FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Implementação de um parser HL7 para integração do ALERT® com aplicações terceiras

Bruno Ricardo da Silva Pereira

Relatório de Projecto realizado no âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Orientador: António Miguel Pontes Pimenta Monteiro (Prof.)

Julho de 2008

Implementação de um parser HL7 para integração do ALERT® com aplicações terceiras

Bruno Ricardo da Silva Pereira

Relatório de Projecto realizado no âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Jorge Manuel Gomes Barbosa (Prof.)

Arguente: José Maria Fernandes (Prof)

Vogal: António Miguel Pontes Pimenta Monteiro (Prof.)

31 de Julho de 2008

Confidencial

Resumo

Neste relatório encontra-se documentado todo o trabalho realizado pelo aluno na ALERT Life Sciences Computing, S.A. ao longo das 20 semanas de duração do projecto. O aluno esteve inserido na equipa de desenvolvimento do INTER-ALERT®, que é o *software* utilizado para fazer a troca de informação entre o *software* clínico ALERT® e aplicações terceiras, interface com outras aplicações.

O projecto corresponde a desenvolver uma aplicação que permita a validação e interpretação de mensagens HL7 versão 2. Mensagens HL7 são mensagens que têm uma estrutura que segue o *standard* de mensagens HL7, Health Level Seven[HLS].

O projecto foi desenvolvido exclusivamente em Java e com duas camadas bem distintas, acesso a dados e lógica da aplicação. O intuito do projecto é ser integrado em outras ferramentas fornecendo um conjunto de métodos que permita a fácil validação e interpretação de mensagens HL7.

Como a estrutura de uma mensagem HL7 não é algo totalmente fixo, existe liberdade durante a criação de mensagens, e criar uma estrutura que permita guardar a informação de uma mensagem de forma estruturada sem perder informação e que respeite o *standard* é um processo complexo. Da mesma forma a tarefa de validação é um processo que tem de ser tolerante, tal como indica o *standard*, mas sem perder informação relevante.

Neste relatório são também documentados os testes realizados, os valores dos testes de performance atingidos e uma análise crítica desses mesmos valores.

Para além da aplicação principal, foi ainda pedido ao aluno o desenvolvimento de outras aplicações que, utilizando o projecto proposto, aumentam as funcionalidades pedidas e facilitam o trabalho com mensagens HL7. A documentação destas funcionalidades também se encontra neste relatório e inclui a arquitectura e a integração com o projecto inicial.

Uma das ferramentas desenvolvidas que utiliza a aplicação de validação e interpretação de mensagens é um serviço que aceita comunicações TCP/IP, e consoante a mensagem HL7 recebida valida-a e envia a mensagem de ACK, contendo informação de acordo com o *standard*, como resposta. Parte dessa informação indica se a mensagem é válida, sendo, no caso contrário, acrescentada uma descrição do erro ocorrido.

De notar que todo o trabalho desenvolvido será incluído na próxima versão do INTER-ALERT® e será colocado em funcionamento em várias instituições, não só em Portugal como em todo o Mundo.

Abstract

In this report it is documented the project work accomplished by the student in ALERT Life Sciences Computing, S.A. throughout the 20 weeks of duration of the project. The student was inserted in the development team of the INTER-ALERT®, which is the software used to make the exchange of information between clinical software ALERT® and third party applications, interface with other applications.

The Project corresponds to develop an application that allows the validation and interpretation of HL7 messages version 2. HL7 messages are messages that have a determined structure that follows the standard of HL7 messages, Health Level Seven[HLS].

The project was developed exclusively in Java and with two well distinct layers, data access and application logic. The intention of the project is to be integrated in other tools supplying a set of methods that allows the easy validation and interpretation of HL7 messages.

Because the structure of a HL7 message isn't something totally fixed, there is freedom during the creation of messages, and creating a structure that allows the keeping of the message information in a structuralized way without losing information and that respects the standard, is one complex process. In the same way the validation task is a process that has to be tolerant, as the standard indicates, but without losing relevant information.

In this report are also documented the accomplished tests, the values of the reached performance tests and a critical analysis of the same values.

Besides the requested application, was also required to the student the development of others applications that, using proposed project, increase the required functionalities and facilitate the work with HL7 messages. The documentation of these functionalities is also included is this report, it includes the architecture and the integration with the initial project.

One of the developed tools that uses the application of validation and messages interpretation, is a service that accepts TCP/IP communications, and consonant the HL7 message received, validates it and sends the ACK message, containing information in accordance with the standard as a reply. Part of this information indicates if the message is valid, in case it is not valid a description of the occurred error is added.

Take in mind that all the developed project work will be included in the next version of the INTER-ALERT® and will be put in functioning in various institutions, not only in Portugal but also in the rest of the world.

Agradecimentos

Estas últimas 20 semanas foram sem dúvida diferentes, proporcionaram uma experiência bastante enriquecedora que me permitiu crescer. Foram semanas sem dúvida diferentes passadas em ambiente empresarial, com o rigor que tal exige, mas sempre com um óptimo ambiente proporcionado por todas as pessoas da empresa.

Este amadurecimento não se deve apenas ao trabalho e estudo durante estas semanas, mas também aos 5 anos passados na FEUP e claro, à companhia e apoio de certas pessoas e amigos que sempre me acompanharam.

À ALERT Life Sciences Computing, S.A. e ao MIEIC por me proporcionarem esta experiência.

Ao Eng. Paulo Almeida, Eng. Gil Fernandes e Eng. Carlos Costa pelo apoio prestado.

Ao Prof. Miguel Pimenta Monteiro, não só pela orientação e supervisão prestada durante este projecto, mas também pela aprendizagem durante o curso.

Aos companheiros de trabalho e de curso David Marquês, Gonçalo Almeida, Luís Maia, Diogo Coelho, Fábio Oliveira e Nelson Oliveira pelo companheirismo prestado.

Às equipas de desenvolvimento do INTER-ALERT® e de interfaces que tornaram a adaptação e integração na empresa mais fácil.

A todos os amigos, pelos bons momentos passados e pelas horas de descanso mental que esses momentos proporcionam.

Aos meus pais, José Carlos Pereira e Maria Isabel da Silva Ferreira, que sempre me ajudaram e apoiaram.

O Autor

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ACK	Acknowledgment Code
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
DOM	Document Object Model
GUI	Graphical User Interface
HL7	Health Level Seven
HTML	HyperText Markup Language
JVM	Java Virtual Machine
LLP	Lower Layer Protocol
SAX	Simple API for XML
SWT	Standard Widget Toolkit
WEB	World Wide Web
XML	EXtensible Markup Language

Capítulo 1

Introdução

1.1 ALERT Life Sciences Computing, S.A.

A ALERT Life Sciences Computing, S.A., é a empresa mãe do Grupo de Empresas ALERT. Está inteiramente dedicada ao desenvolvimento, distribuição e implementação do *software* clínico ALERT® com a missão de criar ambientes clínicos sem papel.

Com sede em Gaia, a empresa iniciou a sua actividade em Dezembro de 1999. Conta hoje com uma equipa multidisciplinar de mais de 500 colaboradores permanentes, incluindo clínicos, designers, arquitectos, engenheiros, matemáticos e gestores.

A ALERT Life Sciences Computing, S.A. é uma empresa de capital inteiramente privado.

Com um volume de facturação em 2007 de 23,16 milhões de Euros, a ALERT tem vindo a duplicar os seus resultados anualmente. Para 2008, a empresa prevê que grande parte da sua facturação provenha do estrangeiro. Estes resultados decorrem do elevado investimento da empresa no desenvolvimento e comercialização de produtos de qualidade.

O ALERT® foi já adoptado em Portugal, Espanha, Itália, Holanda, Estados Unidos, Brasil e Malásia.

Desde 2 de Outubro de 2007, altura em que a ALERT Life Sciences Computing S.A. adquiriu a totalidade da participação na myPartner Healthcare Software Solutions, foi criada uma nova unidade de negócios (HBS - Healthcare Business Solutions) que se dedica exclusivamente ao desenvolvimento de soluções ERP (Gestão de Recursos Empresariais/Hospitalares) e CRM (Gestão do Relacionamento entre o Hospital e o Cidadão/Entidade), para dar resposta às necessidades do mercado da saúde com soluções qualificadas de gestão.

As soluções de gestão desenvolvidas por esta nova unidade da ALERT assentam sobre a plataforma tecnológica Microsoft Dynamics e têm capacidade de se integrar com

qualquer outro *software* que utilize normas internacionais de interoperabilidade. As áreas cobertas são a gestão financeira, gestão de recursos materiais, gestão logística e da farmácia hospitalar, gestão da facturação de serviços de saúde, gestão de recursos humanos, gestão do relacionamento com o cidadão e entidades.

A ALERT Life Sciences Computing, S.A. integra a Rede de PMEs Inovadoras da COTEC e a sua metodologia de formação foi seleccionada pelo Fundo Social Europeu para integrar duas publicações que ilustram casos de sucesso.

A empresa é Platinum Corporate Member da HIMSS (Healthcare Information and Management Systems Society) e tem participado nas iniciativas IHE (Integrating the Healthcare Enterprise), com a obtenção pelo ALERT® de várias certificações de interoperabilidade.

1.1.1 Produtos ALERT

O ALERT® é um *software* clínico que proporciona um ambiente 100% livre de papel. É uma solução preparada para ser implementada nas mais variadas áreas operativas de um hospital, centro de saúde ou mesmo de uma clínica privada, totalmente ou por departamento.

Pensado para interligar as actividades de todos os perfis de profissionais presentes numa instituição clínica (médicos, enfermeiros, pessoal auxiliar, administrativos, técnicos de imagem e laboratório, gestores, assistentes sociais), seguindo o conceito de fluxo de trabalho entre estes, o ALERT® foi desenhado a pensar na facilidade de utilização. A interacção com o sistema é feita via ecrã táctil (touchscreen), dando bastante importância ao design da interface, baseado em ícones intuitivos, e apoiando o seu funcionamento global em conceitos de fluxo de trabalho.

ALERT® PAPER FREE HOSPITAL (ALERT® PFH), é uma solução para informatização integral de hospitais, que possibilita o registo, a interligação, a reutilização e a análise de toda a informação relacionada com a realidade clínica hospitalar.

ALERT® CLINICAL PORTAL, é uma solução integrada para qualquer hospital ou instituição de medicina privada. Permite a criação de um ambiente de trabalho integrado em que todos os sistemas de *software* se agregam.

Se um hospital quiser manter o seu sistema actual, o ALERT® CLINICAL PORTAL é capaz de criar ambientes clínicos isentos de papel, aperfeiçoando e complementado, ao mesmo tempo, a infra-estrutura de sistemas informáticos já existentes.

ALERT® DATA WAREHOUSE (ADW), faz parte das soluções ALERT® e destina-se ao arquivo e análise de informação clínica e operacional, permitindo a realização de pesquisas, análises e relatórios complexos. Através da utilização da informação contida

no ALERT® DATA WAREHOUSE, o utilizador é capaz de detectar tendências e identificar padrões, tornando-o um suporte valioso para a interpretação exacta dos eventos que decorrem no interior de um ambiente clínico.

ALERT® EMERGENCY DEPARTMENT INFORMATION SYSTEM (ALERT® EDIS), é uma solução completa para Serviços de Urgência Hospitalares. Possibilita o registo, a interligação, a reutilização e a análise de toda a informação relacionada com cada episódio de urgência.

ALERT® ELECTRONIC HEALTH RECORD (ALERT® EHR), regista, arquiva e inter-relaciona a informação clínica de cada paciente, incluindo a de outras aplicações e entidades, integrando a história clínica de cada paciente.

ALERT® INPATIENT, é um *software* clínico para Serviços de Internamento Hospitalar. Permite documentar, interligar e reutilizar toda a documentação relacionada com cada episódio de internamento. Em conjunção com o ALERT® ELECTRONIC HEALTH RECORD possibilita, em tempo real, o registo e consulta do processo clínico dos utentes e o planeamento e a análise de toda a actividade clínica do serviço de internamento.

ALERT® OUTPATIENT, é um *software* clínico para Serviços de Consulta Externa hospitalar. Em conjunto com o ALERT® ELECTRONIC HEALTH RECORD possibilita, em tempo real, o registo, consulta e análise dos dados clínicos, dentro e fora dos episódios da consulta.

ALERT® OPERATING ROOM INFORMATION SYSTEM (ALERT® ORIS), é um *software* clínico e de gestão para Bloco Operatório. Contemplando todos os intervenientes no processo e garantindo a comunicação efectiva com todas as áreas e serviços relacionados com o bloco operatório, o ALERT® ORIS constitui uma ferramenta imprescindível para o controlo de todo o processo cirúrgico. Em conjunção com o ALERT® ELECTRONIC HEALTH RECORD, possibilita, em tempo real, o registo, consulta e análise dos dados de todo o processo cirúrgico.

ALERT® P1, é um *software* de referenciação para consultas de especialidade. É um sistema informático integrado (Centros de Saúde / Hospitais / ARS / Ministério da Saúde) que permite a triagem e gestão de pedidos de primeiras consultas de especialidade.

ALERT® PRIMARY CARE, é uma solução de *software* destinada à informatização de Instituições de Cuidados de Saúde Primários. Permite o registo e consulta de toda a informação clínica de cada utente, incluindo a recolhida noutras instituições, através da interligação dos Centros de Saúde a qualquer ambiente de prestação de cuidados de saúde.

ALERT® PRIVATE PRACTICE, é um *software* destinado à informatização de clínicas e consultórios médicos. Com o ALERT® PRIVATE PRACTICE é possível registar e consultar a ficha individual de cada doente, incluindo informação documentada noutras instituições.

1.2 Metodologia utilizada na equipa

Nas equipas de desenvolvimento é utilizada uma metodologia ágil de desenvolvimento de *software*, Scrum, processo de desenvolvimento de *software* iterativo e incremental.

O Scrum caracteriza-se pelo envolvimento de 3 actores:

- Product Owner, dono do produto, é a pessoa que representa o cliente. Responsável por garantir que a equipa tem consciência do que é pretendido com o projecto, e que o desenvolve de acordo com as especificações e requisitos. É ainda responsável pela lista de funcionalidades a implementar e a prioridade a atribuir a cada uma;
- Scrum Master, orienta a equipa de desenvolvimento, é responsável por eliminar qualquer impedimento que exista. Trabalha constantemente para garantir que a equipa tem as melhores condições para atingir os seus objectivos;
- Team, equipa responsável pela entrega do produto, geralmente são equipas pequenas entre 5 a 9 pessoas com diferentes características;

Algumas noções a reter sobre o Scrum:

- Product BackLog (Lista de Funcionalidades), é um documento que contém a listagem de todas as funcionalidades a desenvolver e a sua prioridade;
- Sprint Backlog (Lista de Tarefas), é um conjunto de tarefas extraídas a partir da lista de funcionalidades com uma descrição de como as desenvolver e quais as micro tarefas que as compõem. As tarefas estipuladas neste documento são as tarefas a desenvolver durante um sprint;
- Sprint, é o período de tempo dedicado ao desenvolvimento de um conjunto de tarefas. O Sprint é repetido até todas as funcionalidades estarem desenvolvidas, o período de tempo estipulado para cada sprint é geralmente fixo, sem ultrapassar os 30 dias, e a data de fim de um sprint nunca é alterada. As tarefas que se encontrem incompletas são desenvolvidas no sprint seguinte;
- Sprint Review, reunião no final de cada Sprint com o intuito de mostrar as funcionalidades desenvolvidas neste Sprint, e ainda analisar o Sprint de forma a tentar melhorá-lo;
- Daily Scrum, reunião diária, esta reunião é realizada a cada 24 horas. Caracteriza-se por ser uma reunião muito breve onde são colocadas questões simples a cada elemento da equipa. Algumas dessas questões são o que desenvolveu desde a última reunião, o que planeia ter pronto para a próxima e se tem algum problema que o impeça de cumprir as tarefas estipuladas;

- Burn Down chart, diagrama de esforço, é um diagrama onde é exibido o esforço restante para o sprint corrente;

Conhecendo os intervenientes e as etapas fundamentais desta metodologia, são de seguida apresentadas as etapas a seguir por ordem cronológica. É de salientar que após um sprint terminar se inicia outro até o projecto estar concluído.

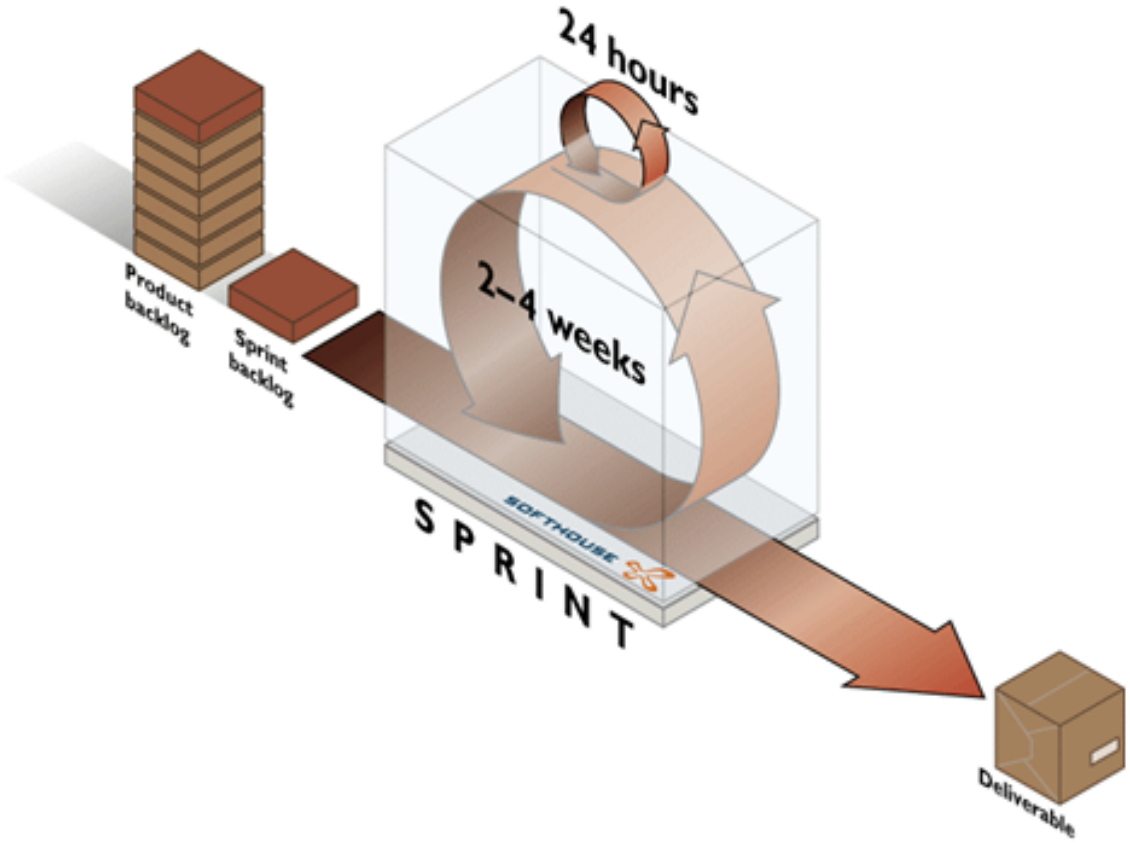

Figura 1.1: Ciclo de vida do SCRUM

- Construção do Sprint backlog, esta lista de tarefas é geralmente construída numa reunião de planeamento no início de cada sprint com a participação de todos os intervenientes;
- Início de um Sprint;
- De 24 em 24 horas é realizada a reunião diária e é actualizado o diagrama de esforço;
- Fim do Sprint, e realização da revisão do mesmo, Sprint Review;
- Início de outro sprint, e assim sucessivamente até o produto estar considerado terminado;

A figura 1.1 pretende demonstrar um ciclo de vida do SCRUM descrito, enquanto a figura 1.2 representa as várias etapas de um projecto desenvolvido segundo esta metodologia.

