),
15   msu.getDpc(), l.getLsid())) {
16                        if (this.getRoutePriority(w.getLabelWidget().getLabel
                              (),
17    msu.getDpc(), l.getLsid()) > priority) {
18                            link = l;
19                            priority = this.getRoutePriority(
20    w.getLabelWidget().getLabel(), msu.getDpc(), l.getLsid());
21                            msu.setCurrentLinksetId(l.getLsid());
22                        }
23                    }
24                }
```

```
25            }
26        }
27        this.sendMessage(msu, link, destination);
28    }
```

B.2.2 Método `processIam`

```
1  private void processIam(MsuWidget iam) {
2        SimScene scene = (SimScene) this.getScene();
3        Ss7Widget dest = scene.findSs7Widget(iam.getOpc());
4        MsuWidget acm = new MsuWidget(this.getPointCode(), dest.getPointCode
             (),
5  this.getOutputLink().getLsid(), MessageType.IsupMessageType.ACM, scene);
6        acm.setLabel("ACM");
7        acm.setCic(cicIdGen);
8        scene.addToMainLayer(acm);
9        scene.validate();
10       this.sendMessage(acm, dest);
11    }
```

B.2.3 Método `processRel`

```
1  private void processRel(MsuWidget rel) {
2        SimScene scene = (SimScene) this.getScene();
3        Ss7Widget dest = scene.findSs7Widget(rel.getOpc());
4        MsuWidget rlc = new MsuWidget(this.getPointCode(), dest.getPointCode
             (),
5  this.getOutputLink().getLsid(), MessageType.IsupMessageType.RLC, scene);
6        rlc.setLabel("RLC");