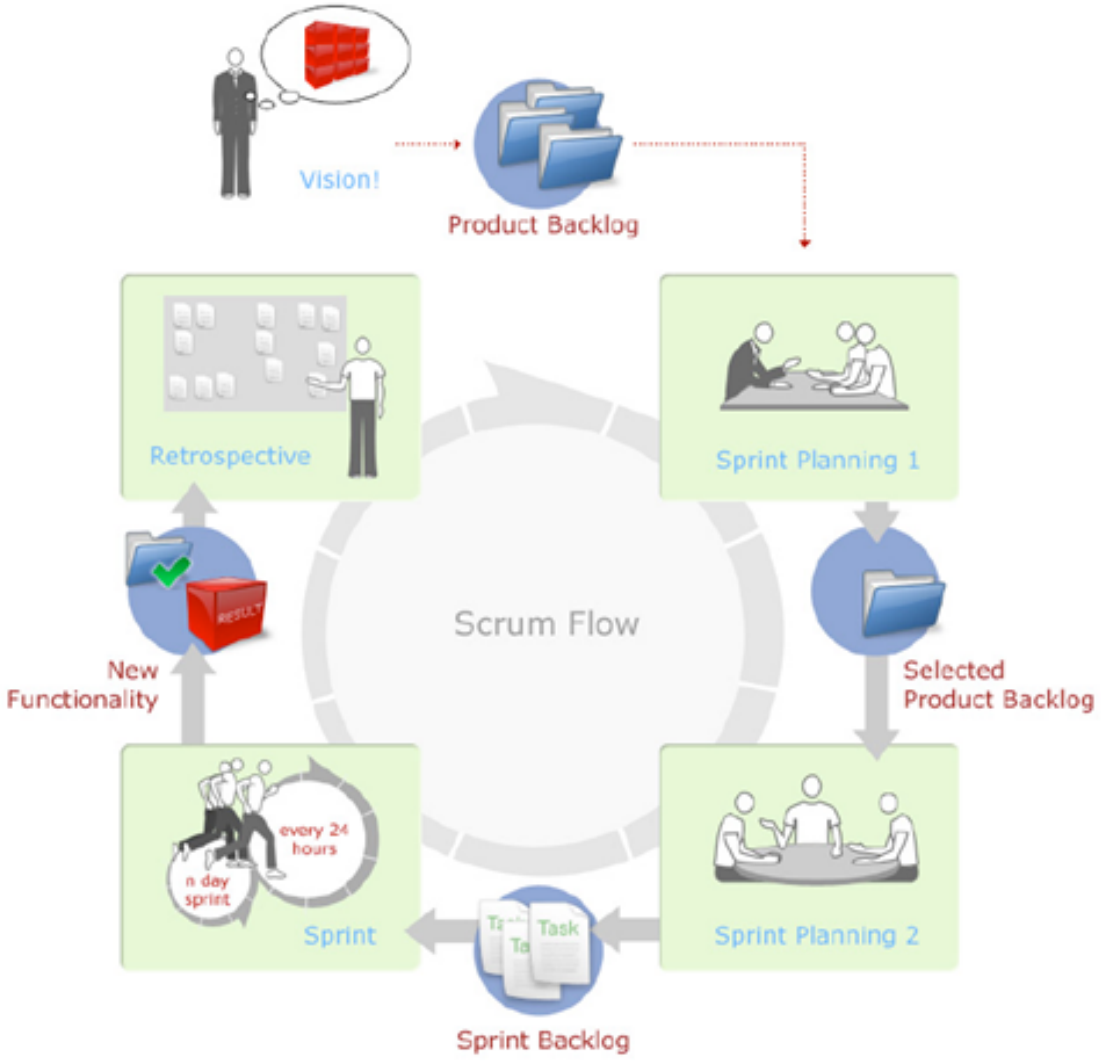

Figura 1.2: Etapas do SCRUM

1.3 Descrição do projecto

Este projecto surge devido à necessidade de criação de uma aplicação que permita receber, criar, manipular, validar e mapear a informação contida em mensagens HL7 da versão 2. Nesta versão as mensagens têm o formato ER7, um formato plano, o que obriga à existência de caracteres delimitadores especiais que permitam efectuar a separação dos vários elementos existentes numa mensagem. O *standard* HL7 é extenso e com várias

regras e definições de estruturas, o que torna a abordagem a este problema um processo complexo.

O ALERT® é um *software* de nível crítico. Assim o mal funcionamento ou pausa deste *software* pode causar incómodos e/ou atrasos que podem originar perdas de vidas humanas. Como *software* crítico que é todas as funcionalidades desenvolvidas para este *software* têm de ser robustas e eficazes, não prejudicando o funcionamento actual do sistema. Esta restrição obriga a ter um cuidado redobrado durante o desenho, planeamento e desenvolvimento do *software* para este projecto.

1.4 Organização e Temas Abordados no Presente Relatório

Anteriormente, neste relatório, foi já apresentada a instituição onde e para a qual o projecto foi realizado, a metodologia de desenvolvimento utilizada e o âmbito do projecto.

A estrutura do presente relatório segue uma sequência lógica; desta forma é constituído pelos seguintes capítulos: Estado da arte onde é feita uma breve descrição do INTER-ALERT® e do *standard* de mensagens HL7. São também analisadas algumas ferramentas existentes e as razões que levaram ao desenvolvimento deste projecto; Análise do Projecto onde é descrito o enquadramento, as possíveis abordagens identificadas, tomadas de decisão e escolha de tecnologia; Arquitectura e detalhes de implementação do projecto e de algumas ferramentas extra; Testes e Conclusões.

Capítulo 2

Estado da arte

2.1 Comunicações com aplicações terceiras

Actualmente na empresa as comunicações com aplicações terceiras são realizadas graças a uma aplicação interna denominada INTER-ALERT®, tendo esta de efectuar comunicações com outros sistemas, utilizando protocolos e formas de comunicação estipuladas entre a ALERT® e as restantes instituições. Estas formas de comunicação podem ser por TCP/IP, FTP, WebService, inserção de dados em bases de dados, por ficheiro, etc.

Para a interpretação de mensagens HL7[HLS] é utilizado um *software* proprietário, o Iguana. No entanto este *software* não suporta todas as formas de comunicação pretendidas pelo INTER-ALERT®, o que obriga à utilização de "remendos", para passar uma mensagem HL7 para o Iguana.

Estes "remendos" originam atrasos na interpretação das mensagens e introduzem mais um ponto de falha e mais um ponto a monitorizar.

2.2 Descrição geral do INTER-ALERT

O INTER-ALERT® é um produto interno da ALERT que permite a implementação de interfaces entre os vários produtos ALERT® e outras aplicações funcionando como uma camada intermédia.

O fluxo de informação entre o ALERT® e outras aplicações é feito nos dois sentidos. Para os fluxos de saída o INTER-ALERT® é responsável por obter a informação a enviar do ALERT®, estruturar essa informação consoante a comunicação com a aplicação externa e enviar essa informação. Para os fluxos de entrada o INTER-ALERT® é responsável por receber a informação, validá-la, interpretá-la e introduzi-la no ALERT®.

A figura 2.1 ilustra os fluxos de informação do INTER-ALERT® e as tecnologias utilizadas pelo ALERT®.

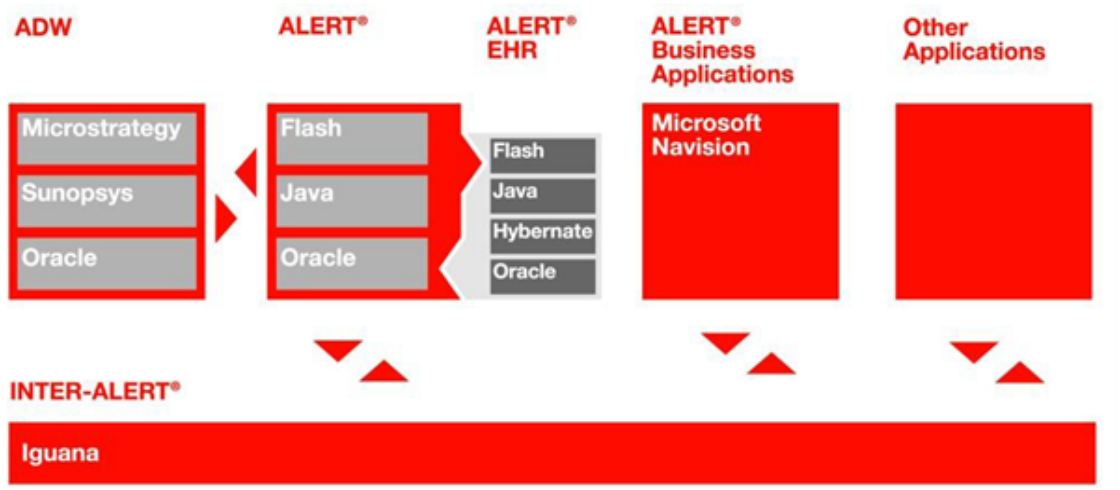

Figura 2.1: INTER-ALERT® fluxos

Quando o INTER-ALERT® é notificado pelo ALERT®, através de *triggers*, que tem de enviar informação para uma outra aplicação, é verificado para quem tem de enviar e o quê. Sabendo isto, descarrega-se da base de dados do ALERT® a informação necessária, prepara-se essa informação de acordo com a especificação acordada com a aplicação externa e envia-se essa informação, que depois de enviada gera ou não uma resposta. Essa resposta poderá ser um simples ACK ou poderá ser informação que seja necessário inserir no ALERT®. Neste último caso o processo é análogo à informação recebida de uma aplicação.

Quando o INTER-ALERT® recebe informação através de uma aplicação externa, tem de validar a informação que recebeu e enviar um ACK de resposta. De seguida interpreta essa informação, verifica onde colocar a informação no ALERT® e preenche esses campos.

Este processo pode ser mais complicado que o descrito, uma vez que alguma da informação recebida pode necessitar de desencadear alguns eventos do *workflow* do ALERT®.

2.3 Mensagens HL7

Para efeitos deste projecto apenas foram consideradas as mensagens da versão 2 do HL7[HLS], uma vez que esta é a versão utilizada para a troca de mensagens entre a instituição onde este projecto foi desenvolvido e as restantes instituições. As mensagens da versão 2, ao contrário da versão 3 onde as mensagens têm formato XML, têm formato ER7, um formato plano.

O projecto foi desenvolvido tendo em consideração as regras descritas no *standard* de mensagens HL7 V2.5, sendo compatível com as versões 2.X anteriores. Devido ao formato das mensagens ser totalmente diferente, não existe compatibilidade entre a versão 2 e a versão 3 deste *standard*.

Neste capítulo irá ser descrito de uma forma bastante superficial o formato e especificação das mensagens HL7. As noções apresentadas são apenas as consideradas fundamentais para a compreensão deste relatório. No entanto, para se conseguir entender em pormenor alguns detalhes e a complexidade envolvente deste *standard* é aconselhada a leitura do *standard* de mensagens HL7 versão 2.5[HLS].

2.3.1 Formato

Uma mensagem HL7 é composta por segmentos sendo, por sua vez, cada segmento composto por vários campos, podendo esses campos ser compostos por vários componentes e sub-componentes. Numa mensagem existem dois elementos que podem ser repetidos, consoante o tipo de mensagem HL7 a analisar: os segmentos e os campos.

A figura 2.2 pretende ilustrar qual é a estrutura de uma mensagem, não sendo feita a distinção entre os vários segmentos e campos ou a repetição destes.

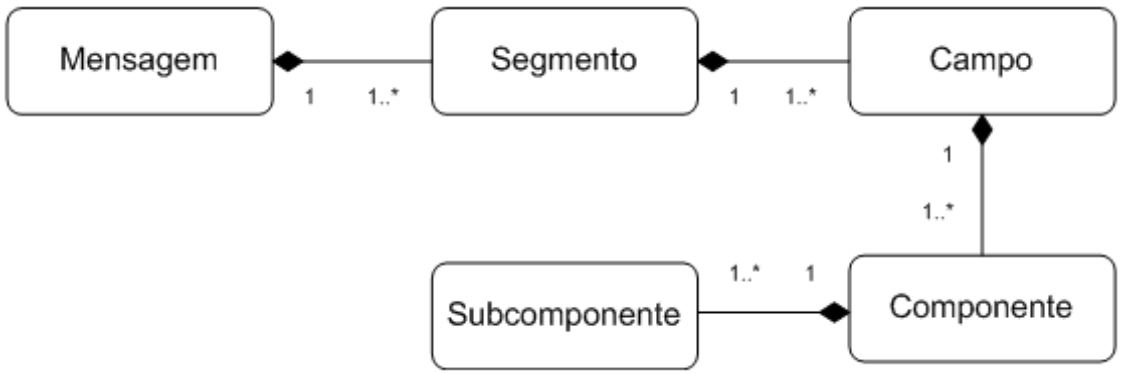

Figura 2.2: Composição das mensagens HL7

Como referido, as mensagens da versão 2 do HL7 têm um formato plano, sendo então necessário existir alguma forma de distinguir os vários elementos que a constituem. Existem desta forma alguns caracteres especiais que possibilitam essa identificação: delimitadores. Estes delimitadores vêm identificados no primeiro segmento de cada mensagem (segmento MSH) nos caracteres 4, 5, 6, 7 e 8. O único delimitador que não vem especificado, isto porque não pode ser alterado, é o delimitador de segmento. Existe um conjunto de delimitadores aconselhados que são geralmente os utilizados, e que podem ser consultados na tabela 2.1.

Tabela 2.1: Delimitadores por defeito de uma mensagem

Delimitador de	Delimitador por defeito
Segmento	<CR> (Carriage Return)
Campo	|
Componente	^
Carácter de Escape	\
Subcomponente	&
Repetição	∼

2.3.2 Especificação das mensagens

O *standard* HL7 tem definido qual é a estrutura de cada um dos seus elementos, e qual o tipo de dados que cada um desses elementos pode conter. No caso dos segmentos, para além da sua estrutura, definem-se ainda que campos constituem determinado segmento, identificando-se quais desses campos podem ser repetidos ou opcionais.

A especificação de cada tipo de mensagem é um pouco mais complexa, uma vez que para além de indicar quais os segmentos que fazem parte da mensagem e quais podem ser repetidos e opcionais, define ainda grupos que não são mais do que uma junção de N segmentos, que podem conter outros grupos. Os grupos tal como os segmentos, podem ser opcionais e/ou repetidos.

Todos os segmentos têm um identificador de 3 caracteres, que é o primeiro elemento de cada segmento. Todas as mensagens começam com um segmento especial e obrigatório, conhecido como o segmento "MSH", e que contém a informação sobre o tipo de mensagem. Na tabela 2.2 encontra-se a descrição de cada campo deste segmento.

No *standard* de mensagens HL7 encontra-se uma descrição mais detalhada sobre este segmento e sobre todos os seus campos. Tendo em consideração a tabela anterior e a descrição sobre as mensagens HL7 é apresentado um exemplo de um segmento MSH: MSH|^∼\&|XX|FEUP|YY|FEUP|200804031221||ADT^A04|123456|ID|2.5|, trata-se do segmento MSH com os delimitadores comuns, onde o 4º carácter corresponde ao delimitador de campo, o 5º ao de componente, o 6º ao de repetição, o 7º ao carácter de escape, e finalmente o 8º ao delimitador de subcomponente. O 2º campo, corresponde à aplicação de origem, neste caso XX, e assim consecutivamente. De salientar que o campo que corresponde à segurança se encontra vazio não existindo qualquer problema, uma vez que este campo é opcional.

O *standard* prevê ainda a necessidade de existirem segmentos especificamente criados por outras entidades. Como tal permite que uma mensagem possua segmentos criados por essas entidades, desde que esses segmentos surjam sempre no final da mensagem e o seu identificador se inicie pelo carácter 'Z'.

Como descrito anteriormente o primeiro segmento de cada mensagem é obrigatoriamente o segmento MSH, que indica quais são os caracteres delimitadores e qual o tipo da

Tabela 2.2: Campos que constituem o segmento MSH

Número do Campo	Tipo de dados	Opcional	Repetido	Descrição
1	ST	Não	Não	Delimitador de campo
2	ST	Não	Não	Caracteres de encode
3	HD	Sim	Não	Aplicação de origem
4	HD	Sim	Não	Instituição de origem
5	HD	Sim	Não	Aplicação de destino
6	HD	Sim	Não	Instituição de destino
7	TS	Não	Não	Data/hora
8	ST	Sim	Não	Segurança
9	MSG	Não	Não	Tipo de mensagem
10	ST	Não	Não	ID de controlo
11	PT	Não	Não	ID de processamento
12	VID	Não	Não	ID de versão
13	NM	Sim	Não	Número sequencial
14	ST	Sim	Não	Apontador de continuação
15	ID	Sim	Não	Tipo de aceitação de confirmação
16	ID	Sim	Não	Tipo de aplicação de confirmação
17	ID	Sim	Não	Código de Pais
18	ID	Sim	Sim	Código de caracteres utilizados
19	CE	Sim	Não	Língua principal da mensagem
20	ID	Sim	Não	Código de caracteres alternativos
21	EI	Sim	Sim	Identificador de perfil de mensagem

mensagem. O tipo de mensagem é constituído pelo código, o evento de trigger e a estrutura da mensagem. Existem vários códigos de mensagem. Alguns exemplos entre os mais utilizados na troca de mensagens na instituição onde o projecto foi desenvolvido são os tipos: ADT, que corresponde a mensagens de administração de pacientes; ORM, que corresponde a mensagens de pedidos (“orders”); OUL/ORU, que corresponde a mensagens de observações. O evento de trigger especifica o evento que levou ao envio da mensagem, enquanto a estrutura da mensagem indica qual é estrutura da mesma. Esta estrutura permite ao destinatário proceder à validação da mensagem. Muitas vezes a estrutura da mensagem corresponde ao código e evento da mensagem.

O campo com o tipo da mensagem é composto por 3 componentes. Supondo que é analisado um campo de uma mensagem com os delimitadores comuns, poderíamos ter: Código da Mensagemˆ Evento de triggerˆ Estrutura da mensagem. Por exemplo, poderíamos ter uma mensagem ADT, com o evento A04 com uma estrutura de mensagem ADT de evento A01, mas o mais usual será ter uma estrutura ADT_A04. Ou seja a seguinte informação é válida: ADTˆA04ˆADT_A01, assim como ADTˆA04ˆADT_A04.

2.4 Análise de aplicações existentes

Actualmente existem algumas aplicações que permitem fazer a recepção, validação e mapeamento de mensagens HL7, mas estas aplicações são proprietárias e há a necessidade de substituir estas aplicações de forma a termos o controlo total sobre todos os passos neste processo. Foi possível analisar uma destas ferramentas e submetê-la a alguns testes o que permitiu ter uma boa perspectiva do tipo de aplicação a desenvolver. No entanto esta ferramenta proprietária (Iguana) não está isenta de erros, tendo sido possível identificar alguns.

Existem também algumas aplicações OpenSource que afirmam que fazem a validação e interpretação de mensagens HL7, mas depois de analisar algumas dessas ferramentas verificou-se que esse não é o caso. Muitas destas ferramentas têm vários defeitos e algumas nem mesmo a validação de mensagens fazem correctamente.

2.4.1 Descrição das aplicações analisadas

Iguana, solução proprietária, é um motor de integração de HL7 e possibilita que sistemas de informação de saúde troquem informação utilizando o protocolo *standard* HL7. O Iguana utiliza um ficheiro de configuração designado VMD, onde são definidas as estruturas das mensagens, assim como os mapeamentos entre os elementos da mensagem e as tabelas onde irá ser armazenada a informação. Esta solução incorpora o Chameleon que permite manipular os ficheiros de configuração. O Iguana oferece ainda uma página onde podemos ver quantas mensagens processou e se foram correctamente processadas, ou se foi detectado algum erro. É ainda possível ver as mensagens e caso seja uma mensagem que não foi correctamente processada saber qual foi a razão.

Chameleon, incorporado no Iguana, é um conjunto de ferramentas para manipular mensagens HL7, permitindo fazer a sua análise e separação assim como alterar a sua estrutura e indicar onde irão os dados da mensagem ser mapeados. Todas estas informações estão contidas no ficheiro de configuração (VMD).

HAPI, solução OpenSource, é um interpretador em java para mensagens HL7 versão 2.x. Esta aplicação não mapeia os dados directamente para uma base de dados como o Iguana, mas popula objectos java, permitindo facilmente aceder a estes. Apesar de apenas terem sido analisadas estas funcionalidades, ou seja, as principais funções do HAPI, esta aplicação também oferece, tal como o Iguana um serviço que fica em execução e à escuta de mensagens. Ao contrário da aplicação anterior, não dispõe de ficheiros de configuração para especificar a estrutura válida das mensagens, o que obriga a alterar directamente o código se for necessário proceder a alterações, como por exemplo acrescentar um novo segmento válido a um tipo de mensagem.

2.5 Comparação entre aplicações existentes

2.5.1 Introdução

Nesta secção são analisadas algumas diferenças entre uma solução proprietária e uma solução OpenSource.

O Chameleon também foi considerado para a comparação que se segue, uma vez que esta ferramenta apesar de incluída no Iguana, permite fazer a análise da mensagem de uma forma simples. Obtiveram-se resultados diferentes entre este e o Iguana.

2.5.2 Análise

Uma vez que duas das ferramentas a analisar são de código fechado a análise comparativa entre as três ferramentas foi apenas feito a nível de entrada/leitura de mensagens. Apesar de o HAPI ser de código aberto e o seu funcionamento ter sido analisado com pormenor, esta análise não é considerada vital para os propósitos deste capítulo.

Na tabela 2.3 é descrito, na coluna correspondente, se a aplicação considerou a mensagem de teste válida ou inválida. A primeira coluna da tabela tem uma pequena descrição sobre a mensagem que foi enviada para as diferentes ferramentas. As cores foram utilizadas para facilitar uma rápida e fácil análise, indicando se o comportamento de uma ferramenta para uma determinada mensagem foi o esperado ou não. O verde indica que o comportamento foi o esperado e o vermelho indica que o comportamento não foi o esperado.

Tabela 2.3: Análise entre algumas ferramentas

Descrição da mensagem	Iguana	Chameleon	HAPI	Resultado esperado
Apenas o segmento MSH	Inválido	Válido	Válido	Inválido
Segmento fora de ordem	Inválido	Válido	Válido	Inválido
Segmentos opcionais fora de ordem e segmentos a mais a meio da mensagem	Inválido	Válido	Válido	Inválido
Segmentos não existentes a mais	Válido	Válido (avisa que segmento não existe)	Válido	Válido (Ignora segmentos a mais)
Segmentos não existentes a mais entre segmentos obrigatórios	Válido	Válido (avisa que segmento não existe)	Válido	Válido (Ignora segmentos a mais)
Campos obrigatórios inexistentes	Válido	Válido	Válido	Inválido
Segmento opcional fora do local	Válido	Válido (devia avisar que segmento será ignorado)	Válido	Válido (Ignora segmento a mais)
Segmento ERR opcional (ou novo segmento) fora do sítio e dentro de um grupo	Válido (ignora segmentos a mais, mas perde informação. Não insere na base de dados os segmentos do grupo posteriores ao segmento fora do sítio)	Válido (devia avisar que segmento será ignorado)	Válido (Mas todos os segmentos posteriores ao segmento fora do sítio não são criados)	Válido (Ignora segmento a mais)

2.5.3 Conclusões

O Chameleon tem algumas falhas a nível de validação, esta aplicação avisa que uma mensagem está incorrecta se surgirem segmentos a mais que não sejam previstos. No entanto não produz qualquer aviso quando se trata de segmentos a menos, considerando a mensagem válida nesta situação. Mensagens que têm os segmentos com ordem trocada também são consideradas válidas, o que não é verdade.

O Iguana apenas valida uma mensagem de acordo com os segmentos obrigatórios e a sua ordem, não produz uma validação a nível da estrutura dos campos, componentes e

sub-componentes, e considera que todos os campos são opcionais e com todos os componentes também opcionais. Considera ainda que todos os tipos de dados são uma cadeia de caracteres (string), o que não é verdade. Para além destas limitações ainda foi encontrado um erro durante o mapeamento da informação para a base de dados, quando surge um segmento estranho ou um opcional fora de ordem. O Iguana muito bem ignora esse segmento, contudo se este segmento surgir dentro de um grupo de segmentos os segmentos seguintes desse grupo não são mapeados para a base de dados, não se produzindo, no entanto, nenhum aviso ou erro retornado para o utilizador.

O HAPI executa uma validação quanto ao tipo de dados, usando uma expressão regular que valida se determinado tipo de dados é válido ou não, no entanto as restantes validações são deficientes. Uma mensagem mesmo que não possua todos os segmentos obrigatórios pode ser considerada válida, ainda mapeando incorrectamente os segmentos de um grupo que se sigam a um segmento estranho nesse mesmo grupo.

2.6 Conclusão

No secção anterior foram comparadas e testadas as aplicações considerando uma determinada mensagem específica. Verificou-se contudo, e no que toca à aplicação Iguana, existirem outras falhas que apenas se detectam quando a aplicação é colocada em funcionamento nas instituições. Estas conclusões foram facultadas pela equipa de interfaces da empresa, que é a equipa responsável pela configuração e monitorização deste *software*. Verificou-se também que este *software* é um pouco instável, devendo-se esta instabilidade a dois factores. Um deles é a instabilidade na manutenção dos canais TCP/IP, aparecendo estes frequentemente desligados dando origem à perda de mensagens. Outro é a perda de algumas mensagens. As mensagens são acusadas como recebidas, mas perdem-se, não sendo inseridas na base de dados.

Este último erro é algo semelhante a um erro detectado e descrito na secção anterior, no entanto a perda total da mensagem não foi reproduzida nos testes efectuados, apenas a perda parcial.

A necessidade deste projecto, resulta de todos estes problemas do Iguana, da instabilidade, do inconveniente do Iguana não suportar todos os meios de comunicação necessários, mais os problemas descritos na secção anterior.

Surge então a necessidade de criar uma aplicação que faça a validação e interpretação de mensagens HL7. É necessário que, independentemente do meio de comunicação acordado entre uma instituição e o ALERT®, seja bastante simples estender a aplicação para que suporte esse meio de comunicação.

Resumindo, pretende-se uma aplicação feita à medida, que seja, modular, escalável, fiável e com uma boa performance.

Capítulo 3

Análise do Projecto

3.1 Enquadramento

Face à análise realizada no capítulo anterior, e tendo chegado à conclusão da necessidade de desenvolver uma nova aplicação para a validação e interpretação de mensagens HL7, que virá substituir o Iguana, este projecto torna-se uma peça fundamental numa próxima implementação do INTER-ALERT®.

Sendo uma peça do INTER-ALERT® a aplicação desenvolvida no âmbito deste projecto será usada não só de norte a sul de Portugal, como também a nível internacional. Este facto aliado ao facto do ALERT® ser um *software* de nível crítico torna este projecto bastante motivador.

3.2 Possíveis abordagens

Durante a análise deste projecto surgiram 3 possíveis abordagens, cada uma com vantagens e desvantagens. Uma dessas hipóteses seria tirar partido da ferramenta Open-Source analisada, HAPI, e complementá-la. Nesta situação era necessário melhorar o sistema de validação de mensagens. Esta hipótese parecia a mais viável até surgirem alguns problemas: um deles foi a existência de um erro quando surge um segmento que não é esperado dentro de um grupo, que faz com que os segmentos seguintes não sejam mapeados; outro problema é o facto das definições das mensagens estarem implementadas em código, o que obriga a compilar o projecto sempre que se alterasse alguma definição.

As outras duas abordagens possíveis eram a criação de raiz da aplicação. Uma delas utilizando regras gramaticais e utilizando ferramentas de análise léxica e sintáctica para fazer a validação e a interpretação da mensagem, outra fazendo a validação e interpretação "manualmente", de raiz, recorrendo simplesmente aos delimitadores e regras definidas

pelo *standard* de mensagens HL7. Destas duas opções foi adoptada a segunda, devido a alguns factores que passo a enumerar:

1. Necessidade de utilizar um ficheiro de configuração onde estejam guardadas as estruturas das diversas mensagens. Este ficheiro pode sofrer alterações o que equivale a alterar a validação das mensagens;
2. Necessidade de, no futuro, construir uma aplicação que indique para cada elemento de uma mensagem onde será guardado na base de dados;
3. Apesar da estrutura das mensagens ser algo complexo de validar, o número de tokens, delimitadores existentes, é pequeno e torna a opção de desenvolver o analisador sem recorrer a ferramentas de análise léxica e sintáctica uma tarefa viável;

3.3 Etapas previstas

As etapas previstas para a execução deste projecto são as seguintes:

1. Construção de ficheiros de configuração
2. Validação de mensagens quanto a segmentos
   (a) Ignorar segmentos não esperados
   (b) Ignorar segmentos fora de ordem
   (c) Garantir que os segmentos obrigatórios existem e estão na ordem correcta
   (d) Garantir que a mensagem termina num segmento válido
3. Validação de mensagens quanto à estrutura dos segmentos
   (a) Garantir que todos os elementos obrigatórios existem
   (b) Rejeitar ou avisar se algum campo obrigatório se encontrar vazio
   (c) Validar o tipo de dados contido em cada elemento
4. Mapeamento da informação contida na mensagem nas estruturas correctas
5. Disponibilizar métodos para facilmente obter a informação mapeada
6. Disponibilizar métodos para facilmente criar e manipular mensagens
7. Testes
   (a) Testes Unitários
   (b) Alguns testes de aceitação; os testes de integração e de sistema serão levados a cabo por uma outra a equipa
   (c) Testes de Desempenho e Carga

3.4 Escolha de tecnologias

A escolha das tecnologias a utilizar foi bastante simples, após tomar a decisão sobre como abordar o problema proposto por este projecto. As diferentes abordagens consideradas, a escolha da abordagem e o porquê dessa escolha foram descritas na secção 3.2.

Tendo em conta que o desenvolvimento deste projecto irá ser criado de raiz e sem a utilização de ferramentas de análise leéxica/sintáctica; que a solução escolhida deverá ser, preferencialmente, uma linguagem orientada a objectos devido ao carácter do projecto e à interpretação e mapeamento da informação contida nas mensagens HL7; que a solução escolhida não pode ser proprietária, e não pode ser solução proprietária que a entidade onde foi desenvolvido o projecto tenha licença, a escolha recaiu sobre o Java, com a utilização de pacotes de leitura e manipulação de ficheiros XML, e de logging para registar a actividade da aplicação desenvolvida. Para fazer a validação das mensagens foi ainda desenvolvido um mecanismo de construção e manipulação de grafos orientados.

3.4.1 Java

A linguagem escolhida, para a realização do presente trabalho, foi então o Java[SUN]. Esta linguagem possui um conjunto de características que oferece um largo conjunto de vantagens aos programadores. Em primeiro lugar é suportada por diferentes plataformas, permitindo que a aplicação funcione em vários ambientes. Adicionalmente, existem muitas bibliotecas e API's para auxiliar na resolução de todo o tipo de problemas, além de existir muito código fonte disponível na Internet. O facto de constituir uma linguagem orientada a objectos também é um factor bastante importante, na medida em que facilita a implementação e a estruturação da aplicação.

3.4.2 JUnit

É uma Framework para Java que permite escrever e correr testes unitários. Esta Framework foi criado por Kent Beck e Erich Gamma e pertence à família de frameworks xUnit. Algumas das características do JUnit são:

- Asserções para testar resultados esperados
- Testes específicos para a partilha de dados comuns
- É fácil executar uma bateria de testes

3.4.3 Log4j

É uma ferramenta que permite ao programador fazer o log da aplicação que está a desenvolver. E esta ferramenta permite fazer o log de uma forma fácil e permite configurar a estrutura do ficheiro de logging ao gosto do programador, tal como descrito em [Foua].

3.4.4 XML (EXtensible Markup Language)

É um subtipo de SGML (*Standard Generalized Markup Language*) capaz de descrever diversos tipos de dados. O seu propósito principal é a partilha de informação por parte de sistemas de informação.

Algumas das suas principais características são:

- Simplicidade de legibilidade, não só para máquinas mas também para humanos
- Possibilidade de especificar e utilizar novas tags
- Criação de especificações que permitem fazer a validação de ficheiros construídos

De forma a trabalhar em Java com ficheiros XML, foram consultados algumas referências, das quais se destacam [Har01] e [VV06].

3.4.5 XSD (XML Schema Definition)

É uma linguagem de definição e validação de ficheiros XML. Um XSD[W3C00] é ele próprio escrito em XML e permite definir a estrutura de um ficheiro XML específico para um determinado fim.

3.4.6 Document Object Model

O DOM[W3C] é um *standard*, independente da plataforma e linguagem, do modelo de dados para representar HTML ou XML e formatos similares a esses.

Suporta navegação em qualquer direcção (por exemplo nós pai e nós "irmãos") e permite modificações dos mesmos. A implementação deste *standard* tem de, pelo menos, carregar em memória todo o documento lido até ao nó actual. Assim, DOM é melhor utilizado em aplicações onde o documento tem de ser acedido repetidamente ou sem uma sequência. Para uma aplicação que apenas precise de acessos sequenciais, o modelo SAX é normalmente mais rápido e utiliza menos memória.

3.4.7 SWT (Standard Widget Toolkit)

O *Standard Widget Toolkit*[Com] constitui uma framework de construção de interfaces gráficas implementada em Java. Foi desenvolvida pela IBM e tem como objectivo uma construção de interfaces mais leves, completas e ao mesmo tempo mais rápidas que o Swing. É actualmente mantida e suportada pela comunidade do Eclipse Foundation.

3.4.8 Eclipse IDE

Um IDE é um software especificamente desenvolvido para programadores, de modo a auxiliá-los na tarefa de desenvolver software. Para além do vulgar editor de código,

com funções especiais de highlight de palavras reservadas da linguagem de programação, funções de auto-complete de código, está também em voga o uso do IDE com sistemas de programação visual. Nestes, a GUI é desenhada por construções em que se arrastam e sobrepõem diferentes peças, para desenhar a aplicação, facilitando muito a tarefa de construção de interfaces gráficas.

O Eclipse[Fou01] está arquitectado de forma a permitir um desenvolvimento rápido e fácil de plugins. Estes constituem a base de que é feita a plataforma Eclipse, correndo sobre o seu núcleo. O programador de uma aplicação Rich Client no Eclipse, poderá utilizar os plugins de base, além de outros quaisquer que pretenda incluir na sua aplicação.

O uso desta tecnologia foi uma escolha quase imediata após a decisão da linguagem de programação, visto ser uma ferramenta reconhecidamente boa e com um bom suporte à construção de interfaces gráficas com SWT (ao contrário do NetBeans).

3.4.9 Struts 2

Esta *Framework open-source* é utilizada para o desenvolvimento de aplicações empresariais, Java EE, em ambiente Web. Utiliza uma arquitectura de 3 camadas *model-view-controller* (MVC), esta arquitectura permite uma clara distinção e separação entre os componentes visuais e a lógica de negócio implementada. Foi originalmente criada por Craig McClanahan e doada à Apache Foundation em Maio de 2000.

Esta *Framework* foi utilizada para desenvolver uma aplicação que permita a construção e manipulação da definição das mensagens HL7. Estas definições depois de criadas/editadas são guardadas num ficheiro XML, que pode ser armazenado na máquina pretendida.

Para compreender esta *Framework* foi consultada a documentação técnica e tutoriais provenientes de [Fouc] e os tutoriais presentes em [Ind].

3.4.10 JavaScript

O JavaScript é uma linguagem sobretudo utilizada no lado do cliente, em aplicações Web. Foi utilizado o dialecto actual do *standard* ECMAScript. Esta linguagem foi desenvolvido com o intuito de se assemelhar à linguagem Java, mas mais simples, isto para permitir que outras pessoas que não programadores consigam trabalhar com ela.

O JavaScript [Des] foi utilizado neste projecto não só para controlar do lado do cliente alguns erros, mas também para tornar o trabalho do utilizador uma experiência mais agradável.

3.4.11 Script.aculo.us

É um conjunto de bibliotecas JavaScript, que permitem a melhoria da interface com os utilizadores em sítios Web. Este conjunto de bibliotecas fornece uma API que permite o desenvolvimento de funcionalidades relacionadas com aspectos visuais de uma forma simples.

A documentação consultada desta ferramenta pode ler-se em [Fuc].

3.4.12 Ajax (Asynchronous JavaScript and XML)

Surge do relacionamento de algumas técnicas do desenvolvimento Web, permitindo a criação de aplicações Web interactivas. Caracteriza-se por permitir enviar pedidos ao servidor sem que exista a necessidade de actualizar a página, ou pelo menos toda a página, que nos encontramos a visualizar. Esta funcionalidade permite uma interacção mais amistosa de um sítio Web com o seu visitante.

3.5 Trabalho desenvolvido

Para este projecto estava previsto ser desenvolvida uma ferramenta que fizesse a validação e interpretação de mensagens HL7. Este objectivo foi atingido em pleno, uma vez que a ferramenta quando sujeita à análise efectuada às ferramentas descritas na secção 2.4 não apresenta qualquer falha e os objectivos quanto ao desempenho, escalabilidade e facilidade de alteração da definição das mensagens foram atingidos.