7        rlc.setCic(cicIdGen);
8        scene.addToMainLayer(rlc);
9        scene.validate();
10       this.sendMessage(rlc, dest);
11    }
```

B.2.4 Método `createMessage`

```
1  public SignalUnitWidget createMessage(MessageType type, Ss7Widget
       destination) {
2      switch (type) {
3          case FISU:
4              return new FisuWidget(this.getScene());
5          case MSU:
6              return new MsuWidget(this.getPointCode(),
7  destination.getPointCode(), this.getOutputLink().getLsid(), this.getScene
       ());
8          default:
9              return null;
10     }
11   }
```

APÊNDICE C – Diagramas de classes do SS7Sim

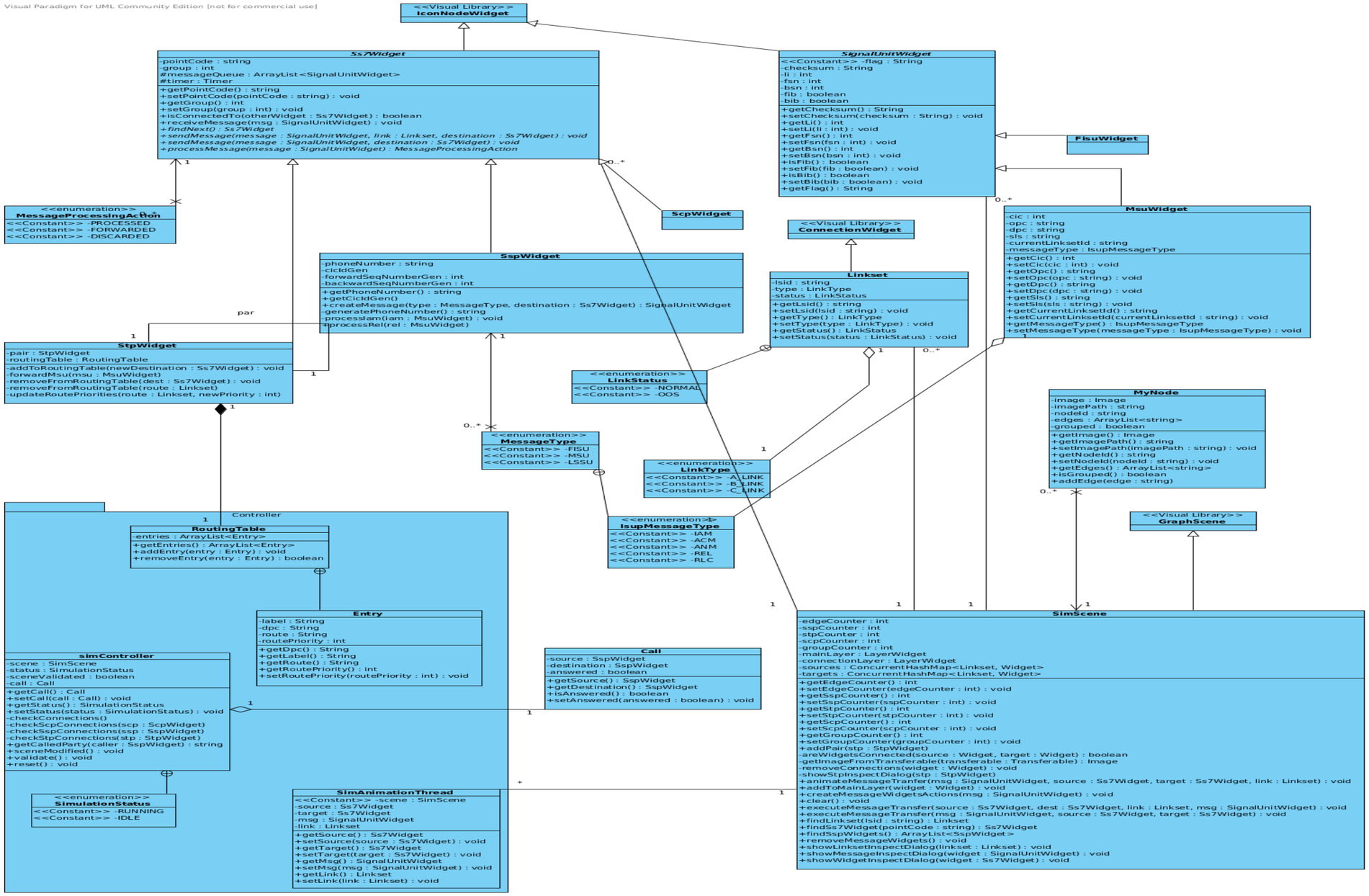


Figura C.1: Diagrama de classes do SS7Sim - geral

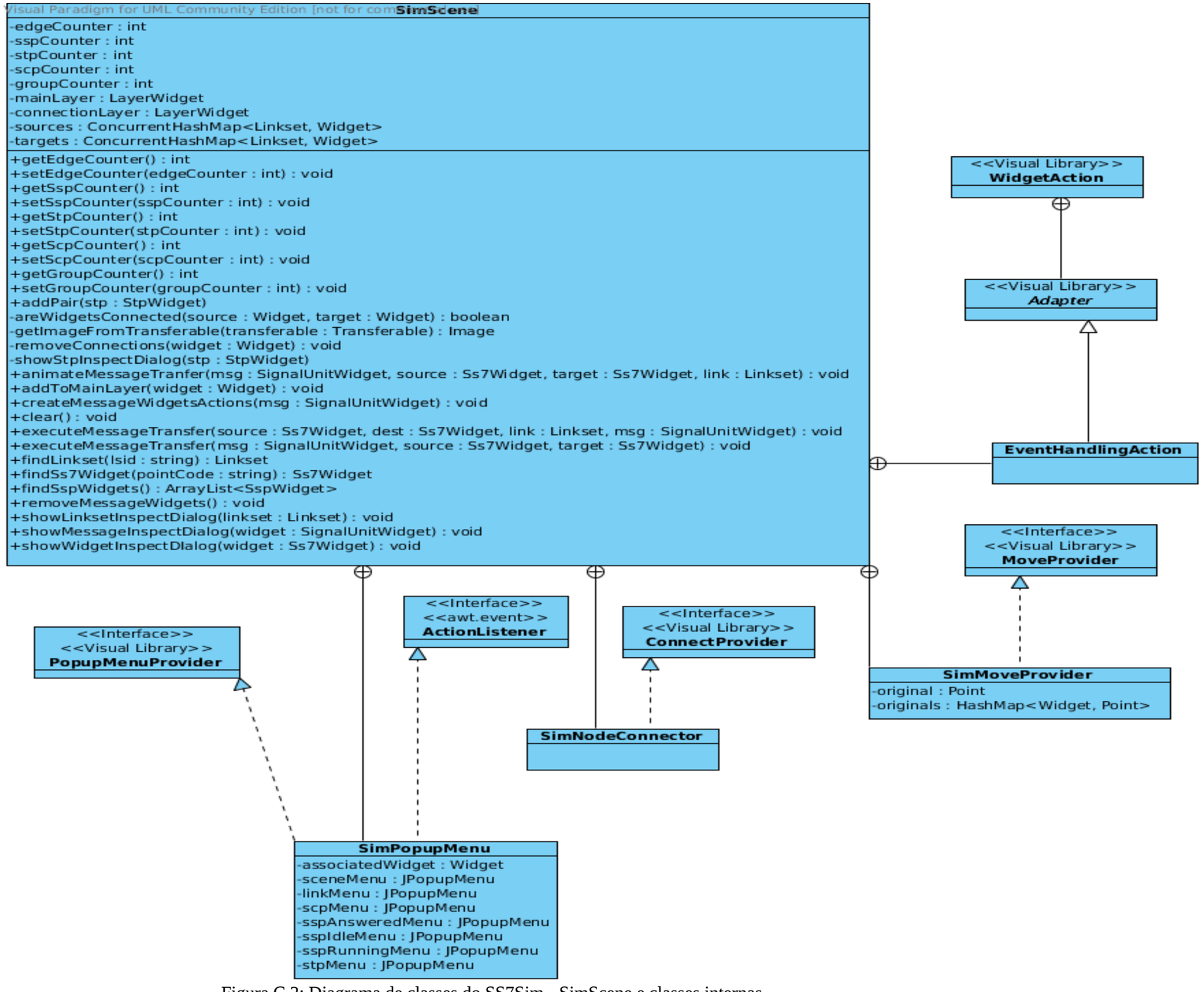