De realçar que foram desenvolvidas algumas ferramentas para permitirem a utilização mais eficaz e simples do motor HL7 desenvolvido. Uma das ferramentas permite construir os ficheiros de definição de mensagens de uma forma gráfica, fácil e remota. As outras ferramentas utilizam este motor para fazer a visualização de mensagens HL7 de uma forma estruturada e para permitir o envio e recepção de mensagens HL7 através de sockets TCP/IP, utilizando o protocolo descrito no *standard* HL7 como protocolo LLP. Este protocolo prevê que a aplicação que recebe uma mensagem deve validá-la e enviar um ACK com informação descritiva da validação que efectuou, mensagem válida ou inválida. E em caso de ser inválida qual o erro. A criação deste cliente e servidor seguindo o protocolo estipulado mostra a facilidade em estender o motor de HL7 desenvolvido.

As aplicações desenvolvidas foram denominadas da seguinte forma:

1. *HL7 Parser*, esta aplicação corresponde à ferramenta que surge deste projecto, é responsável por validar e fazer a interpretação de mensagens HL7.

2. *HL7 MessageBuilder*, aplicação Web que permite construir de raiz ou editar um ficheiro de definição de mensagens HL7. O formato deste ficheiro é o formato

pedido pelo *HL7 Parser* ou seja um formato XML com a estrutura validada por um *schema*.

3. *Message Browser*, esta aplicação permite visualizar uma mensagem HL7 num formato estruturado em árvore. No caso da mensagem a visualizar conter erros, estes são indicados.

4. *Cliente & Servidor LLP*, corresponde a duas aplicações, cliente e servidor. A aplicação servidora fica à escuta de uma mensagem e quando a recebe valida-a e responde para o cliente com uma mensagem ACK preenchida de acordo com a validação que efectuou. O cliente é uma aplicação que envia uma mensagem para o servidor e fica à espera de uma de resposta.

Os capítulos seguintes descrevem não só o motor HL7 desenvolvido (*HL7 Parser*), mas também todas estas ferramentas.

Capítulo 4

HL7 Parser

Esta aplicação corresponde à solução desenvolvida de acordo com o projecto previsto. Tal como era indicado como objectivo do projecto, esta aplicação permite fazer a validação de mensagens HL7 e a interpretação das mensagens válidas.

É ainda permitido que um utilizador edite ou crie uma mensagem HL7 que será posteriormente validada. Para isto são geradas, de acordo com a definição das mensagens, as classes que correspondem aos elementos referenciados. Estas classes permitem que a criação ou manipulação de uma mensagem seja simples, uma vez que os métodos gerados são *get's* e *set's* dos atributos que compõem um determinado elemento.

4.1 Casos de utilização

Este projecto prevê 2 tipos de utilizador:

- O "Alert User" é o utilizador comum, que pode criar ou manipular mensagens HL7 e pode também verificar se as operações que efectuou correspondem a uma mensagem HL7 válida ou inválida;
- O "Sender" é um utilizador externo, um utilizador ou uma aplicação que envia uma mensagem para o *HL7 Parser*. Quando este projecto recebe uma mensagem procede à sua leitura e validação. Uma mensagem após ter sido lida pode ser utilizada pelo "Alert User" para efectuar modificações;

Os casos de uso para cada um dos utilizadores encontram-se representados na figura 4.1.

A aplicação não tem uma função de leitura de mensagens, sendo o caso de uso ilustrado como "Read a message" usado para simbolizar que quem envia uma mensagem tem de invocar o *HL7 Parser* com o texto da mensagem a analisar e pedir que esta mensagem

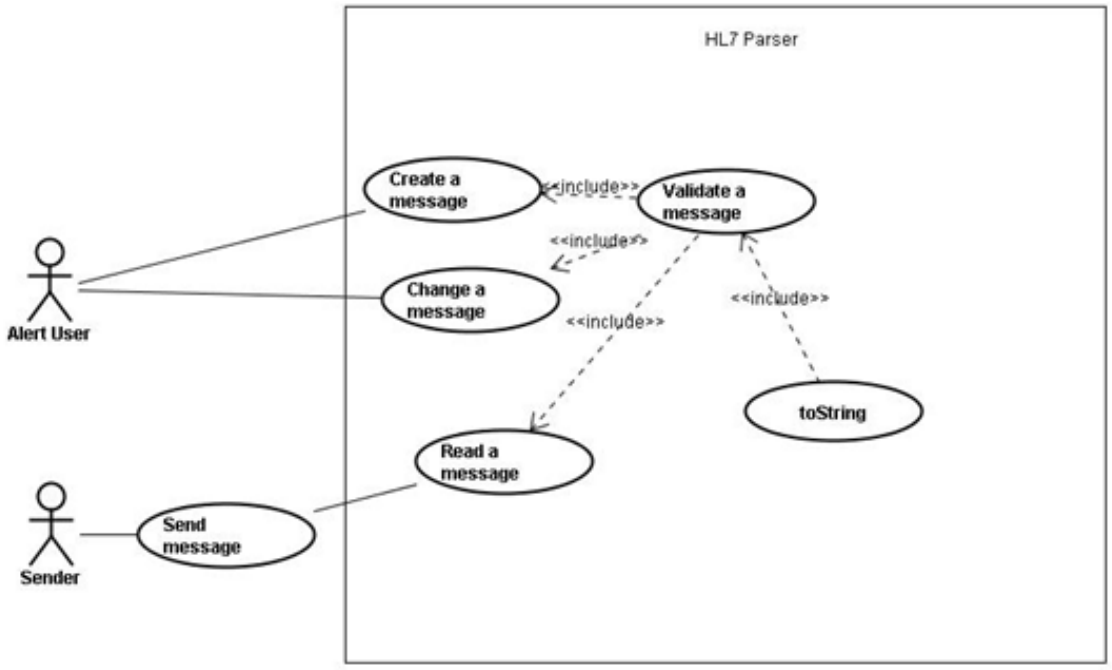

Figura 4.1: Diagrama de casos de uso

seja validada. Quando integrado com outra aplicação, como por exemplo o servidor LLP que é descrito neste relatório, esta indica que chegou uma nova mensagem, e caso seja pretendido, pede que esta seja validada e interpretada.

4.2 Ficheiros de configuração

A definição das mensagens HL7 e de todos os seus constituintes devem estar definidos em ficheiros independentes. Foi escolhido o formato XML, para poderem ser facilmente alterados conforme a necessidade da instituição na qual será implementado o projecto.

Estes ficheiros devem conter: para os tipos simples qual o seu comprimento máximo e qual a expressão regular que permite validar o seu tipo; para os tipos complexos, segmentos e mensagens qual a estrutura destes.

Exemplo da definição de um tipo simples:

```
1         <type name="ST" validation=".{0,199}" len="199" />
```

Exemplo da definição de um tipo complexo:

```
1 <complexType name="TS">
2             <component len="999" maxOccurs="1" minOccurs="0"
3                 name="Time" type="DTM"/>
4             <component len="999" maxOccurs="1" minOccurs="0"
5                 name="Degree␣of␣Precision" type="ID"/>
6 </complexType>
```

Exemplo da definição de um segmento:

```
<segmentDef name="EVN">
            <compose len="0" maxOccurs="1" minOccurs="0"
                name="Event␣Type␣Code" type="ID"/>
            <compose len="0" maxOccurs="1" minOccurs="1"
                name="Recorded␣Date/Time" type="TS"/>
            <compose len="0" maxOccurs="1" minOccurs="0"
                name="Date/Time␣Planned␣Event" type="TS"/>
            <compose len="0" maxOccurs="1" minOccurs="0"
                name="Event␣Reason␣Code" type="IS"/>
            <compose len="0" maxOccurs="unbounded" minOccurs="0"
                name="Operator␣ID" type="XCN"/>
            <compose len="0" maxOccurs="1" minOccurs="0"
                name="Event␣Occurred" type="TS"/>
            <compose len="0" maxOccurs="1" minOccurs="0"
                name="Event␣Facility" type="HD"/>
</segmentDef>
```

Exemplo da definição de uma mensagem:

```
<message name="ADT_A04">
            <segment maxOccurs="1" minOccurs="1" name="MSH"/>
            <segment maxOccurs="1" minOccurs="1" name="EVN"/>
            <segment maxOccurs="1" minOccurs="1" name="PID"/>
            <segment maxOccurs="unbounded" minOccurs="0" name="NK1"/>
            <segment maxOccurs="1" minOccurs="1" name="PV1"/>
            <segment maxOccurs="1" minOccurs="0" name="PV2"/>
            <group maxOccurs="unbounded" minOccurs="0" name="ADT_A04.INSURANCE"
                >
                <segment maxOccurs="1" minOccurs="1" name="IN1"/>
                <segment maxOccurs="1" minOccurs="0" name="IN2"/>
            </group>
</message>
```

Estes ficheiros foram inicialmente gerados a partir de *schemas* para validação de mensagens HL7 da versão 2.5 que se encontrem em formato XML. Foi desenvolvida uma aplicação que converteu a informação contida nesses *schemas* para ficheiros XML válidos segundo os *schemas* a seguir apresentados.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--
        Schema
-->
  <xsd:attributeGroup name="dataTypeGroup">
    <xsd:attribute name="name" type="xsd:string" />
    <xsd:attribute name="validation" type="xsd:string" />
  </xsd:attributeGroup>

  <xsd:element name="types">
```

```
12      <xsd:complexType>
13        <xsd:sequence>
14 <!--
15          SimpleTypes
16 -->
17          <xsd:element name="dataTypes">
18            <xsd:complexType>
19              <xsd:sequence>
20                <xsd:element name="type" maxOccurs="unbounded">
21                  <xsd:complexType>
22                    <xsd:attribute name="name" type="xsd:string" use="required"/>
23                    <xsd:attribute name="validation" type="xsd:string" use="
                          required"/>
24                  </xsd:complexType>
25                </xsd:element>
26              </xsd:sequence>
27            </xsd:complexType>
28          </xsd:element>
29 <!--
30          ComplexTypes
31 -->
32          <xsd:element name="complexTypes">
33            <xsd:complexType>
34              <xsd:sequence>
35                <xsd:element name="complexType" maxOccurs="unbounded">
36                  <xsd:complexType>
37                    <xsd:sequence>
38                      <xsd:element name="component" maxOccurs="unbounded">
39                        <xsd:complexType>
40                          <xsd:attribute name="name" type="xsd:string" use="
                                required"/>
41                          <xsd:attribute name="type" type="xsd:string" use="
                                required"/>
42                          <xsd:attribute name="len" type="xsd:integer"/>
43                          <xsd:attribute name="minOccurs" type="xsd:integer" use=
                                "required"/>
44                          <xsd:attribute name="maxOccurs" type="xsd:string" use="
                                required"/>
45                        </xsd:complexType>
46                      </xsd:element>
47                    </xsd:sequence>
48                    <xsd:attribute name="name" type="xsd:string" use="required"/>
49                  </xsd:complexType>
50                </xsd:element>
51              </xsd:sequence>
52            </xsd:complexType>
53          </xsd:element>
54        </xsd:sequence>
55      </xsd:complexType>
```

```
56    </xsd:element>
57  </xsd:schema>
58
59  <?xml version="1.0"?>
60  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
61  <!--
62    Schema
63  -->
64    <xsd:element name="config">
65      <xsd:complexType>
66        <xsd:sequence>
67  <!--
68    Segments
69  -->
70            <xsd:element name="segments">
71            <xsd:complexType>
72              <xsd:sequence minOccurs="0">
73                <xsd:element name="segmentDef" maxOccurs="unbounded">
74                  <xsd:complexType>
75                    <xsd:sequence minOccurs="1">
76                      <xsd:element name="compose" maxOccurs="unbounded">
77                        <xsd:complexType>
78                          <xsd:attribute name="name" type="xsd:string" use="
                                required"/>
79                          <xsd:attribute name="type" type="xsd:string" use="
                                required"/>
80                          <xsd:attribute name="len" type="xsd:integer"/>
81                          <xsd:attribute name="minOccurs" type="xsd:integer" use=
                                "required"/>
82                          <xsd:attribute name="maxOccurs" type="xsd:string" use="
                                required"/>
83                        </xsd:complexType>
84                      </xsd:element>
85                    </xsd:sequence>
86                    <xsd:attribute name="name" type="xsd:string" use="required"/>
87                  </xsd:complexType>
88                </xsd:element>
89              </xsd:sequence>
90            </xsd:complexType>
91          </xsd:element>
92  <!--
93    Message
94  -->
95          <xsd:element name="messages">
96            <xsd:complexType>
97              <xsd:sequence minOccurs="0">
98                <xsd:element name="message" maxOccurs="unbounded">
99                  <xsd:complexType>
100                   <xsd:sequence>
```