Figura C.2: Diagrama de classes do SS7Sim - SimScene e classes internas

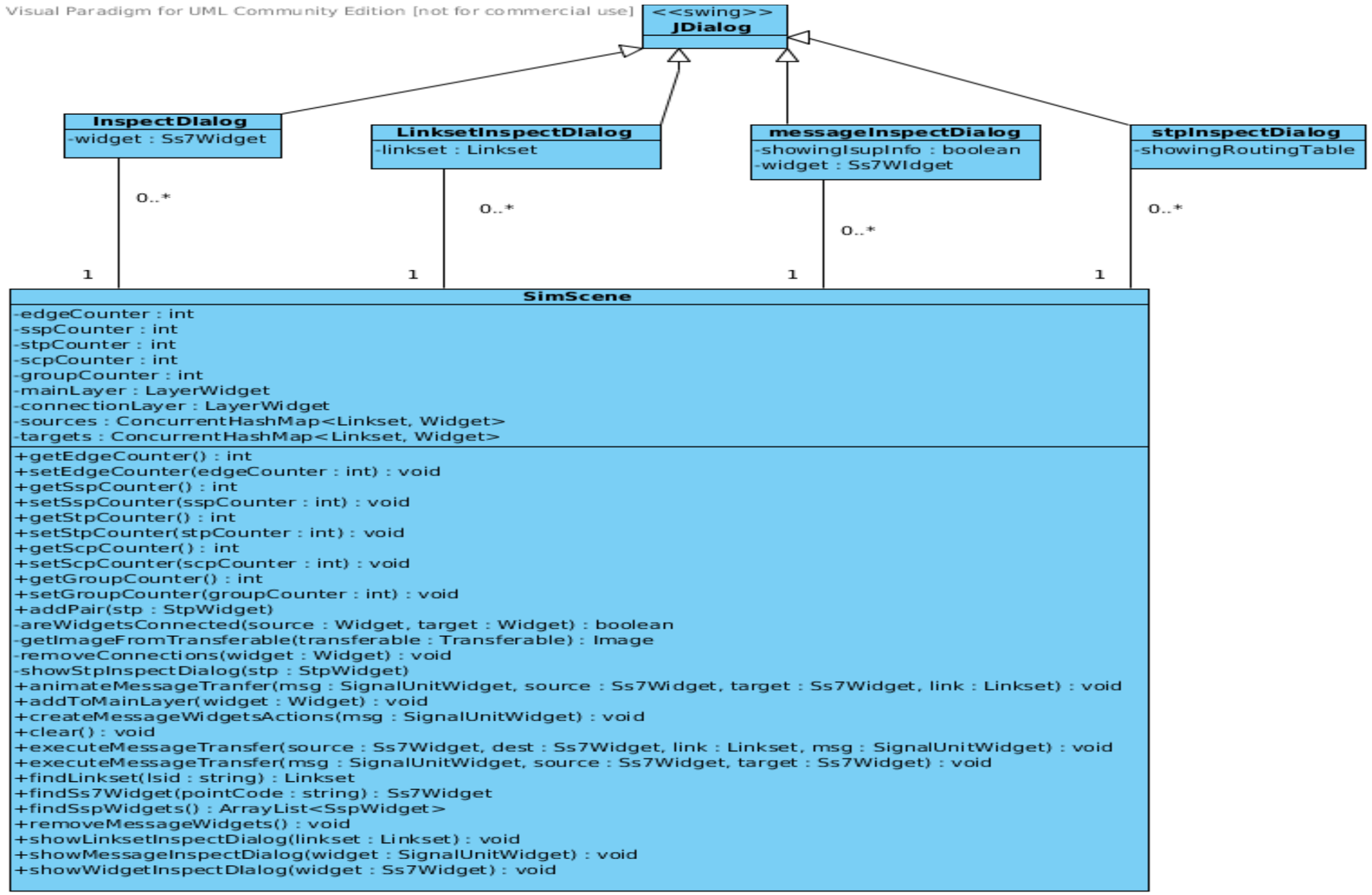


Figura C.3: Diagrama de classes do SS7Sim - janelas

APÊNDICE D – Cenários de testes

D.1 Cenário 1 - Dois pares de STP's e dois SSP's

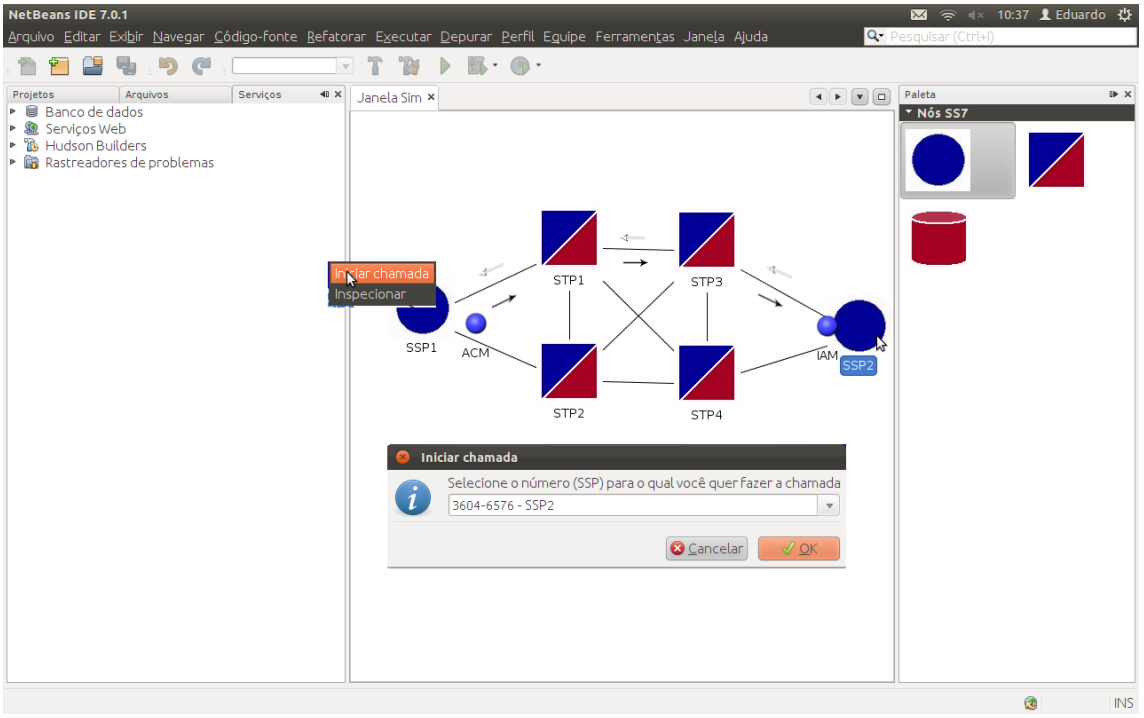


Figura D.1: Cenário 1 - início da chamada

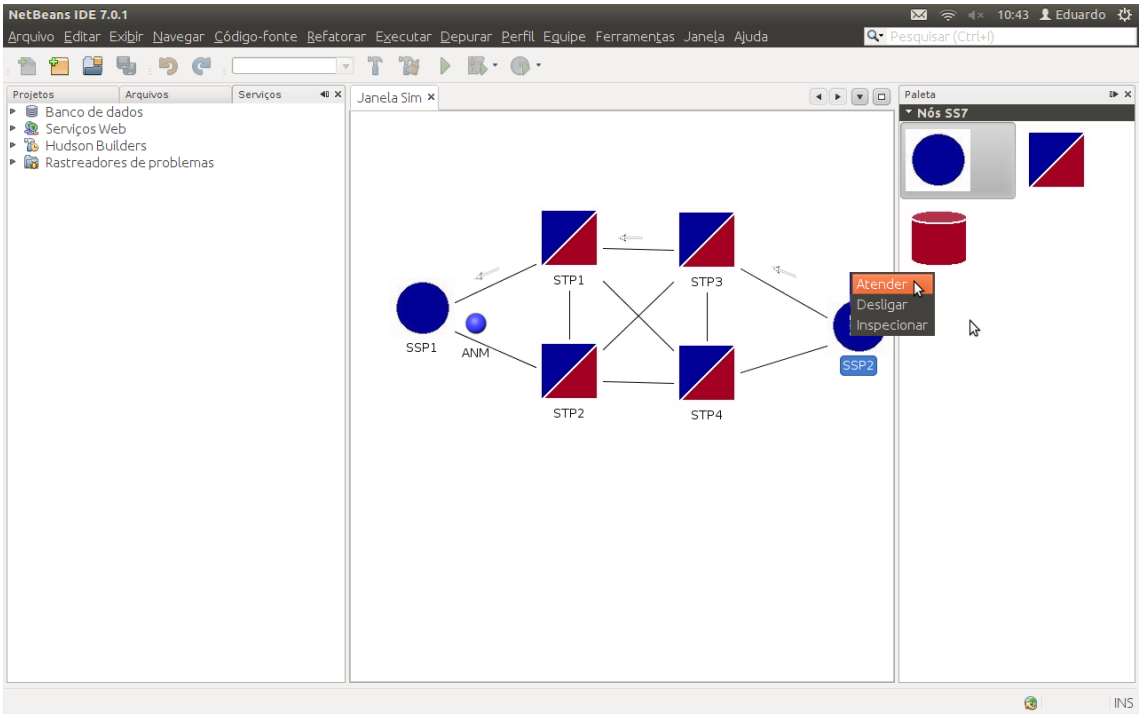


Figura D.2: Cenário 1 - Atendimento

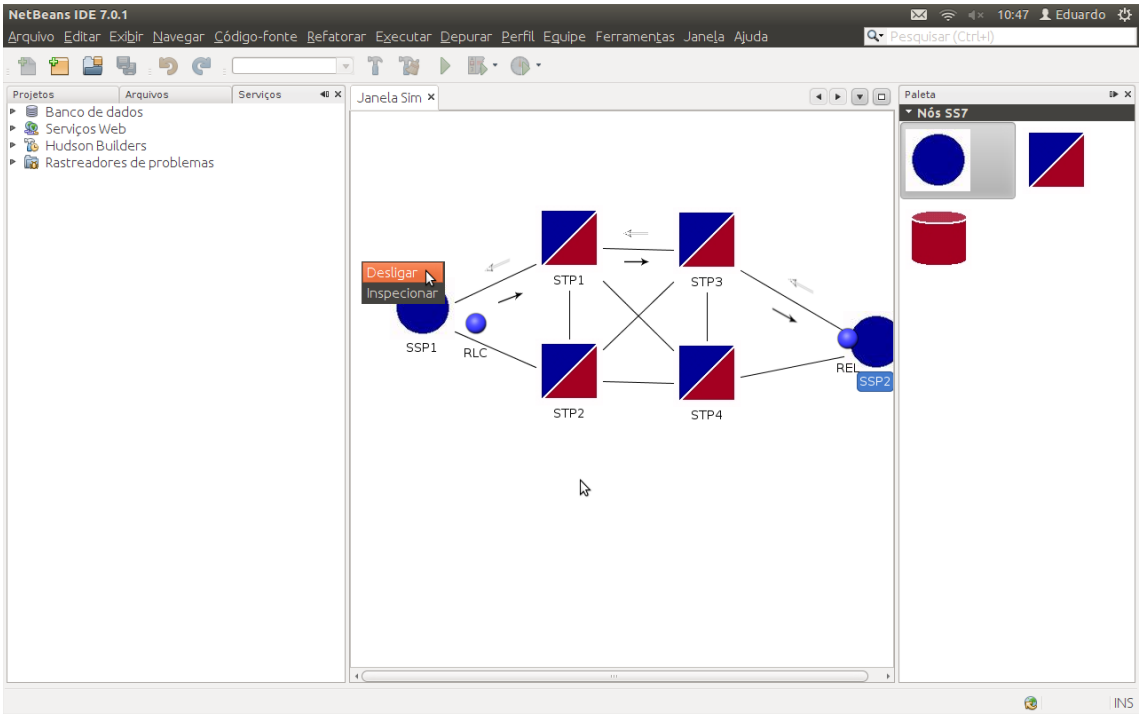


Figura D.3: Cenário 1 - Desligamento

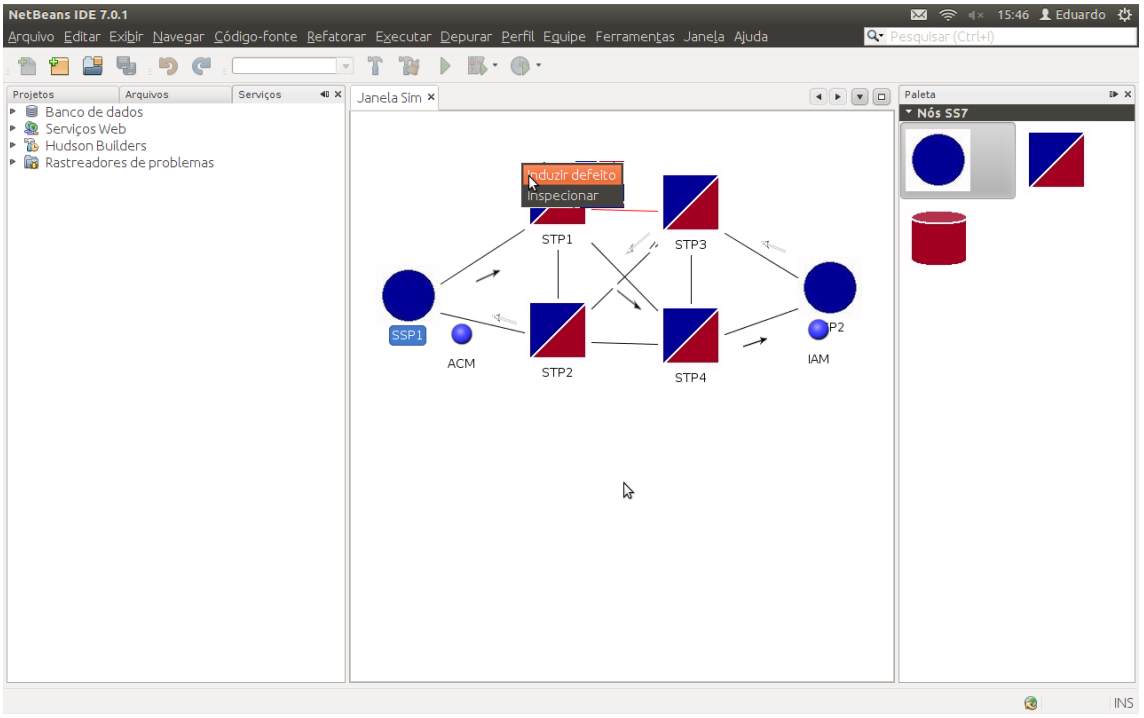


Figura D.4: Cenário 1 - início da chamada após indução de defeito em enlace

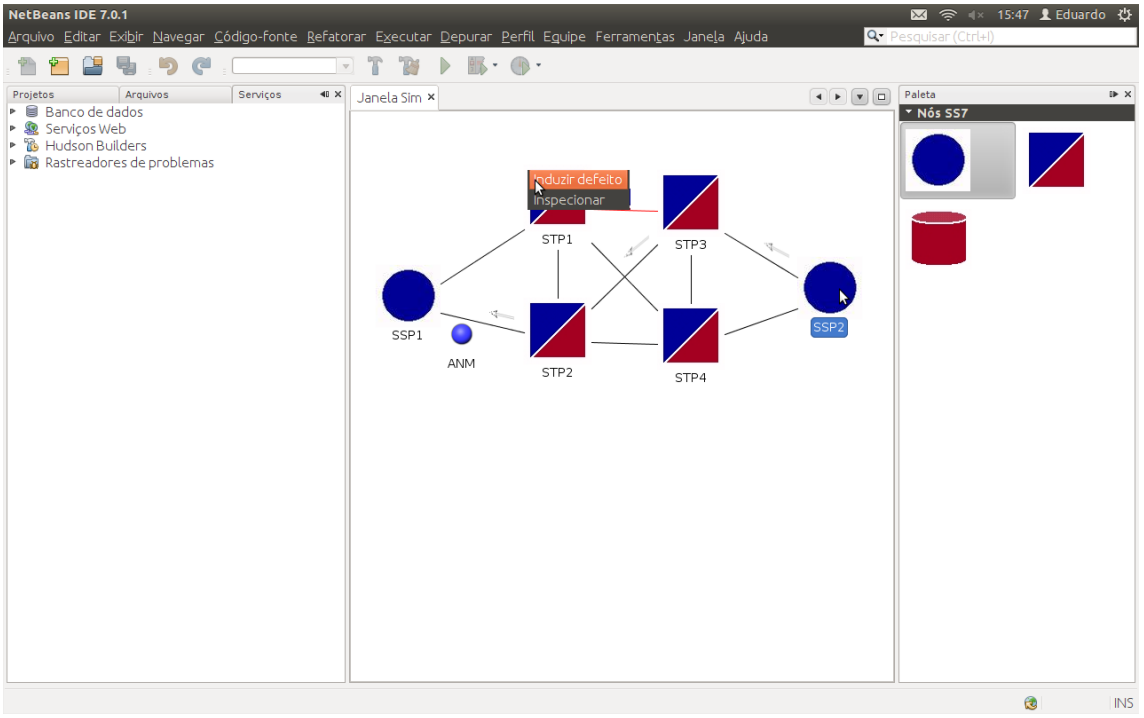


Figura D.5: Cenário 1 - atendimento após indução de defeito em enlace

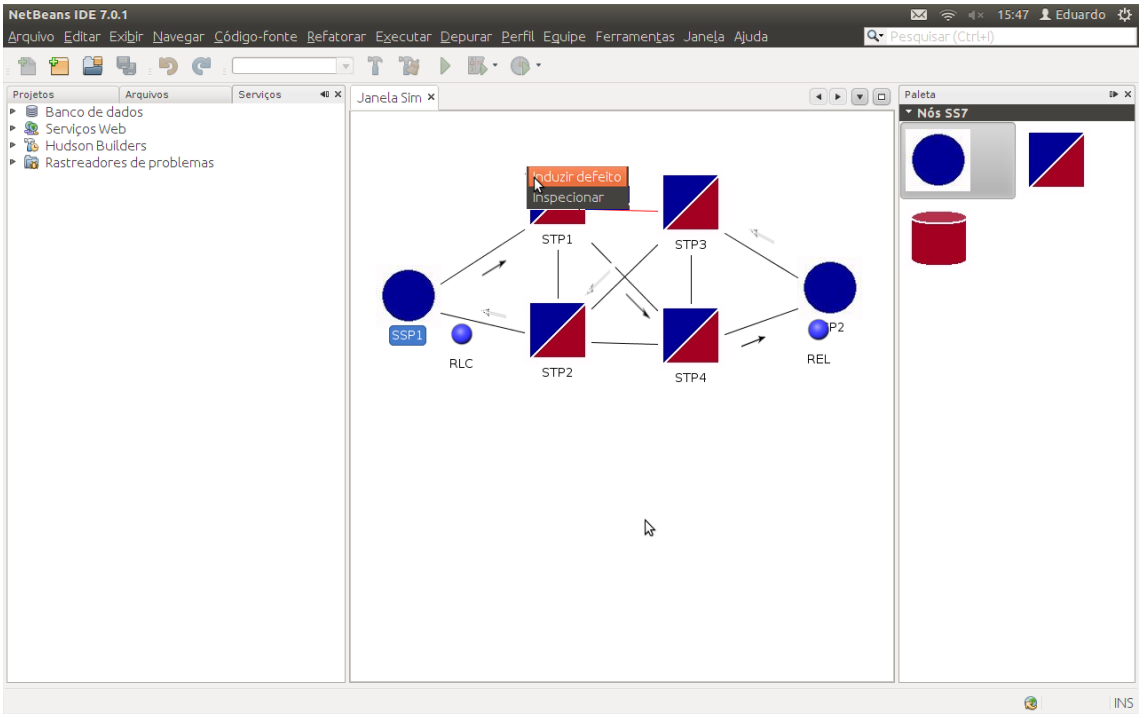