```
101                       <xsd:choice maxOccurs="unbounded">
102                         <xsd:element ref="segment" minOccurs="1" maxOccurs="
                                unbounded" />
103                         <xsd:element ref="group" minOccurs="0" maxOccurs="
                                unbounded"  />
104                       </xsd:choice>
105                     </xsd:sequence>
106                     <xsd:attribute name="name" type="xsd:string" use="required"/>
107                   </xsd:complexType>
108                 </xsd:element>
109               </xsd:sequence>
110             </xsd:complexType>
111           </xsd:element>
112         </xsd:sequence>
113       </xsd:complexType>
114     </xsd:element>
115
116     <xsd:element name="segment">
117       <xsd:complexType>
118         <xsd:attribute name="name" type="xsd:string" use="required"/>
119         <xsd:attribute name="minOccurs" type="xsd:integer" use="required"/>
120         <xsd:attribute name="maxOccurs" type="xsd:string" use="required"/>
121       </xsd:complexType>
122     </xsd:element>
123
124     <xsd:element name="group">
125       <xsd:complexType>
126         <xsd:sequence>
127           <xsd:choice maxOccurs="unbounded">
128             <xsd:element ref="segment" minOccurs="0" maxOccurs="unbounded" />
129             <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded" />
130           </xsd:choice>
131         </xsd:sequence>
132         <xsd:attribute name="name" type="xsd:string" use="required"/>
133         <xsd:attribute name="minOccurs" type="xsd:integer" use="required"/>
134         <xsd:attribute name="maxOccurs" type="xsd:string" use="required"/>
135       </xsd:complexType>
136     </xsd:element>
137 </xsd:schema>
```

Existem dois *schemas* porque as definições foram divididas em dois ficheiros, um com as definições das mensagens e dos segmentos, e outro com as definições dos tipos complexos e simples.

Apesar de no *standard* não se falar de tipos complexos, este nome é apropriado para referir campos e componentes. Estes na realidade são um elemento que pode ou não englobar outros: os campos podem englobar componentes e os componentes podem ser constituídos por sub-componentes. Se um campo ou componente não for constituído por vários elementos, mas sim por um tipo apenas, ou seja, não for constituído na sua

definição por vários elementos (quando representado não tem caracteres delimitadores de componente ou sub-componente), podemos dizer que é de um tipo simples. Caso contrário é um tipo complexo.

Nos exemplos de definição apresentados é possível ver que cada elemento que pretendemos definir contém o atributo *name* e um conjunto de elementos filhos. O atributo *name* indica qual o nome do elemento a definir enquanto, o conjunto de elementos filhos representam a sua estrutura. Cada elemento filho apresenta os seguintes atributos:

- name, nome do elemento que faz parte da definição;
- minOccurs, se o elemento é opcional ou obrigatório. Os valores usados geralmente são 0 – opcional; 1 – obrigatório. No futuro pode haver a necessidade de obrigar um segmento a aparecer X número de vezes;
- maxOccurs, se o elemento pode ou não ser repetido. Os valores usados geralmente são 1 – não pode ser repetido; unbounded - pode ser repetido;
- type, este atributo não é usado nas definições das mensagens, mas nas restantes indicam qual a identificação do tipo de dados esperado;

4.3 Arquitectura do Projecto

Como especificado na análise funcional este projecto não prevê o desenvolvimento de uma interface GUI para com o utilizador. Isto deve-se à natureza do projecto que prevê que este venha a ser integrado num outro projecto. Assim sendo, ao contrário das arquitecturas com 3 camadas, este projecto irá ser desenvolvido seguindo apenas 2 camadas: a camada de acesso a dados e a camada da lógica de negócio.

A figura 4.2 representa as 2 camadas da arquitectura e os principais mecanismos de cada uma dessas camadas.

A camada de acesso aos dados lê os ficheiros de configuração e guarda em memória a definição das diversas mensagens, segmentos, tipos e opções da aplicação. Esta operação está prevista que apenas ocorra quando o projecto é iniciado, contudo quando esta aplicação é incorporada em outras aplicações pode-se utilizar o código dessas aplicações para invocar o método de *load* que carrega um novo ficheiro de definição. Esta método raramente será invocado, contudo caso exista essa necessidade a funcionalidade está disponível.

A informação obtida da leitura dos ficheiros fica armazenada numa única classe que possui tabelas de *hashing* para um acesso fácil a uma determinada definição. A classe que armazena esta informação é uma classe de instanciação única (singleton).

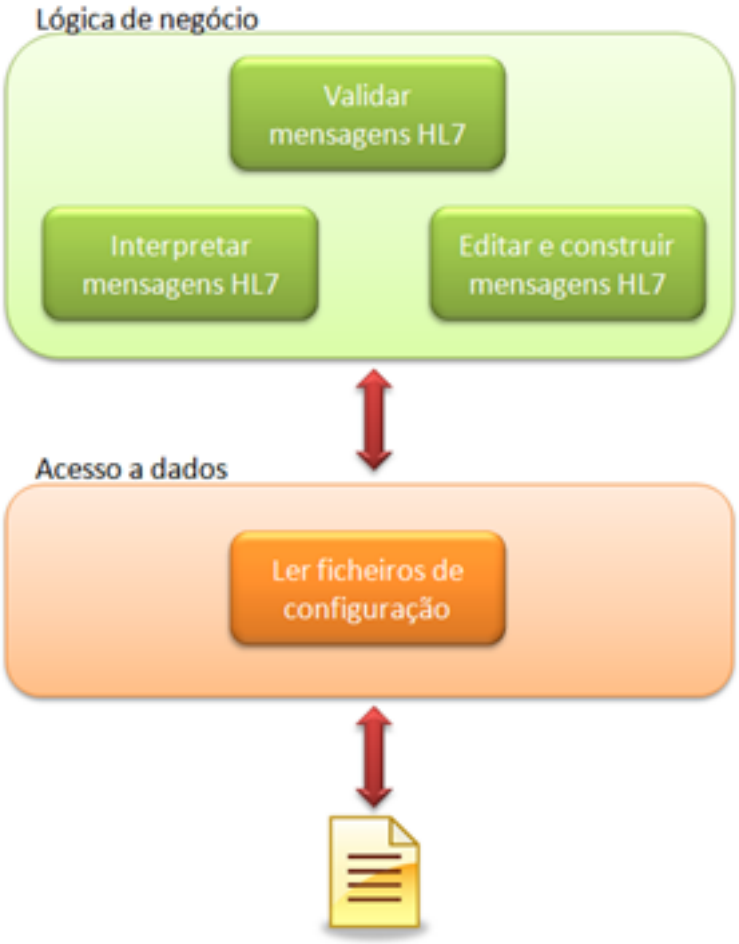

Figura 4.2: Arquitectura do projecto

Uma classe singleton é uma classe que apenas tem um único objecto. O acesso a este objecto, dentro da mesma máquina virtual corresponde sempre ao mesmo objecto, com o mesmo estado com o qual ficou quando da sua última utilização.

A camada de lógica é a responsável por todo o tratamento efectuado às mensagens HL7, que se pode dividir em três grandes tarefas:

- Interpretação de uma mensagem HL7 e dos respectivos elementos e guardar a sua informação, caso seja válida, numa estrutura em memória para poder posteriormente ser manipulada e guardada;

- Validação de uma mensagem HL7; esta validação é efectuada tendo em conta o tipo da mensagem e a definição obtida através da leitura dos ficheiros de configuração;

- Manipular e construir mensagens HL7; é possível editar mensagens HL7 que tenham sido previamente lidas e consideradas válidas, assim como construir uma

mensagem de raiz. É possível criar segmentos e grupos assim como inserir componentes e sub-componentes;

4.3.1 Diagrama de classes

O projecto encontra-se dividido em quatro pacotes, onde cada pacote agrupa um conjunto de ficheiros que são usados para a execução de determinadas funcionalidades. Esses pacotes são:

- *configuration*; neste pacote encontram-se as funcionalidades que dizem respeito à leitura dos ficheiros de configuração. Para além dessas funcionalidades, que são as mais importantes, ainda contém as funcionalidades utilizadas para converter os *schemas* do *standard* HL7 nos ficheiros de configuração usados;
- *generatedClasses*; neste pacote encontram-se as funcionalidades que dizem respeito à geração de classes em conformidade com as configurações lidas;
- *Generated*; neste pacote encontram-se as classes geradas. Estas classes servem para o utilizador editar e criar mensagens HL7 de uma forma cómoda;
- *grammar*; neste pacote estão as classes que dizem respeito à estrutura de uma mensagem e as classes para mapear a sua informação;
- *Store*; neste pacote encontram-se as classes de auxílio para mapear e validar a informação de uma mensagem;
- *parser.validate*; neste pacote encontram-se as funcionalidades de validação e interpretação de uma mensagem HL7;

As classes que irão guardar e manter a informação de cada mensagem HL7 encontram-se representadas na figura 4.3.

Esta estrutura permite representar uma mensagem sem perder qualquer informação. Como é possível observar, para além desta estrutura permitir guardar todos os elementos de uma mensagem, permite também estruturar a informação de uma mensagem de acordo com os grupos de segmentos que podem ser especificados.

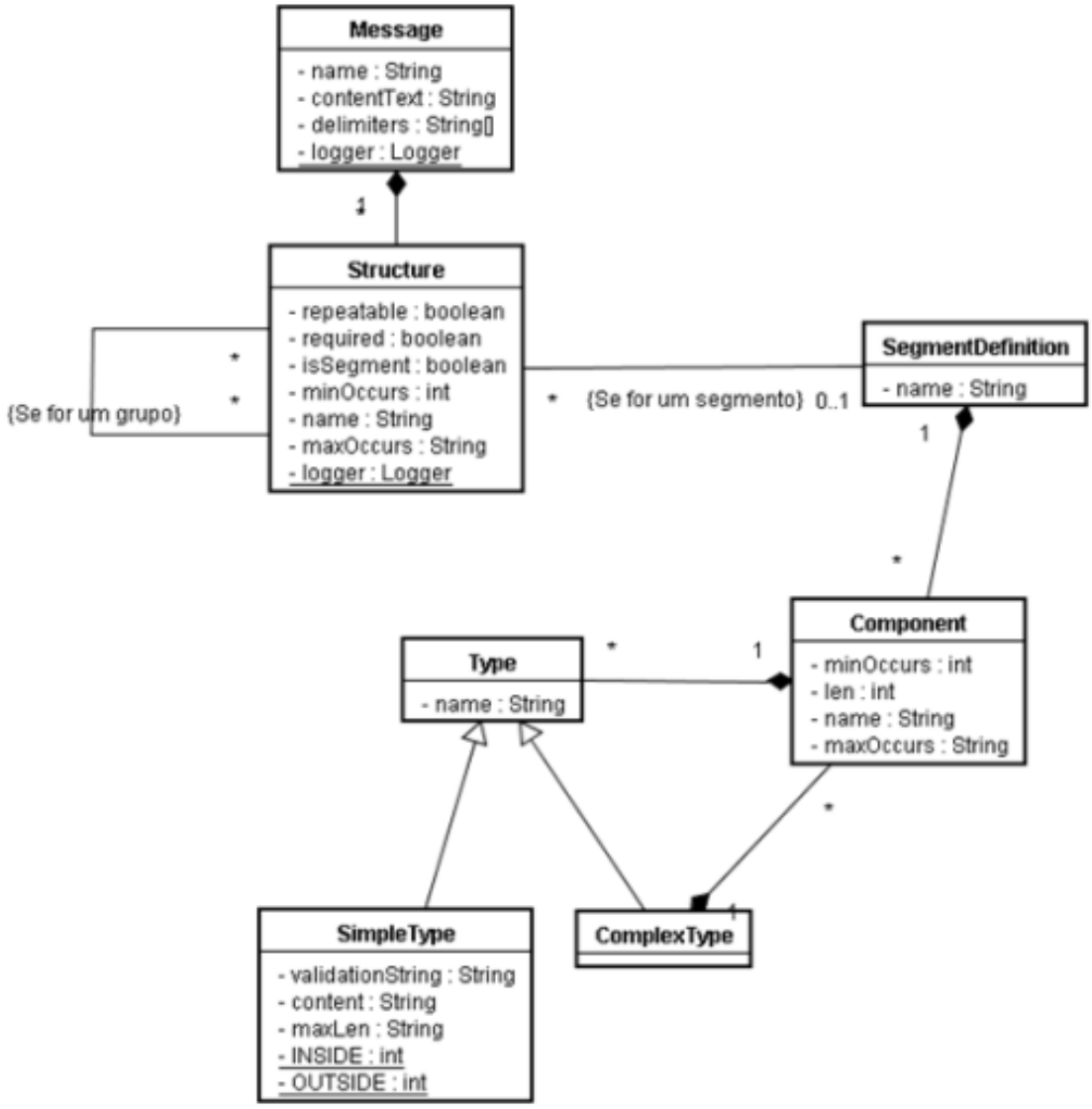

Figura 4.3: Estrutura de uma mensagem

4.4 Validação das mensagens

A validação das mensagens é efectuada em dois passos, o primeiro verifica se a mensagem é composta por um conjunto válido de segmentos, e o segundo valida a estrutura dos segmentos e seu conteúdo.

Uma descrição de qualquer erro que surja durante a validação ou alguma situação não comum, como o caso de ignorar um segmento, é guardada num ficheiro de log.

4.4.1 Primeiro passo da validação

Para se poder proceder à validação a primeira coisa a saber é o tipo da mensagem que està a ser analisada, sendo então carregada a sua definição. Esta definição indica quais os segmentos e grupos que constituem a mensagem, a sua ordem e se cada um deles é obrigatório e/ou repetido. Com esta informação é construído um grafo orientado, em que cada nó do grafo corresponde a um possível segmento ou grupo e as ligações representam quais os elementos seguintes possíveis.

Um exemplo de uma definição mapeada num grafo é apresentada de seguida. Na definição da mensagem os segmentos ou grupos de segmentos que se encontram entre parênteses rectos são opcionais e aqueles entre as chavetas podem ser repetidos.

```
MSH
EVN
PID
[{NK1}]
PV1
[PV2]
[{
IN1
[IN2]
}]
```

A definição apresentada tem um grupo composto pelos segmentos IN1 e IN2. Um grupo pode ter qualquer nome o que não influencia a validação, contudo tem que ter sempre um nome identificador para permitir a construção do grafo. Para este exemplo o grupo foi denominado de INSURANCE.

Na figura 4.4 é possível consultar o grafo gerado para a definição anterior.

De seguida encontra-se um outro exemplo, um pouco mais complexo em que o nome atribuído a cada grupo se encontra indicado na própria definição, seguido dos caracteres '- -'. O grafo gerado para esta definição encontra-se representado na figura 4.5.

```
MSH
PID
[{ NTE }]
[ PV1 ]
{                -- ORDER
  [ ORC ]
  {              -- REQUEST
```

```
    OBR
    [{ NTE }]
    [{          --RESULT
      [ OBX ]
      [{ NTE }]
    }]
  }
}
[{ ZPS }]
```

O primeiro nó de um grafo é sempre o nó MSH. Isto acontece porque todas as mensagens têm de ter obrigatoriamente como primeiro segmento o segmento MSH.

Para uma mensagem ser considerava válida tem que conter os segmentos necessários e pela ordem correcta para navegar pelo grafo e terminar com um segmento que corresponda a um nó com ligação para o nó END.

Para efectuar a validação, para além do grafo, é necessário obter também a lista dos segmentos da mensagem que pretendemos validar. Obter esta lista de segmentos é simples, uma vez que os segmentos estão obrigatoriamente separados pelo carácter delimitador <CR>. Os identificadores dos segmentos desta lista têm que ter a mesma ordem pela qual foram obtidos, ou seja, a mesma ordem que possuem na mensagem recebida.

Tendo estas duas estruturas, o grafo e a lista de segmentos, sabemos que se obtivermos o tipo de mensagem para carregar o grafo, é porque o primeiro segmento é o segmento MSH. A navegação pelo grafo inicia-se já no segmento MSH, de modo que para avançar para o nó seguinte é analisado qual o segmento seguinte da mensagem. Se se conseguir fazer a correspondência, o nome do segmento corresponde a um nó possível de visitar, e então avançamos para esse nó. Caso não exista uma ligação para um nó com o nome do segmento a analisar, esse segmento é descartado, porque não está contemplado na definição da mensagem. Por um segmento ser descartado não significa que a mensagem seja inválida, apenas é ignorado/retirado da mensagem para as etapas seguintes.

Quando as ligações de um nó apontam para um nó grupo, as ligações desse nó grupo são estendidas e consideradas como ligações directas do primeiro nó. A existência dos nós grupo devem-se a definições de mensagens que não possuem um segmento obrigatório como o primeiro elemento a surgir num grupo. Esta situação não devia ocorrer na versão 2.5, mas existem definições de mensagens que não respeitam esta regra (algumas das quais presentes na própria documentação do *standard* de mensagens HL7).

De seguida é possível ver um exemplo da validação quanto à estrutura de uma mensagem para o primeiro exemplo atrás referido. Nesta mensagem exemplo não foi preenchido

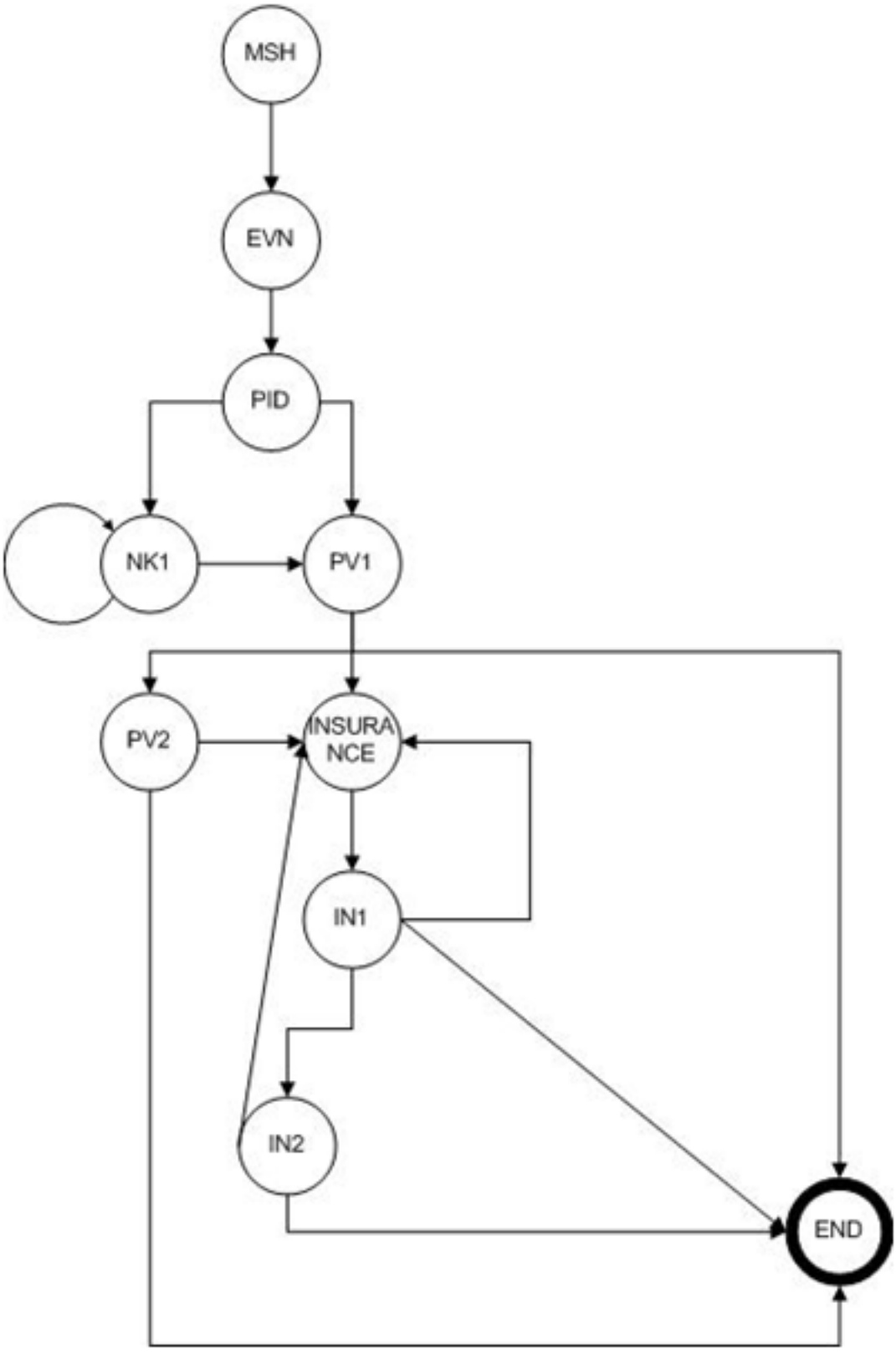

Figura 4.4: Grafo gerado para a definição de uma mensagem

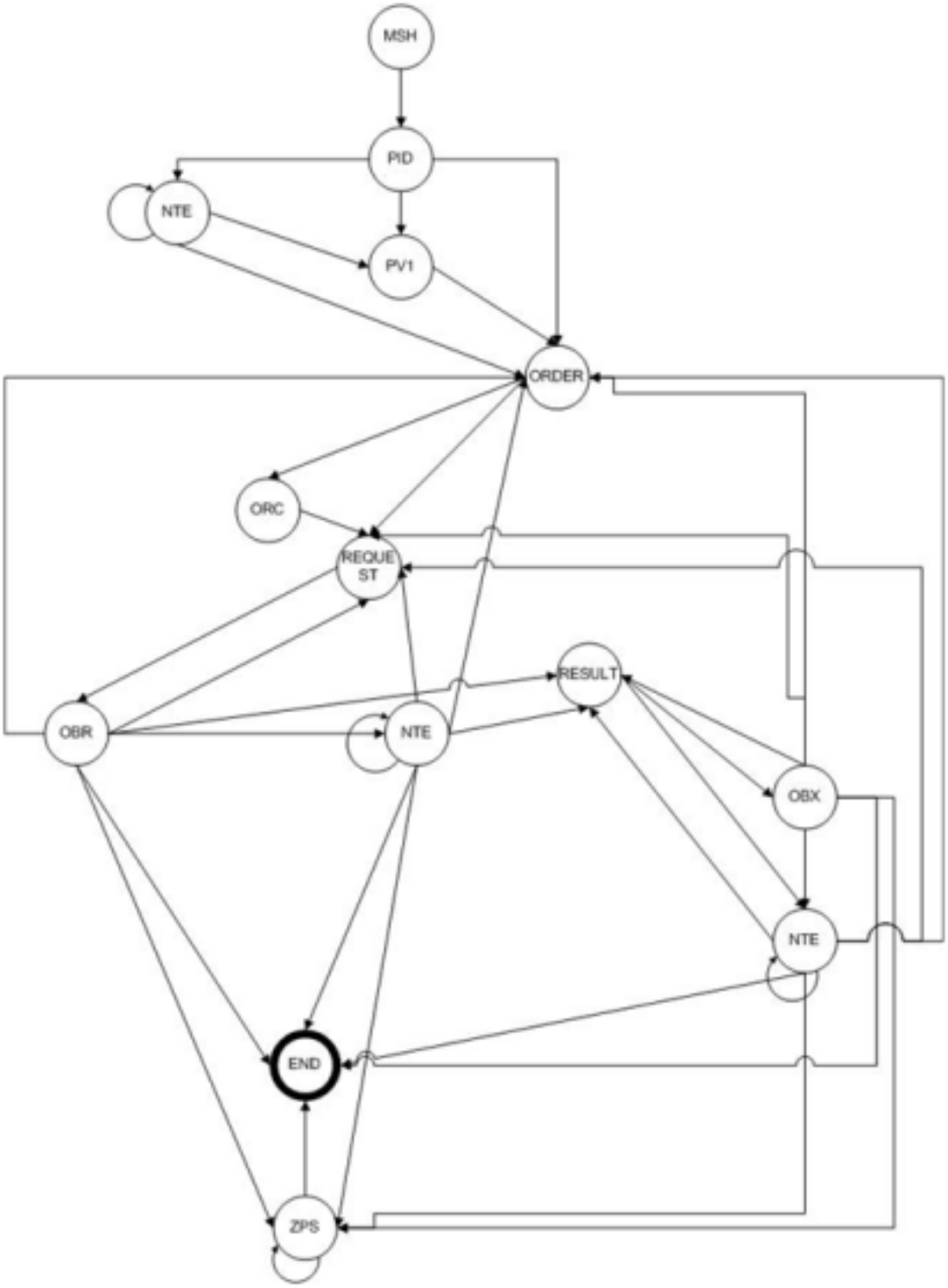

Figura 4.5: Grafo gerado para a definição de uma mensagem exemplo 2

o conteúdo dos segmentos, apenas é pretendido validar a mensagem quanto aos segmentos.