Figura D.6: Cenário 1 - desligamento após indução de defeito em enlace

D.2 Cenário 2 - Dois pares de STP's e quatro SSP's

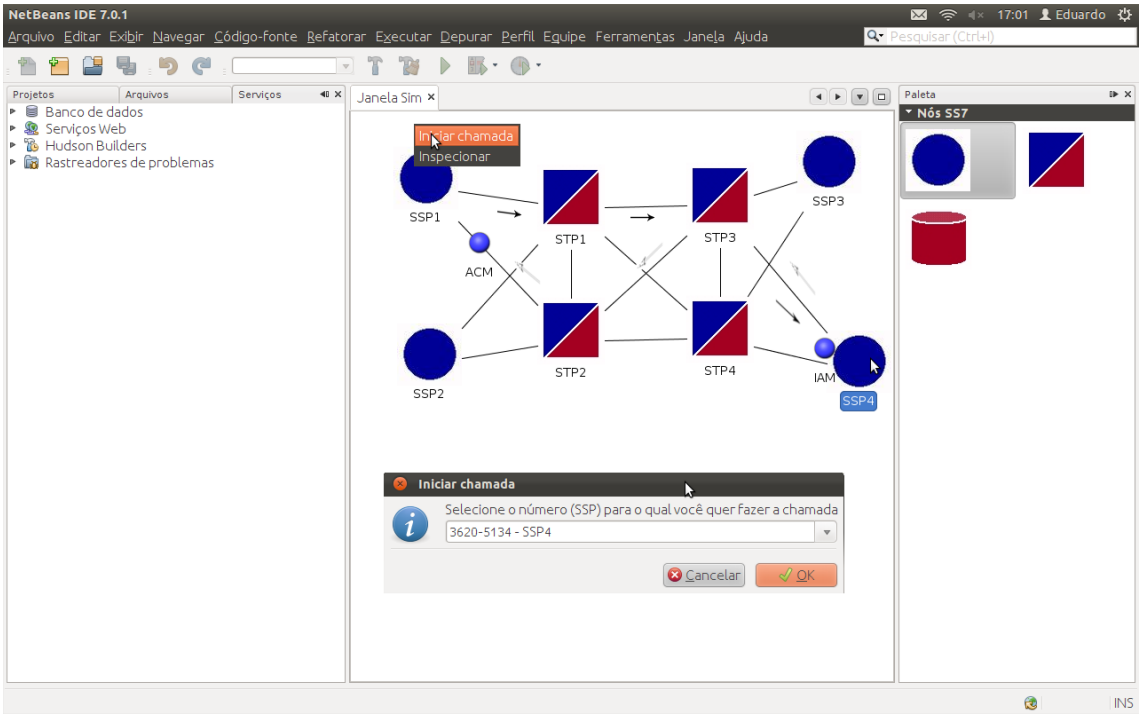


Figura D.7: Cenário 2 - início da chamada

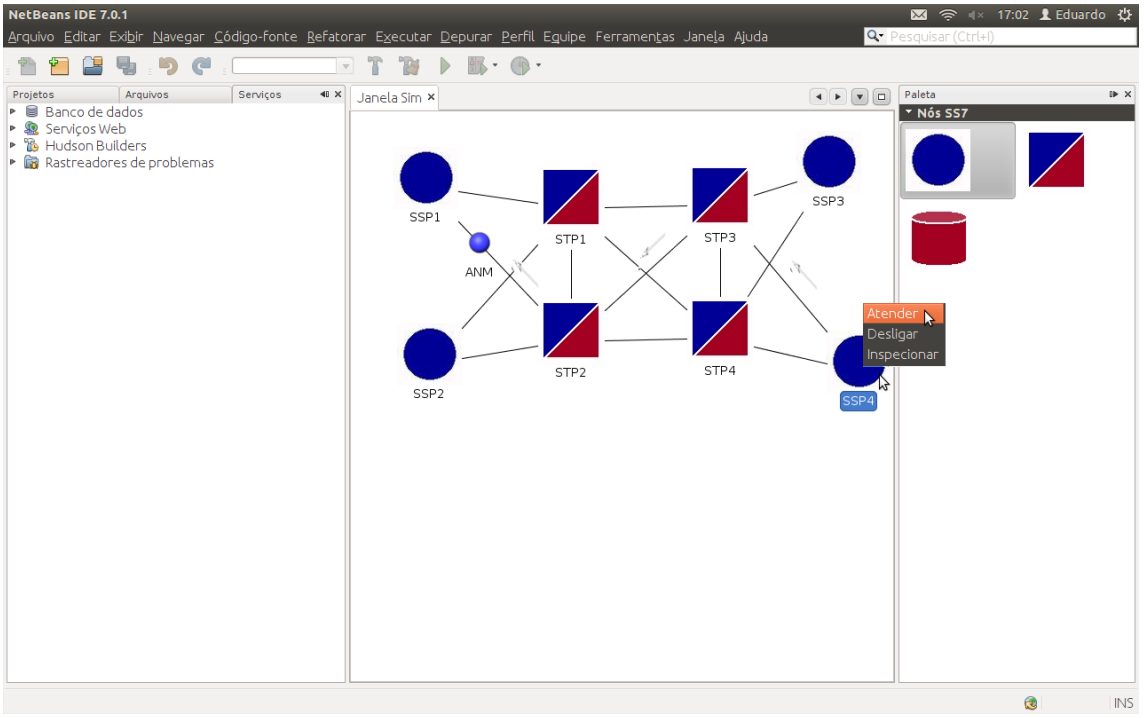


Figura D.8: Cenário 2 - Atendimento

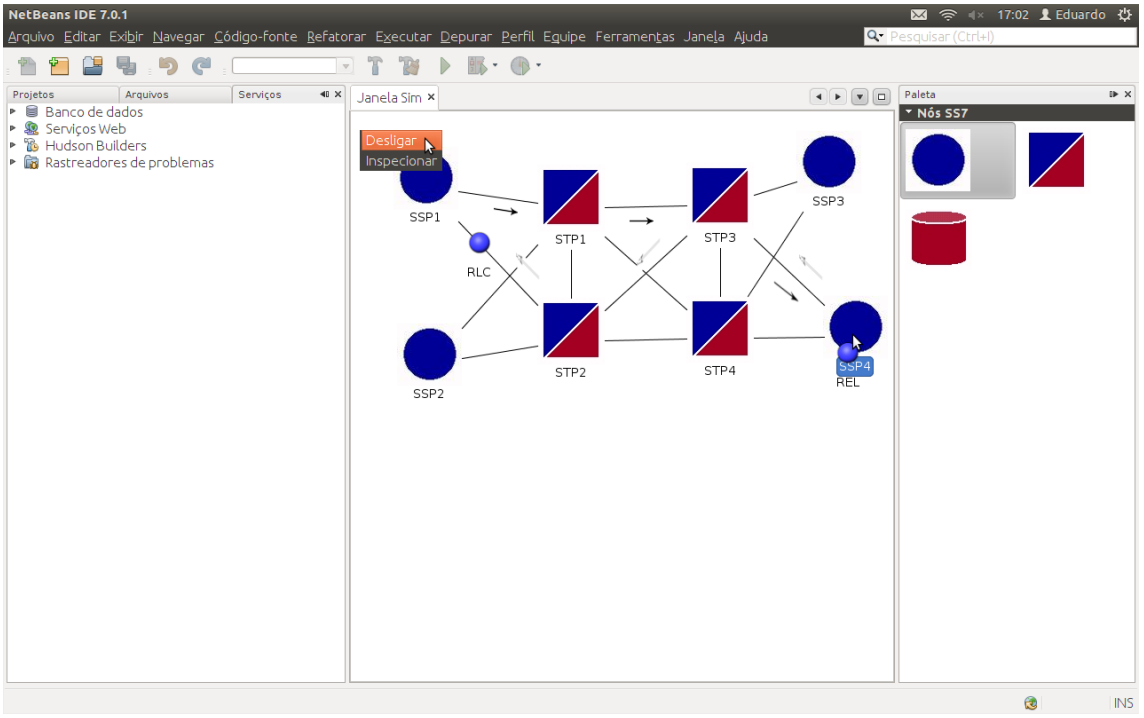


Figura D.9: Cenário 2 - Desligamento

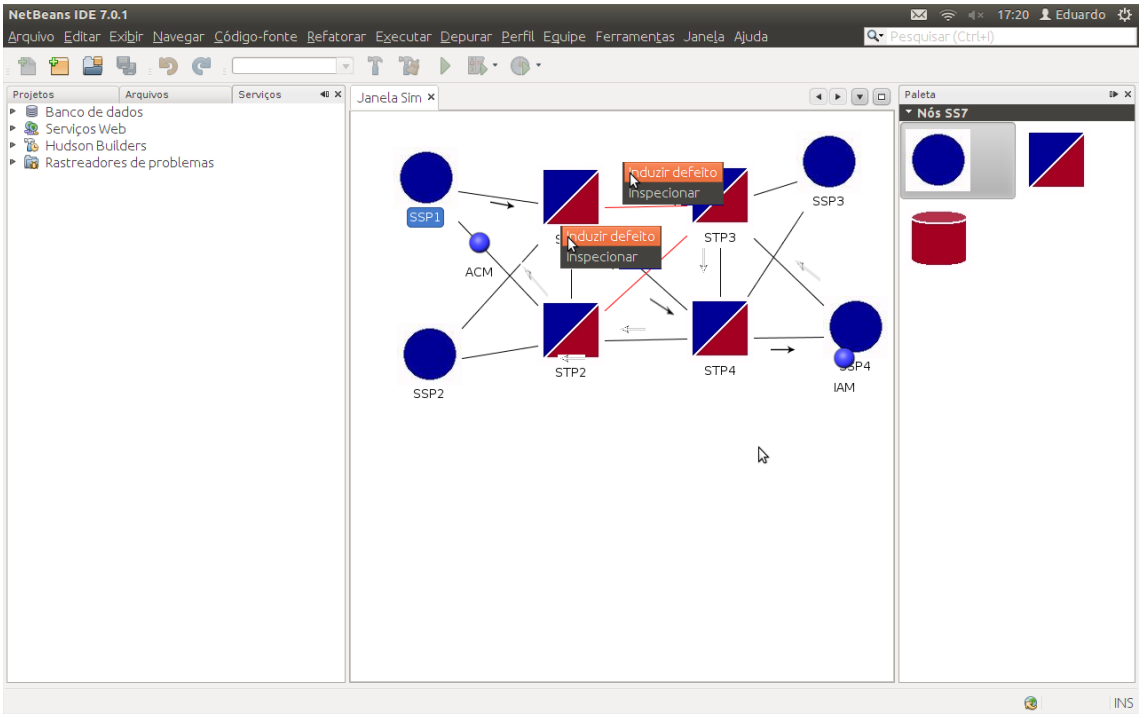


Figura D.10: Cenário 2 - início da chamada após indução de defeito em enlace

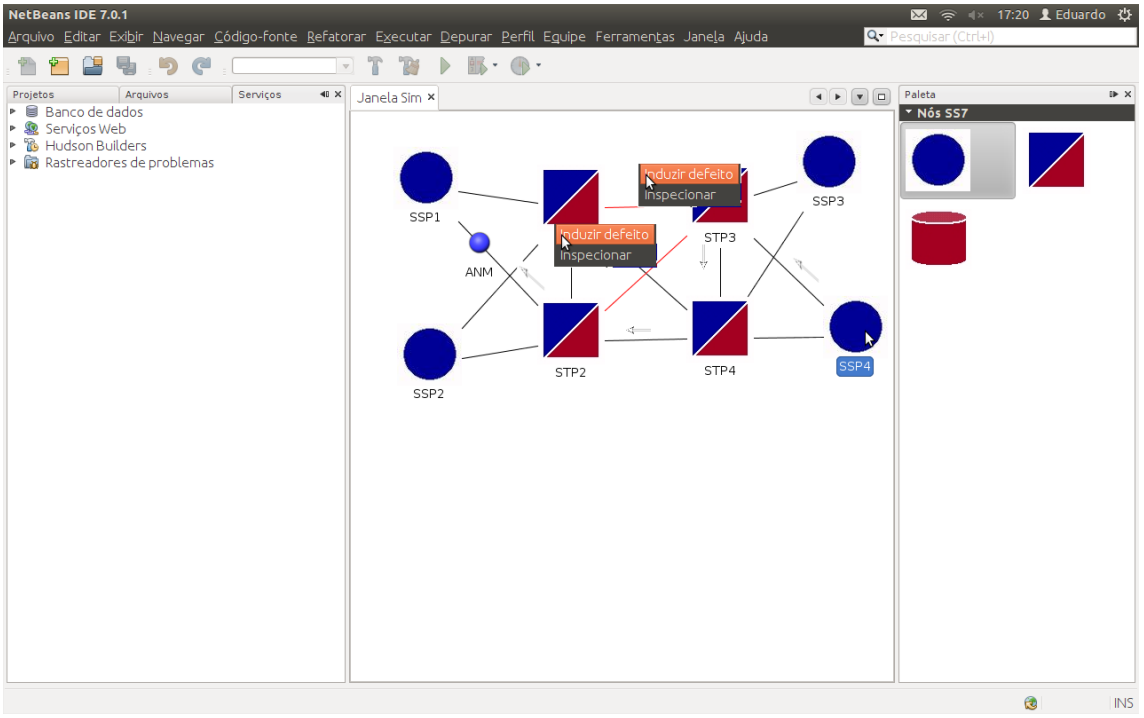


Figura D.11: Cenário 2 - atendimento após indução de defeito em enlace

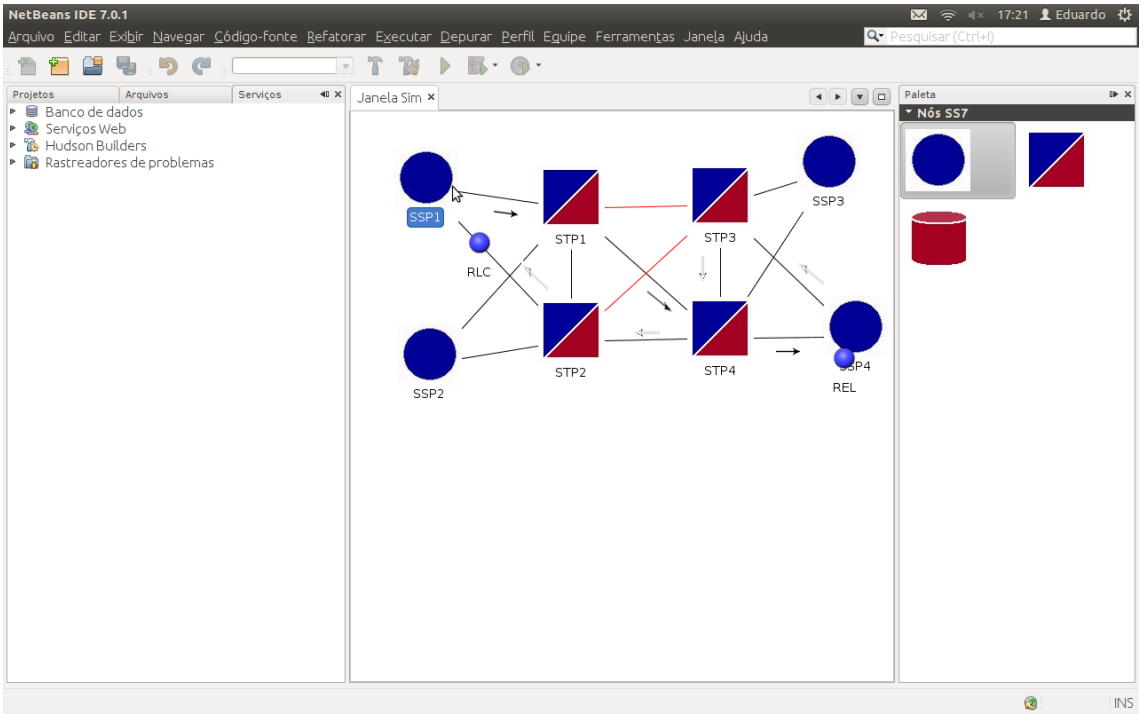


Figura D.12: Cenário 2 - desligamento após indução de defeito em enlace

D.3 Cenário 3 - Três pares de STP's e dois SSP's

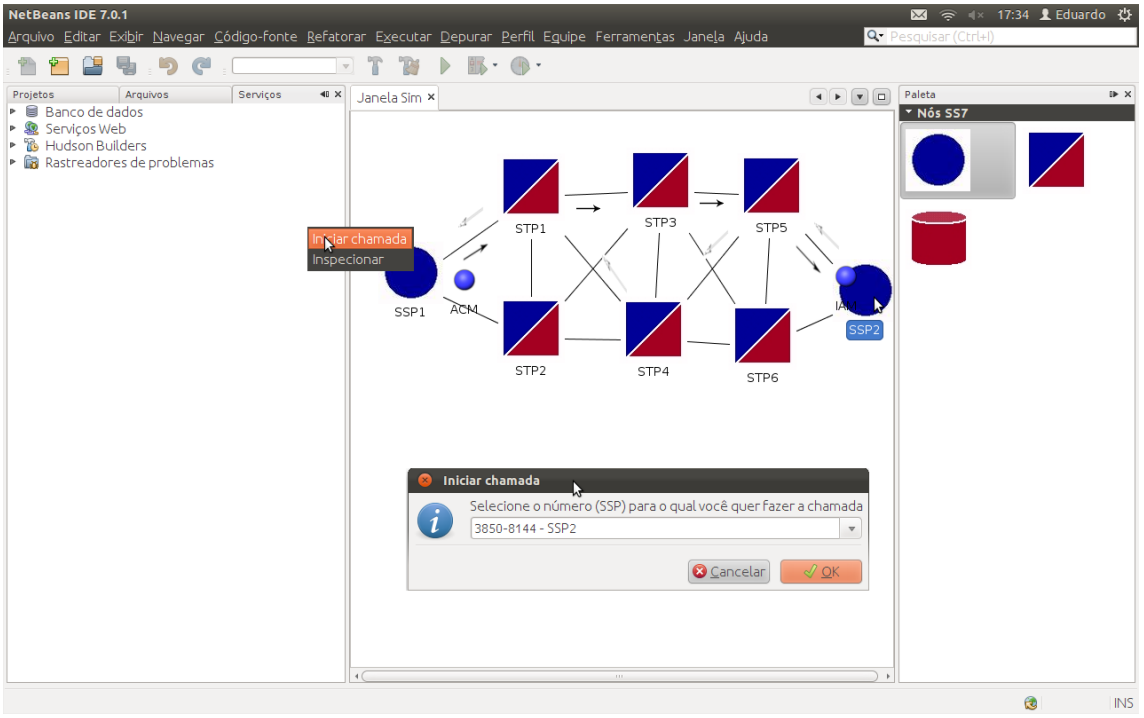


Figura D.13: Cenário 3 - início da chamada

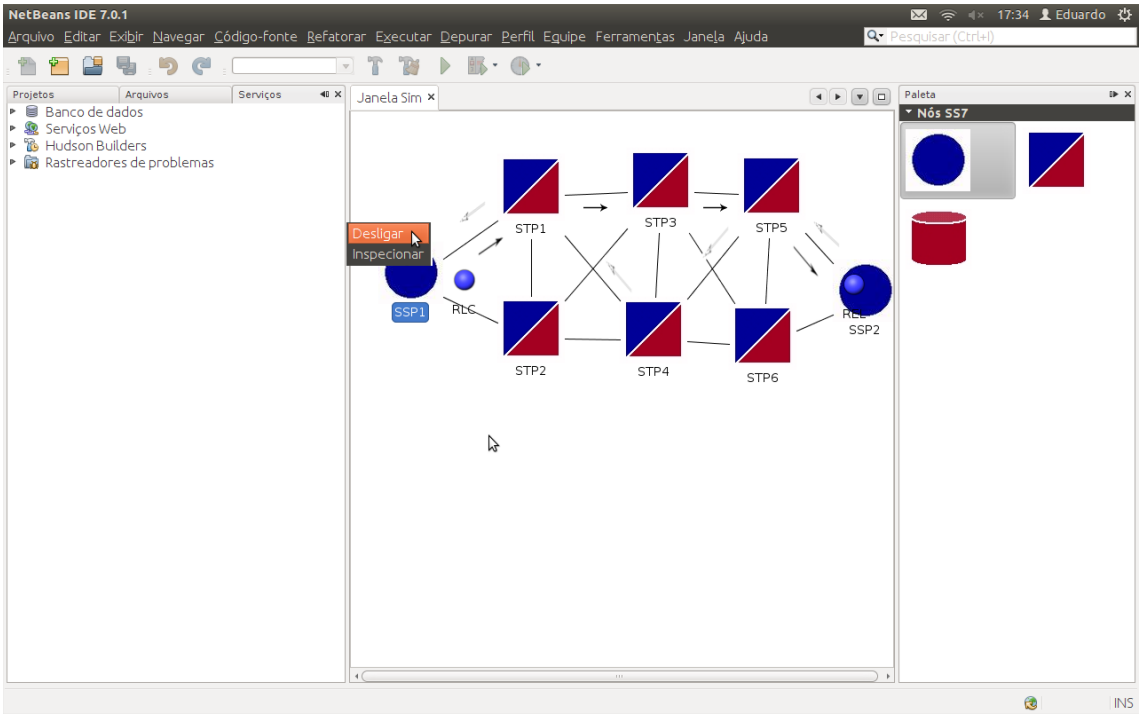


Figura D.14: Cenário 3 - Desligamento

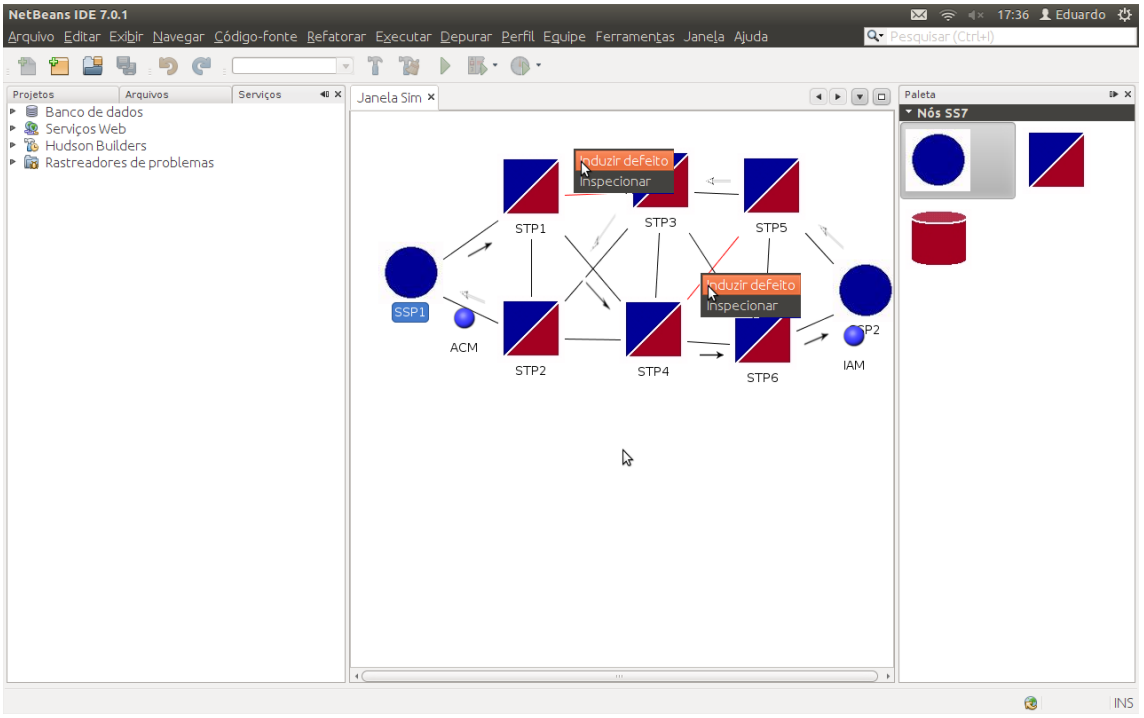


Figura D.15: Cenário 3 - início da chamada após indução de defeito em enlace

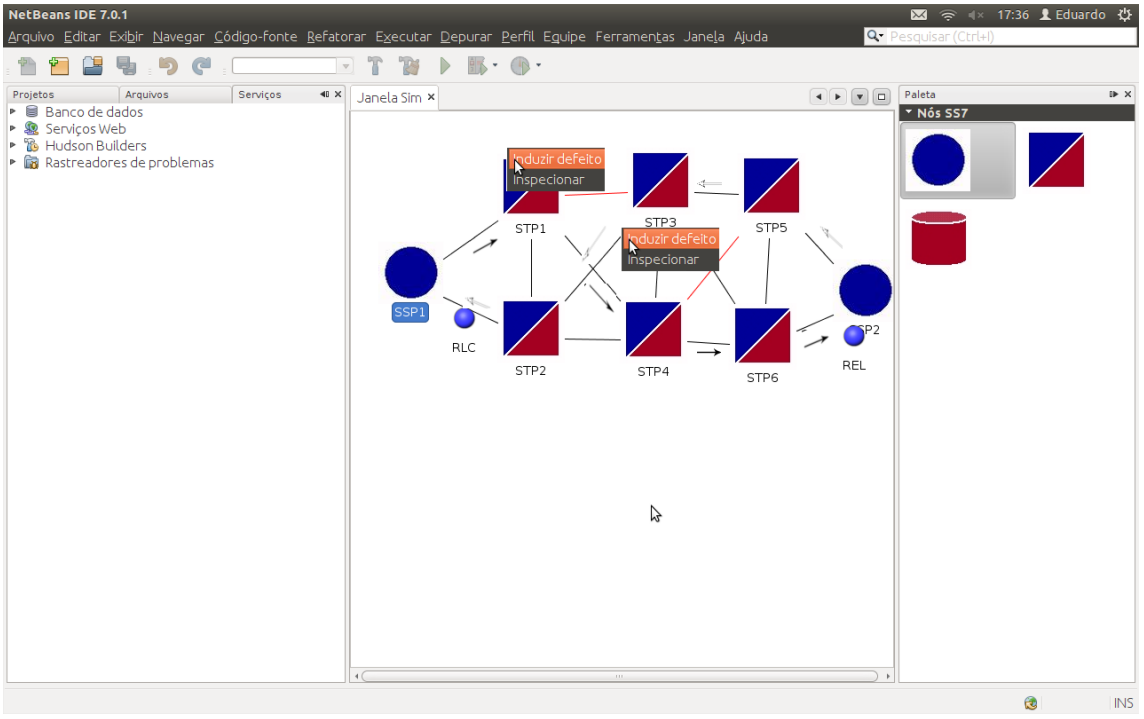


Figura D.16: Cenário 3 - desligamento após indução de defeito em enlace

APÊNDICE E – Autorização para uso do SS7Sim

Autorizo a utilização do simulador para o Sistema de Sinalização SS7 SS7Sim, incluindo alterações em seu código fonte, para fins didáticos ou de pesquisa acadêmica, desde que corretamente referenciada sua autoria e devidamente documentadas as alterações no código fonte.

Referências Bibliográficas