```
MSH|||||
```

```
EVN|||||
PID|||||||
NK1|||||||
ZXX|||
NK1|||||||
NK1|||||||
PV1|||
IN1|||
IN1|||
IN1|||
IN1|||
ZXX|||
IN2|||
IN1|||
IN2|||
IN1|||
IN1|||
IN2||
```

Na figura 4.6 encontra-se a representação do grafo utilizado para fazer o primeiro passo da validação e alguns apontamentos que demonstram em que nó se encontrava o grafo na altura em que se valida um determinado segmento.

O nó INSURANCE encontra-se destacado a vermelho para salientar que se trata de um nó grupo. O número, à frente de cada segmento nas ligações, indica qual o segmento da mensagem que está a ser analisado. O segmento ZXX está destacado no grafo porque é um segmento descartado.

Como é possível ver o nó em que o grafo terminou, o nó IN1 tem uma ligação para o nó END e como tal a mensagem é considerada válida segundo a estrutura e passa para o segundo passo da validação. De salientar que os segmentos descartados não são analisados no segundo passo da validação nem considerados para a interpretação da mensagem.

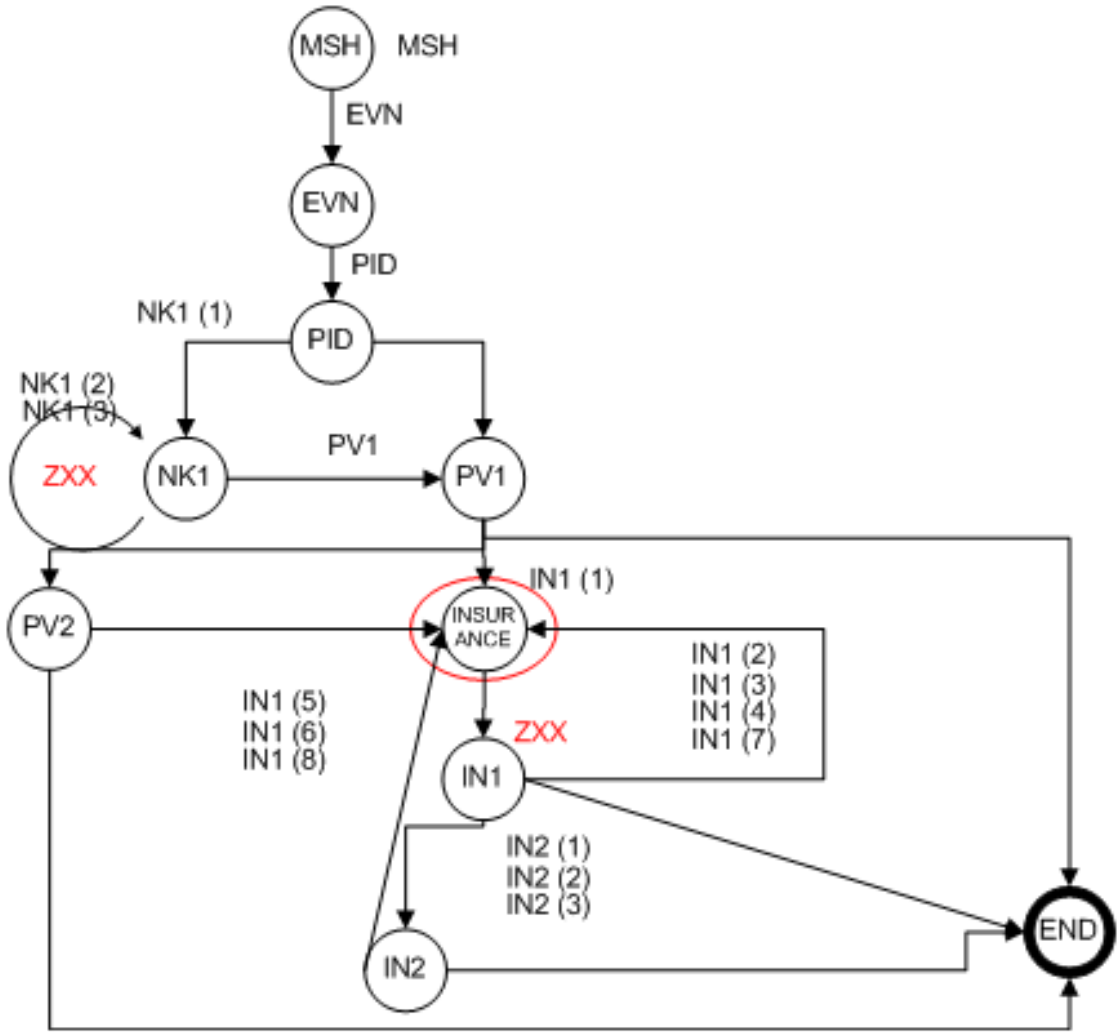

Figura 4.6: Percorrer o grafo para uma dada mensagem

A figura 4.7 resume num esquema os passos executados para a execução do primeiro passo da validação.

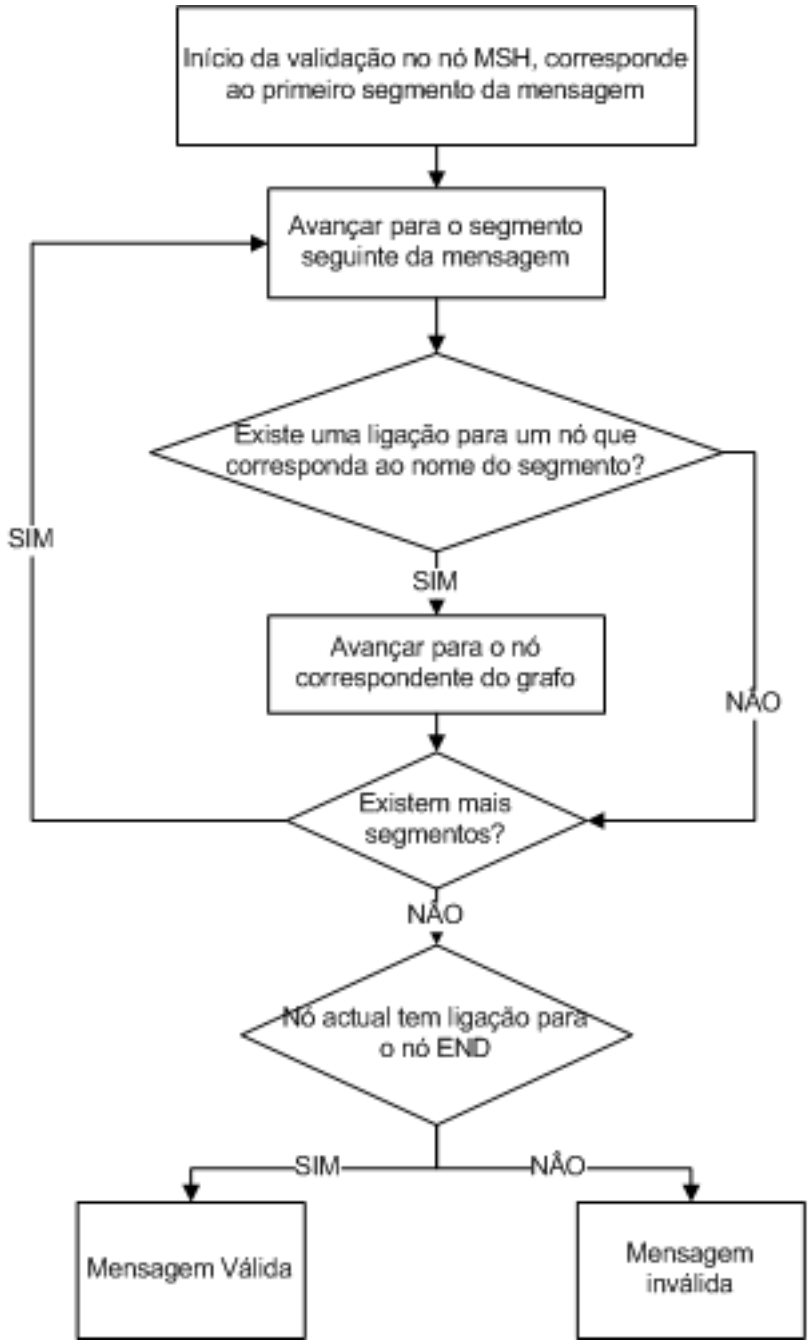

Figura 4.7: Algoritmo para validar uma mensagem segundo a estrutura

4.4.2 Segundo passo da validação

O segundo passo da validação consiste em verificar a estrutura e conteúdo de cada um dos segmentos. Para esta etapa e tendo agora a mensagem validada segundo a sua estrutura e apenas contendo os segmentos que interessam, é verificado para cada um a sua constituição quanto a campos, campos repetidos, componentes, sub-components e finalmente a informação contida em cada elemento.

Esta etapa, para além de executar a validação, é responsável por guardar a informação contida na mensagem em estruturas apropriadas para mais tarde serem acedidas para visualização, ou mesmo edição. Como tal, e uma vez que o número de segmentos e grupos não é fixo, é necessário construir a estrutura onde a informação será guardada. Esta estrutura vai sendo preenchida conforme a validação dos segmentos vai sendo efectuada.

Para se dar início a esta etapa da validação é necessário mais uma vez ter os segmentos da mensagem, sendo de seguida analisados cada um destes segmentos. A definição de cada segmento está carregada em memória e assim o processo corresponde a identificar o nome do segmento, procurar a sua definição e verificar se os campos obrigatórios se encontram presentes. Para isso é pesquisado qual o último campo obrigatório e verificado se os segmentos que estamos a analisar têm no mínimo esses campos. Se não tiverem e nas opções estiver assinalada a validação dos campos obrigatórios a mensagem será considerada inválida.

No caso de ser válida a validação continua e serão validados os campos de cada segmento. Como alguns campos podem ser repetidos, o analisador verifica se existem campos repetidos. Se existirem divide-os e valida cada um dos campos, no caso de a definição aceitar campos repetidos, caso não aceite apenas será validado o primeiro. A validação dos componentes e sub-componentes é idênticas à validação dos campos, só que não aceita repetições.

Por último é efectuada a validação quanto ao tipo de dados, que consiste em verificar se a informação respeita o tipo de dados esperado pela definição da mensagem. Esta verificação pode não ser efectuada caso esta opção se encontre desligada. No caso de a informação não respeitar o tipo de dados esperado a mensagem é considerada inválida, caso a informação seja válida é guardada na estrutura.

4.4.3 Construção da estrutura que sustenta a informação de uma mensagem

Esta estrutura tem o objectivo de conter a informação de cada mensagem. Essa informação encontra-se validada e guardada de forma a ser facilmente acedida e se necessário proceder à sua alteração. A estrutura tem obrigatoriamente de ser construída dinamicamente uma vez que os segmentos e grupos que constituem uma mensagem não são fixos. A maioria das vezes existem segmentos que podem ser repetidos, o que faz que para o mesmo tipo de mensagem podem existir estruturas diferentes, bastando para isso que uma

mensagem tenha um campo opcional que outra não tem ou, um número de segmentos a mais.

A definição da mensagem não deve ser ambígua, se for ambígua a estrutura construída pode ser diferente da esperada, no entanto válida. Uma definição é ambígua quando uma mensagem ao ser analisada sem recorrer a ferramentas, a "olho nu", não se saber se um segmento corresponde a um, ou outro ponto da definição da mensagem.

A construção desta estrutura é baseada no grafo construído para a validação e na lista de segmentos que constitui a mensagem. A estrutura é constituída por uma hierarquia que corresponde à hierarquia dos grupos do grafo. Para cada segmento é verificado, pelo grafo, se esse segmento pertence a algum grupo, caso pertença é preciso analisar se pertence a algum grupo novo, ou a um já criado anteriormente. A parte complicada desta construção encontra-se aqui, em identificar correctamente a posição em que esse segmento deve ser inserido na estrutura. Concluída a localização é inserido um segmento vazio do tipo do segmento analisado na estrutura.

Esta estrutura é preenchida quando é efectuada a validação dos tipos e esta é considerada válida.

4.5 Regras de logging

Todo o processo do *HL7 Parser* regista informação num ficheiro de log. Este ficheiro regista quais os ficheiros de definição carregados, assim como o processo de validação de uma mensagem, desde que é recebida para validação até que é considerada válida ou inválida.

Durante a validação da mensagem, caso surja alguma situação imprevista, essa informação é registada. Uma situação imprevista é um erro na mensagem recebida. Este erro pode dar origem a um aviso e a mensagem ignora o elemento que deu origem a esse erro e continua o processamento, ou dá origem a um erro considerado grave e a validação considera a mensagem inválida. Em qualquer um dos casos toda a informação é registada no ficheiro de log.

Caso os erros encontrados sejam graves a informação é registada da log como sendo um erro. No caso de se tratar de um erro não grave então é registado como um aviso.

A informação de um erro grave pode dar origem a várias entradas de erro na log, isto porque quando surge um erro num sub-componente, além da identificação desse sub-componente e a descrição do erro, é também escrito na log qual o componente e segmento a que pertence esse sub-componente. Desta forma torna-se mais simples a identificação do sub-componente que originou o erro.