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. [S.l.]: Elsevier, Rio de Janeiro, 2002.

BORATTI, I. *Programação Orientada a Objetos em Java*. [S.l.]: Visual Books, Florianópolis, 2007.

BöCK, H. *The Definitive Guide to NetBeans Platform*. [S.l.]: Apress, 2009.

CISCO. The evolution of ss7. In: ____. 2012. Disponível em: <http://www.cisco.com/univercd/cc/td/doc/product/tel_pswt/vco_prod/ss7_fund/ss7fun01.htm#xtocid10>. Acesso em: 7 fev. 2012.

DRYBURGH, L.; HEWETT, J. *Signaling System No. 7 (SS7/C7): Protocol, Architecture, and Services*. [S.l.]: Cisco Press, 2004.

IBM. The ss7 protocol stack. In: ____. *SS7 user's guide*. 2012. Disponível em: <http://publib.boulder.ibm.com/infocenter/wvraix/v6r1m0/index.jsp?topic=%2Fcom.ibm.wvraix.newss7.doc%2Fthess7protocolstac3.html>. Acesso em: 7 fev. 2012.

NETBEANS COMMUNITY. Visual library 2.0 - documentation. In: ____. 2011. Disponível em: <http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files /documentation.html#Introduction>. Acesso em: 27 dez. 2011.

OMNET++ COMMUNITY. What is omnet? In: ____. 2012. Disponível em: <www.omnetpp.org/home/what-is-omnet>. Acesso em: 7 fev. 2012.

RUSSEL, T. *Signaling System #7*. [S.l.]: McGraw-Hill Communications, New York, 2006.

SOURCEFORGE. Overview. In: ____. *The DESMO-J Tutorial*. 2012. Disponível em: <http://desmoj.sourceforge.net/tutorial/overview/0.html>. Acesso em: 7 fev. 2012.

UC BERKELEY. The ptolemy ii project. In: ____. 2012. Disponível em: <ptolemy.eecs.berkeley.edu/ptolemyII/summary.htm>. Acesso em: 7 fev. 2012.