É obrigatório que a primeira mensagem de erro tenha o identificador de origem de erro, sendo estes identificadores aproveitados do *standard* HL7 à excepção dos dois códigos

que foram adicionados para se ter um maior detalhe na descrição desse identificador. Os identificadores possíveis estão descritos na tabela 4.1.

Tabela 4.1: Código de identificadores de erros

Código	Descrição	Comentário
100	Erro na ordem dos segmentos	A estrutura da mensagem é inválida, não termina num segmento válido
101	Campo obrigatório inexistente	Existe um campo obrigatório que está vazio ou não se encontra na mensagem
102	Erro no tipo de dados	Um elemento possui informação em formato errado. Por exemplo um campo numérico com caracteres de texto
151	Componente obrigatório inexistente	Existe um componente obrigatório que está vazio ou não se encontra na mensagem (código adicional ao *standard* HL7)
152	Subcomponente obrigatório inexistente	Existe um subcomponente obrigatório que está vazio ou não se encontra na mensagem (código adicional ao *standard* HL7)
200	Tipo de mensagem não suportado	O tipo da mensagem não é suportado ou não se encontra na mensagem

Capítulo 5

HL7 Message Builder

O *HL7 Message Builder* tem o objectivo de facilitar a construção do ficheiro de definição de mensagens, que contém as definições das mensagens e segmentos.

Esta aplicação é uma aplicação Web e foi desenvolvida em Java utilizando a Framework Struts 2.

Esta aplicação está disponível em dois idiomas que podem ser alterados através da página inicial da aplicação, sendo estes o inglês e o português.

As operações permitidas pela aplicação são:

- Alterar idioma, alterar entre o idioma inglês e português;
- Carregar um ficheiro de configurações para a aplicação;
- Pesquisar e editar uma mensagem existente;
- Criar uma nova mensagem:
  - Com uma definição base de uma outra mensagem:
    - Definida localmente;
    - Definida num ficheiro com mensagens padrão;
- Criar um novo segmento;
- Pesquisar e editar um segmento existente;
- Adicionar ao ficheiro que estamos a manipular mensagens e segmentos do ficheiro padrão;
- Validar e efectuar o download do ficheiro de configuração que estamos a manipular;

Quando são adicionadas mensagens ou é criada uma mensagem a partir de uma definição existente no ficheiro padrão, todos os segmentos que fazem parte da definição da mensagem e que ainda não existam no ficheiro local, mas existam no ficheiro padrão, são copiados para o ficheiro local. Desta forma estes segmentos encontram-se disponíveis para o utilizador editar e/ou acrescentar à definição da mensagem. Poder-se-ia copiar todos os segmentos do ficheiro padrão para o ficheiro local, mas efectuar esta operação seria carregar o ficheiro local com muita informação que provavelmente não será utilizada. Ou seja, iria acarretar um maior consumo desnecessário de memória por parte do *HL7 Parser*.

Quando é criada uma mensagem a partir de uma definição do ficheiro padrão existe a possibilidade de algumas definições de segmento ainda não existirem localmente. Assim, quando é pedida a definição de um segmento esta é primeiro pesquisada no ficheiro local e caso não exista, no ficheiro padrão. Na eventualidade da definição não existir em nenhum destes ficheiros é dada a possibilidade de criar localmente a definição desse. Esta situação apenas ocorre se o utilizador apagar a definição de um segmento criado por ele, ou por outro utilizador.

O aspecto desta aplicação não se encontra terminado uma vez que o *design* será refeito quando a equipa de *design* terminar e aprovar os desenhos para esta ferramenta.

5.1 Casos de Utilização

Na figura 5.1 encontram-se representados os casos de uso que esta aplicação permite.

As funcionalidades de editar e criar uma mensagem possibilitam que o utilizador execute esta tarefa de uma forma muito simples. O utilizador simplesmente arrasta os segmentos colocando-os da forma pretendida. Este pode, desta forma, adicionar segmentos arrastando-os para a área de definição de mensagem, ou também arrastando pode trocar a ordem dos segmentos. É possível arrastar um grupo de segmentos ou colocar segmentos dentro ou fora de um grupo. Para visualizar a área que permite alterar as características de um segmento ou grupo, se é repetido ou opcional, efectua-se um duplo clique sobre o segmento a editar.

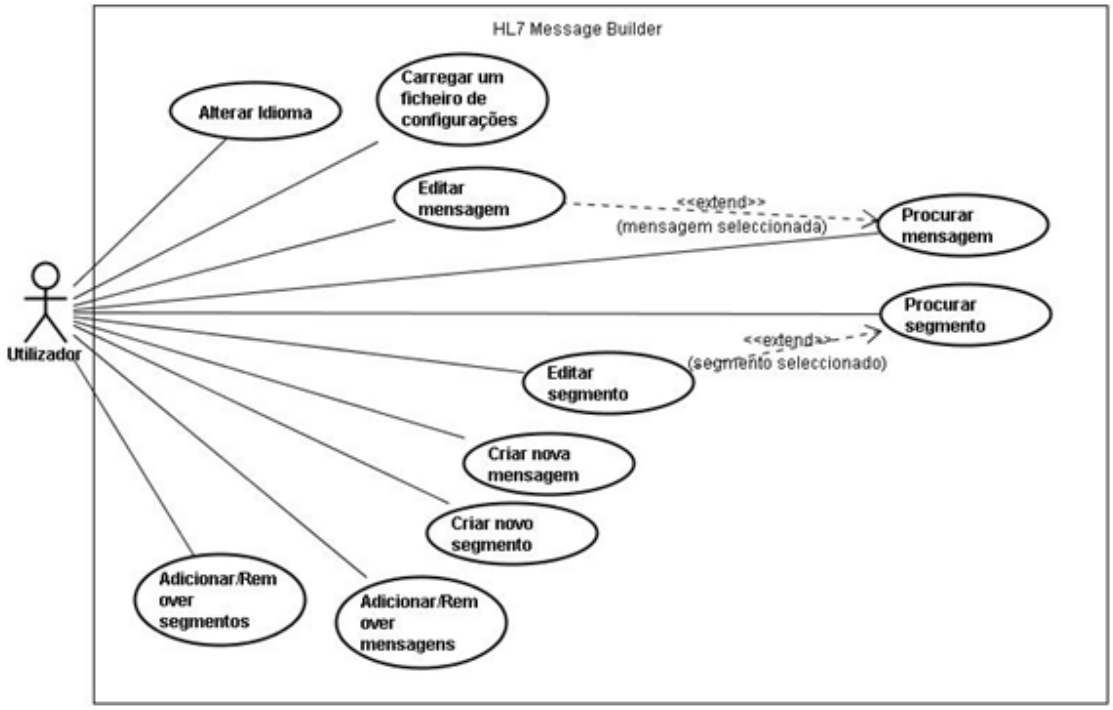

Figura 5.1: Diagrama de casos de uso

5.2 Arquitectura

Esta aplicação foi desenvolvida em ambiente Web utilizando a tecnologia Java e tirando partido de uma Framework, Struts 2, que permite o desenvolvimento seguindo uma arquitectura MVC, *Model-View-Controller*.

Este modelo de arquitectura impõe a separação entre dados, tratamento e apresentação, o que nos dá as três partes fundamentais da aplicação: Modelo (Model), Vista (View), Controlador (Controler). A figura 5.2 ilustra esta arquitectura de três camadas.

Modelo, implementa a estrutura de mais baixo nível do projecto, representa a reacção da aplicação a pedidos do cliente e o tratamento de dados.

Vista, esta camada pode ser vista de duas formas. A Vista pode representar a interface com o utilizador propriamente dita ou representar a codificação (em JSP) da mesma. A Vista limita-se apenas a gerar e fornecer ao utilizador a visualização dos tratamentos efectuados pelo Modelo.

Controlador, Implementa a camada responsável pela gestão de eventos de modo a actualizar a Vista. De facto, o Controlador não efectua qualquer tratamento, não modifica nenhum dado, apenas analisa o pedido do cliente e o redirecciona chamando o Modelo adequado. Este por sua vez deve processar o pedido e ficar a aguardar pela resposta que lhe permitirá actualizar a Vista.

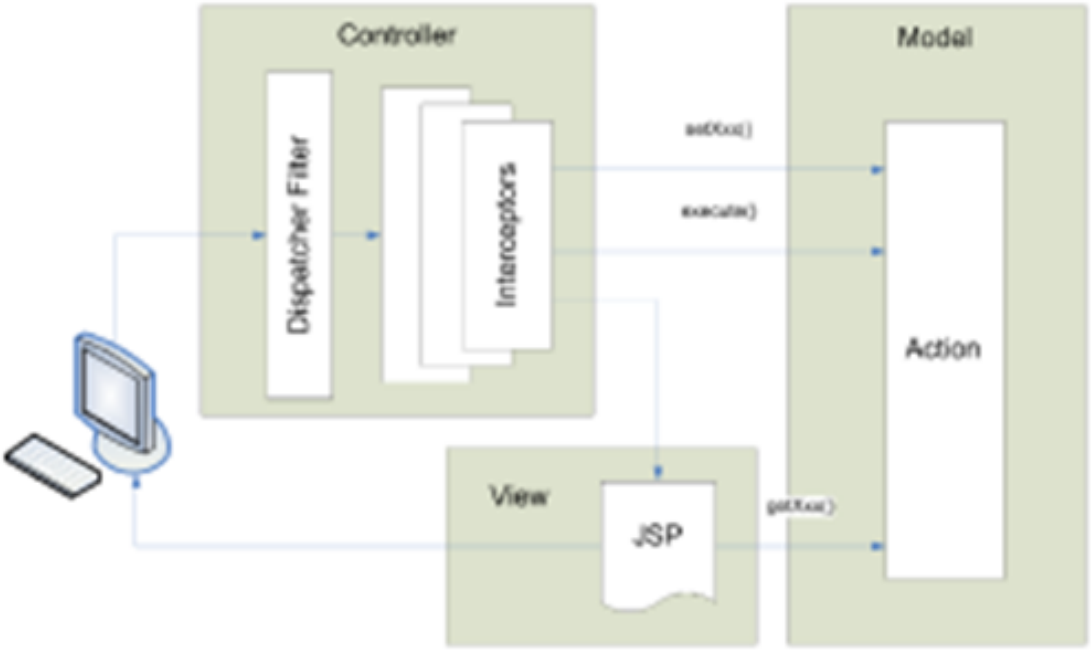

Figura 5.2: Arquitectura MVC

5.3 Estrutura de Classes

Para esta aplicação não existe propriamente um diagrama de classes, uma vez que a aplicação se resume à criação e manipulação de um ficheiro XML. As classes existentes são as *actions* criadas para a construção do modelo MVC. No entanto convém referir duas classes que foram criadas para reunir algumas funções e definições utilizadas por toda a aplicação, estas classes são *XMLUtils* e *XMLGeneralFunctions*.

As classes estão organizadas nos seguintes *Packages*:

- *messages*, agrupa as classes que dizem respeito a operações sobre mensagens;
- *segments*, agrupa as classes que dizem respeito a operações sobre segmentos;
- *xml*, agrupa as classes que dizem respeito a operações sobre ficheiros xml;
- *utils*, agrupa os ficheiros de idioma e o template geral que todas as classes que dizem respeito a uma *action* devem estender;

5.4 Estrutura dos JSP's

A estrutura dos JSP's, tal como a estrutura de classes, está organizado de uma forma lógica em que na raiz apenas se encontram os ficheiros *index.html* e *Welcome.jsp*. Os restantes ficheiros encontram-se em nos seguintes directórios:

- *Common*, onde estão os jsp's que serão invocados com frequência por outros, como o caso do cabeçalho, rodapé, etc;
- *Css*, onde está o ficheiro com o css;
- *Images*, onde estão as imagens;
- *InfoMessages*, onde estão alguns dos jsp's carregados com ajax, os que imprimem mensagens informativas;
- *JavaScript*, onde estão os ficheiros javascript;
- *Messages*, onde estão os jsp's que dizem respeito às mensagens;
- *Segments*, onde estão os jsp's que dizem respeito aos segmentos;
- *Xml*, onde estão os jsp's que dizem respeito apenas a ficheiros xml;

5.5 Aplicação

Esta ferramenta foi desenvolvida para ambiente Web e algumas imagens da interface gráfica com o utilizador são disponibilizadas nesta secção.

A particularidade de arrastar os segmentos para definir a estrutura de uma mensagem não é visível nas imagens disponibilizadas, contudo será feito um esforço para retratar da melhor forma esta particularidade.

As páginas de adicionar e remover mensagens e segmentos são muito semelhantes. Na parte esquerda da página podemos ver as mensagens ou segmentos existentes no ficheiro padrão e que podemos copiar para o ficheiro local. Do lado direito temos as mensagens ou segmentos que se encontram actualmente no ficheiro local e que podemos eliminar.

Na figura 5.5 pode-se ver um exemplo da página de adicionar e remover mensagens.

A interface que permite a manipulação da estrutura de uma mensagem é constituída por duas colunas, a parte mais à esquerda corresponde a uma listagem de segmentos que se encontram actualmente no ficheiro que está a ser manipulado. É possível pesquisar elementos nesta lista, conforme vamos escrevendo os segmentos que se encontram listados vão sendo filtrados, sendo exibidos apenas os que correspondem à pesquisa. Os

Figura 5.3: Página inicial

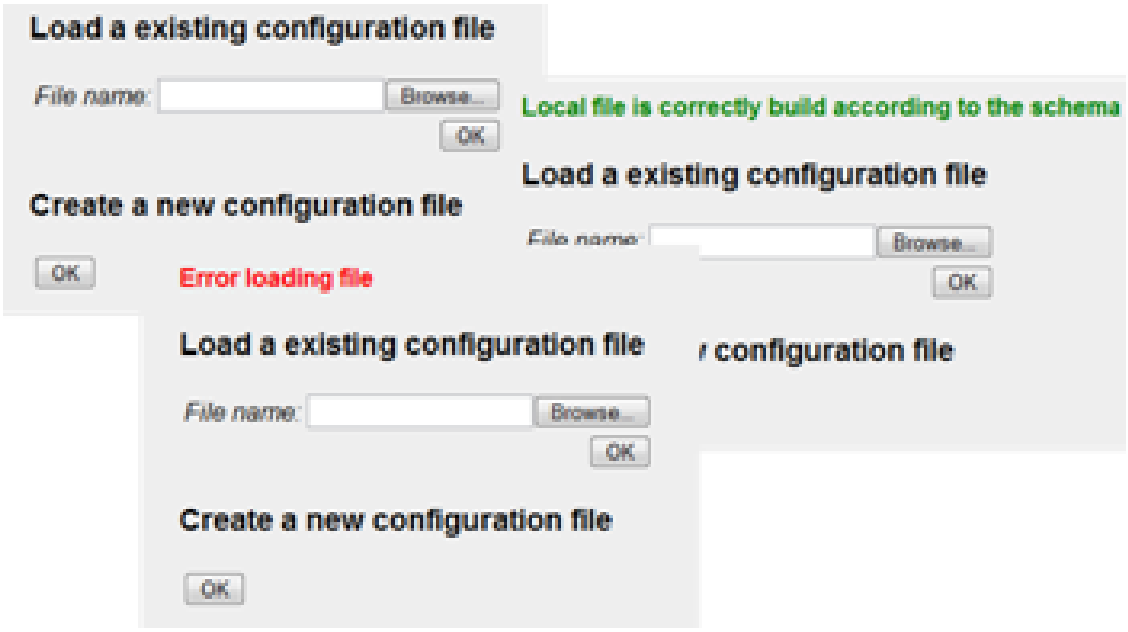

Figura 5.4: Escolher o ficheiro de configuração a utilizar e possíveis mensagens

segmentos desta lista podem ser arrastados para a parte direita, de modo a que um segmento ao ser largado na parte direita é adicionado à definição da mensagem e pode ser posteriormente movido para o local desejado.

A parte direita corresponde à definição da estrutura de uma mensagem. Os segmentos aqui listados correspondem a essa estrutura. Estes segmentos, ou grupos de segmentos, podem ser movidos para outra posição de uma forma fácil e sendo arrastados também podem ser movidos para dentro ou fora de um grupo.

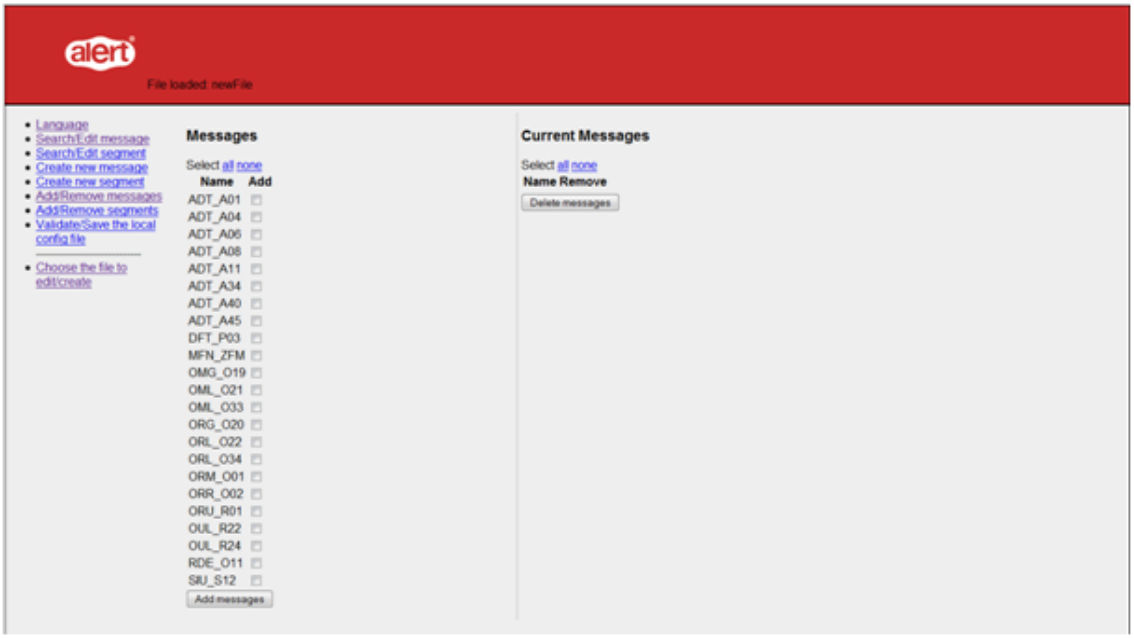

Figura 5.5: Adicionar/remover mensagens

As imagens 5.6, 5.7, 5.8, 5.9, 5.10 e 5.11 demonstram alguns exemplos de manipulação da estrutura de uma mensagem.

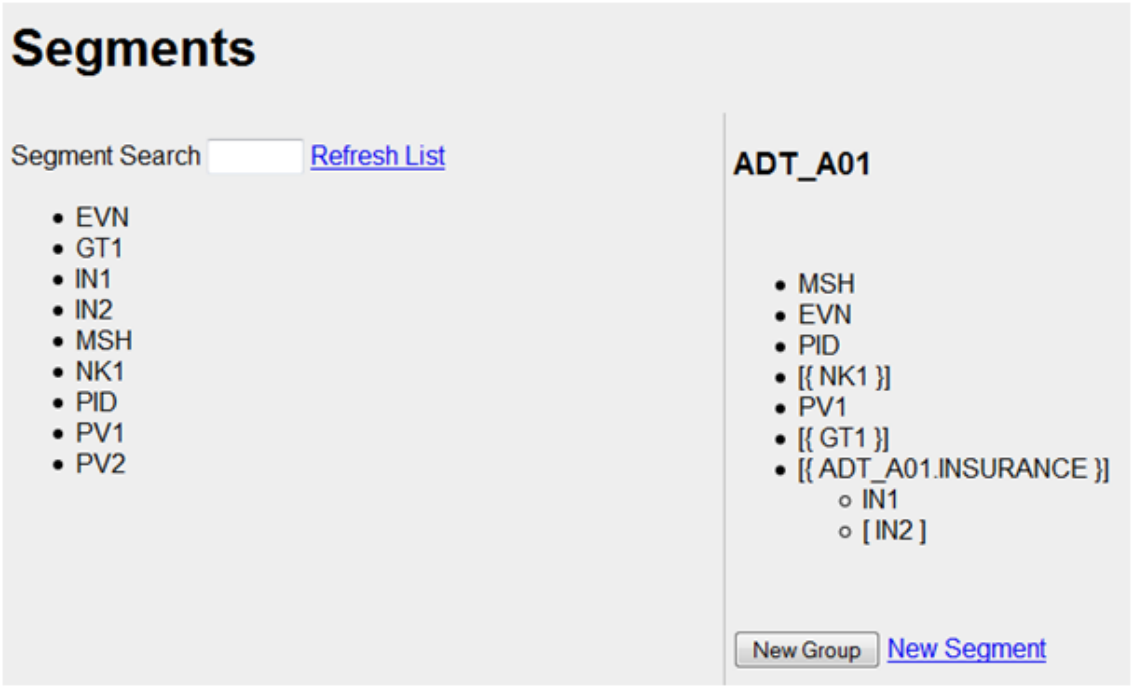

Figura 5.6: Definição de uma mensagem

Algumas das mensagens e operações que podem ser executadas na alteração da estrutura de uma mensagem encontram-se na figura 5.12 e correspondem a guardar a definição

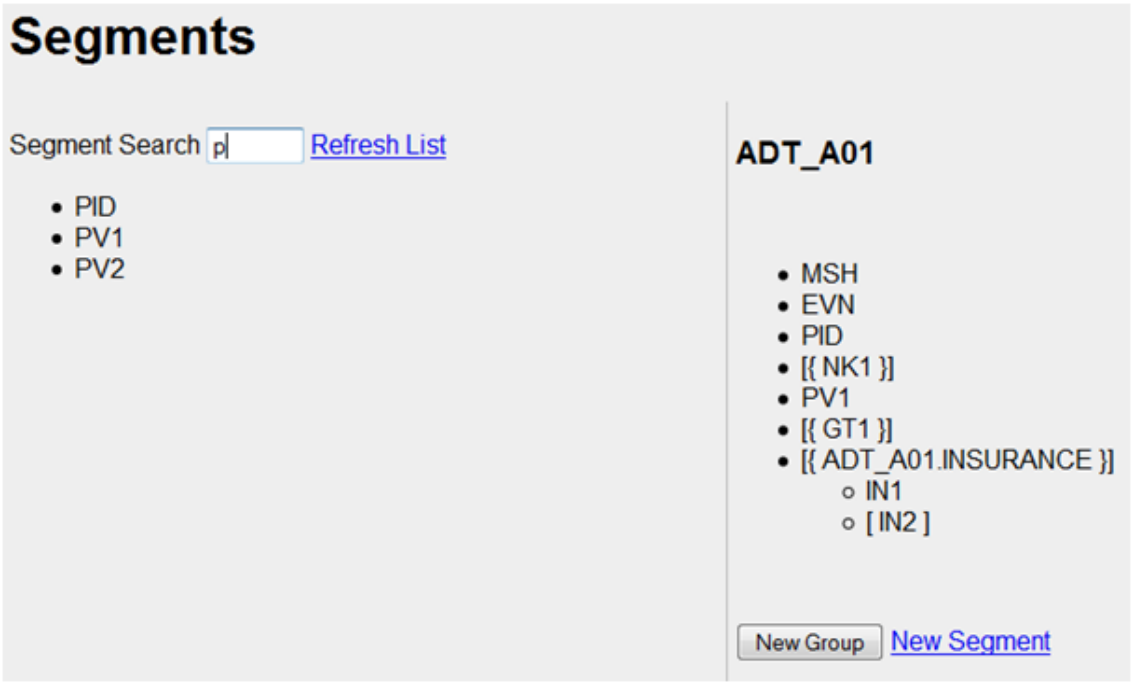

Figura 5.7: Filtrar lista de segmentos

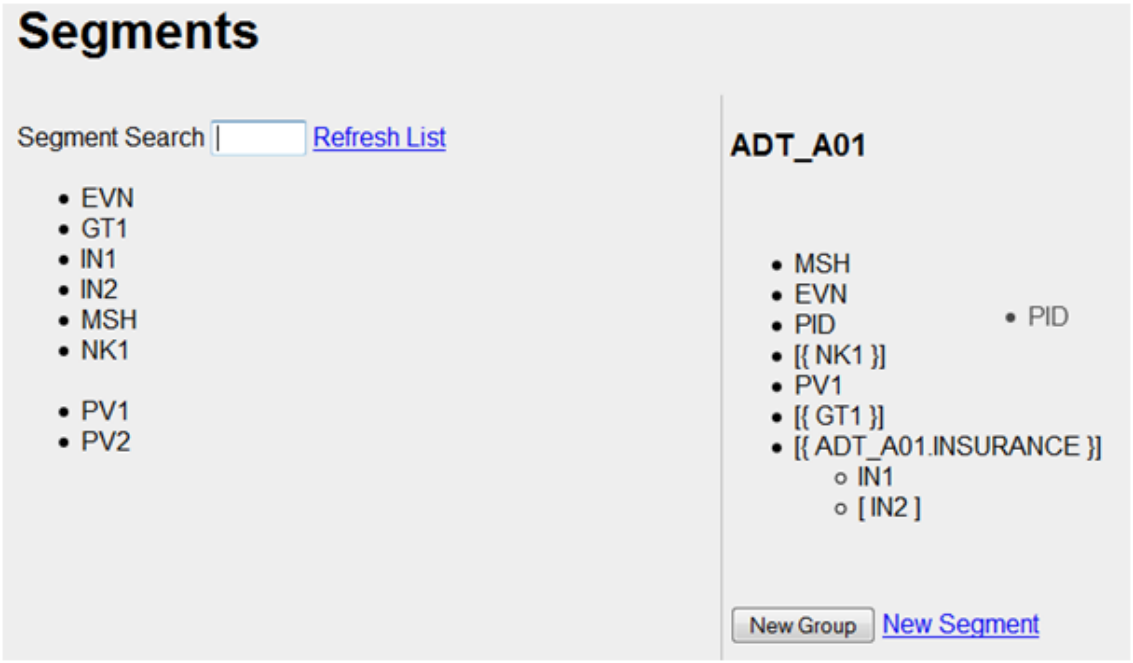

Figura 5.8: Adicionar um segmento

alterada, possíveis mensagens de erro, criar/adicionar um novo grupo e alterar as propriedades dos segmentos. Também é possível editar os campos de um segmento.

Ao criar uma nova mensagem, é possível escolher uma estrutura base de uma mensagem existente, do ficheiro local ou padrão. O nome de uma mensagem que esteja a ser

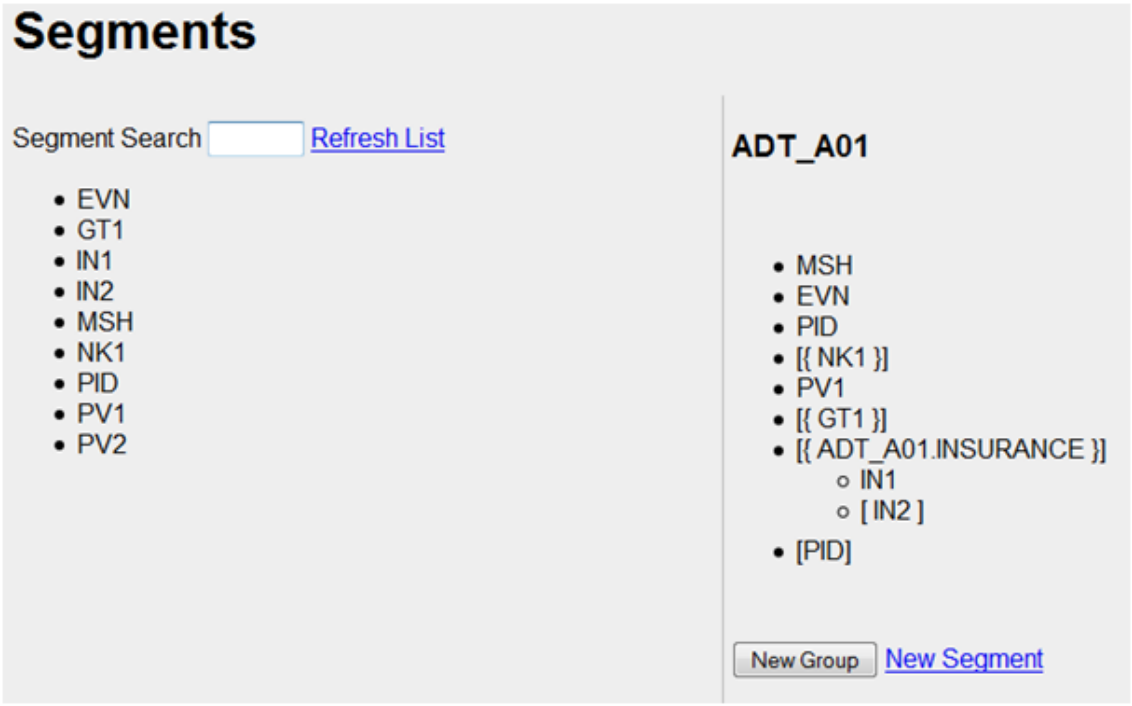

Figura 5.9: Segmento adicionado

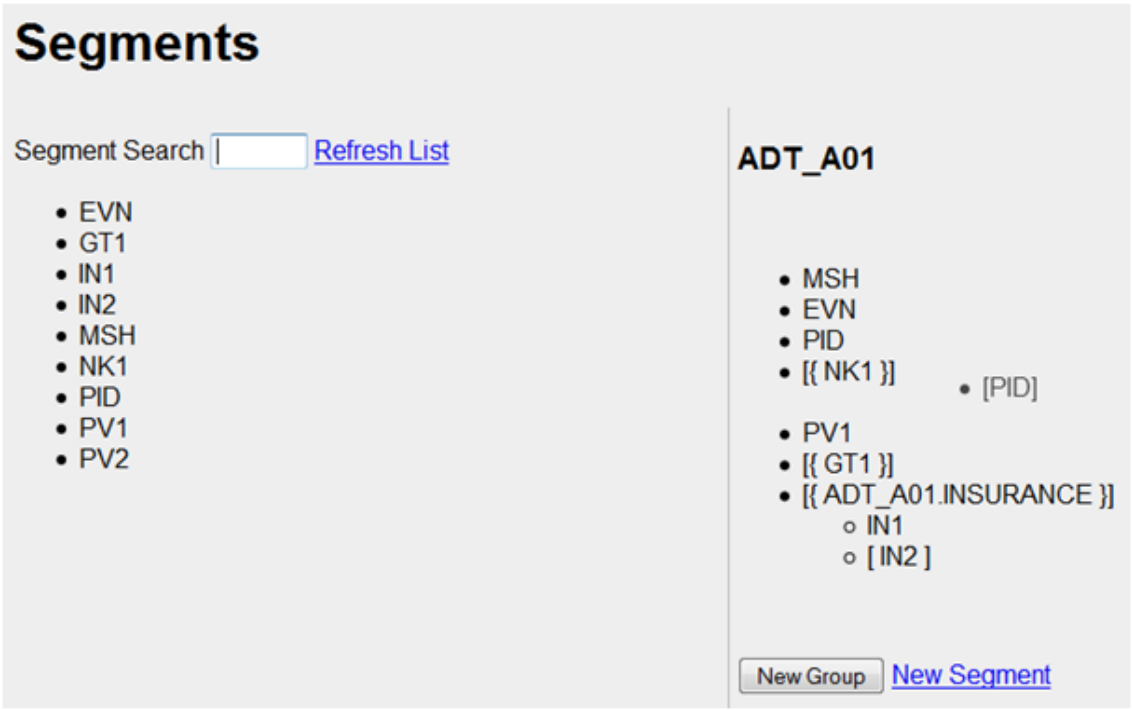

Figura 5.10: Alterar a posição de um segmento na estrutura da mensagem

criada não pode ainda existir, e tem que ter um comprimento mínimo de 3 caracteres. Algumas das mensagens de erro obtidas ao tentar criar uma nova mensagem encontram-se

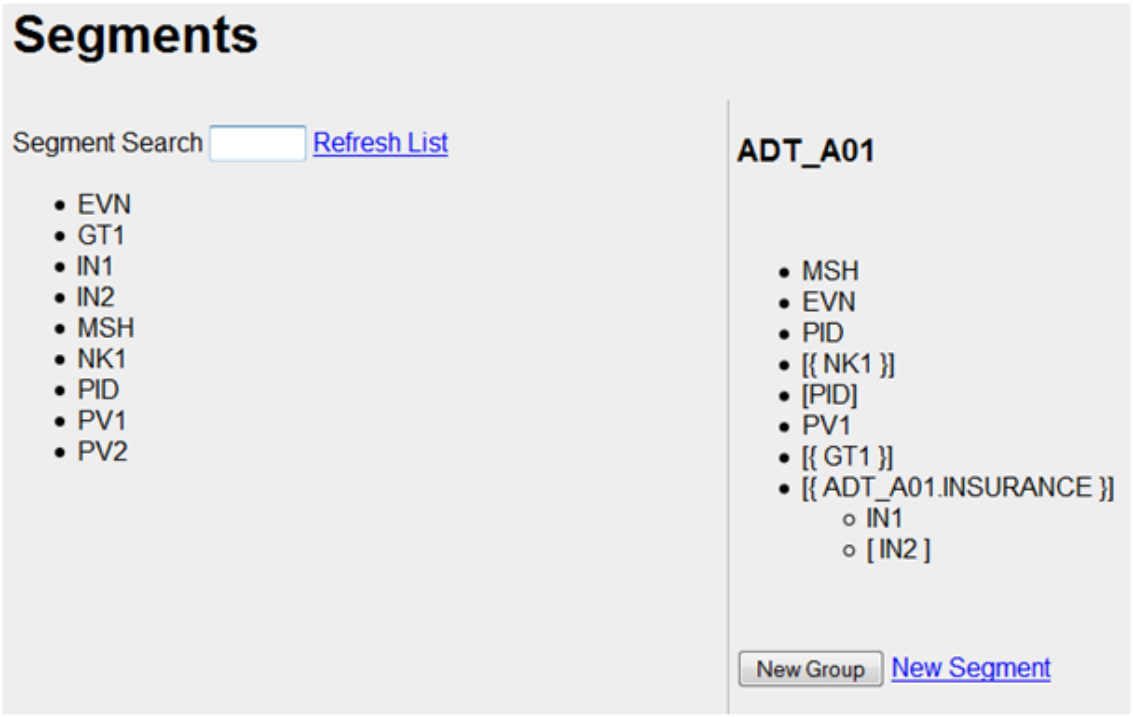

Figura 5.11: Posição de segmento alterada

ilustradas na figura 5.13.

As páginas de editar e criar um segmento são muito semelhantes. As únicas diferenças são as validações quando a informação é submetida e a possibilidade de definir o nome do segmento quando estamos a criar um novo. Algumas das mensagens exibidas nestas páginas e o aspecto destas podem ser visualizados na figura 5.14.

Um excerto, exemplo do ficheiro de definição criado por esta aplicação, pode ser consultado na figura 5.15.

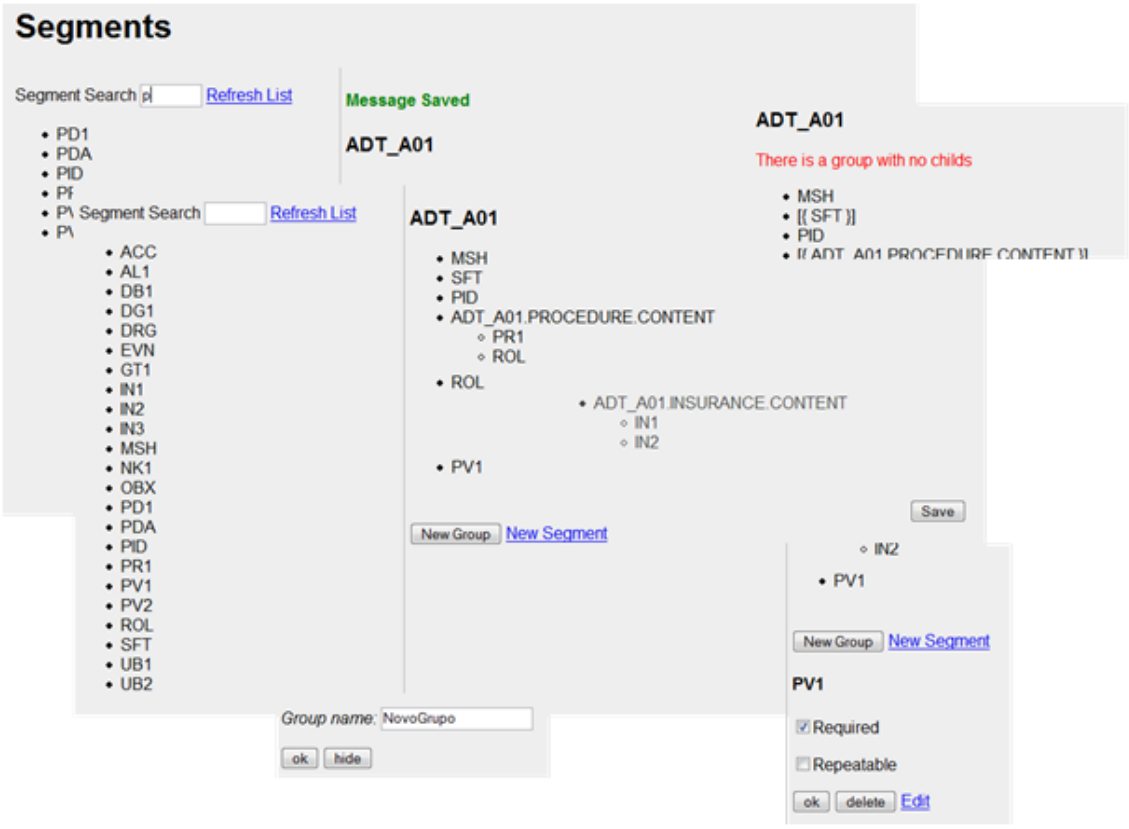

Figura 5.12: Editar mensagem exemplos

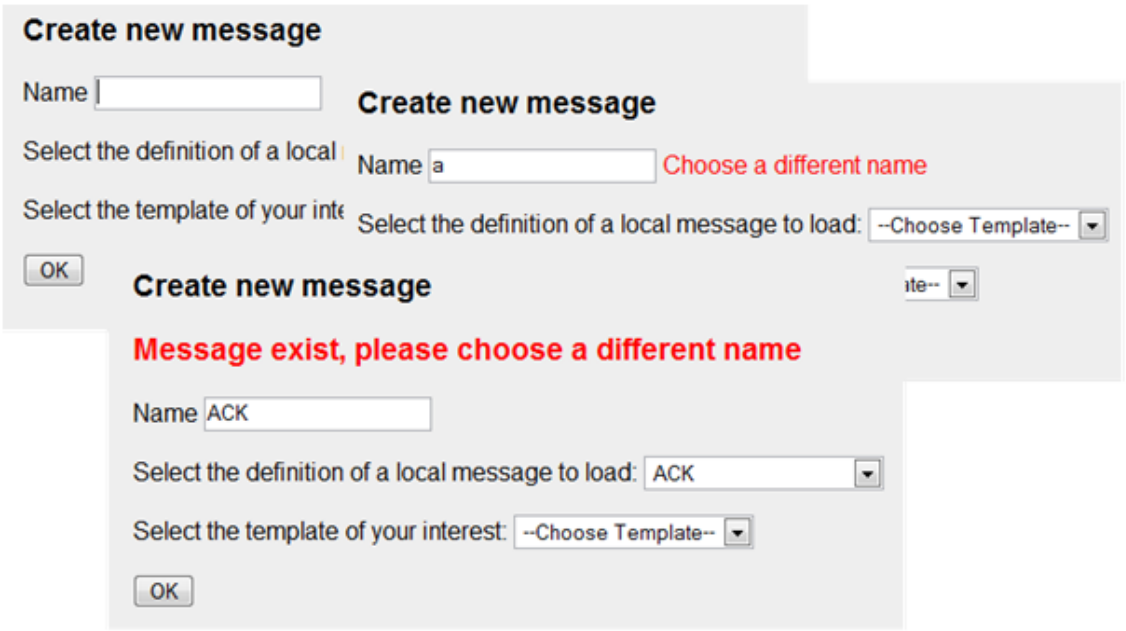

Figura 5.13: Criar uma nova mensagem

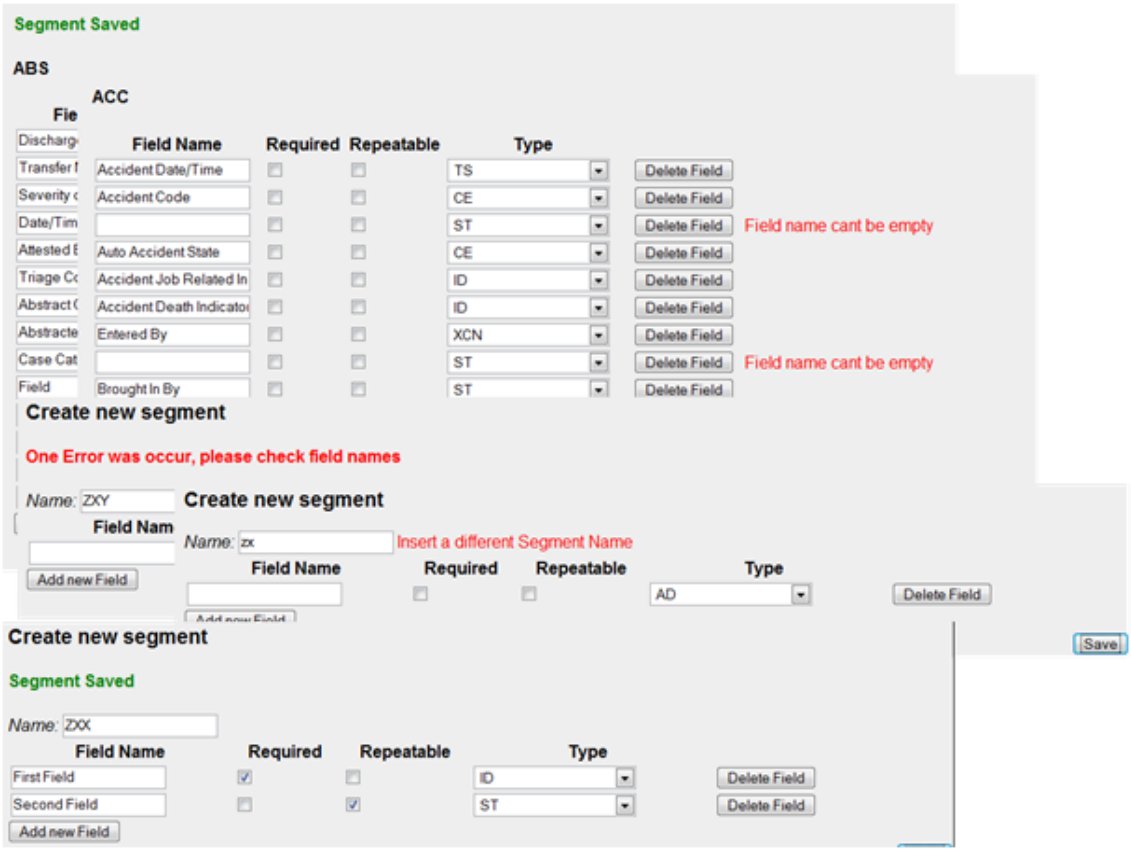

Figura 5.14: Criar/Editar um segmento

5.6 Desenvolvimento Futuro

Com o desenvolvimento das capacidades do *HL7 Parser* e de futuros módulos que incrementem as suas funcionalidades, esta ferramenta irá certamente acompanhar as necessidades de configuração/definição que advenham das novas funcionalidades.

Uma destas novas funcionalidades que serão brevemente desenvolvidas é a possibilidade de gravar a informação interpretada de uma mensagem válida numa base de dados. Esta ferramenta irá permitir indicar concretamente e de forma fácil em que tabela e campo se pretende guardar um determinado subcomponente.

O mecanismo para fazer este mapeamento será algo semelhante ao desenvolvido para a definição das mensagens em que basta arrastar o campo, componente, ou sub-componente para o campo de uma tabela onde pretendemos que essa informação seja guardada.

```
        <compose len="0" maxOccurs="unbounded" minOccurs="0" name="UB92 Locator 49 (National)" type="ST"/>
        <compose len="0" maxOccurs="unbounded" minOccurs="0" name="UB92 Locator 56 (State)" type="ST"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="UB92 Locator 57 (National)" type="ST"/>
        <compose len="0" maxOccurs="unbounded" minOccurs="0" name="UB92 Locator 78 (State)" type="ST"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Special Visit Count" type="NM"/>
      </segmentDef>
    - <segmentDef name="PDA">
        <compose len="0" maxOccurs="unbounded" minOccurs="0" name="Death Cause Code" type="CE"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Death Location" type="PL"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Death Certified Indicator" type="ID"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Death Certificate Signed Date/Time" type="TS"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Death Certified By" type="XCN"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Autopsy Indicator" type="ID"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Autopsy Start and End Date/Time" type="DR"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Autopsy Performed By" type="XCN"/>
        <compose len="0" maxOccurs="1" minOccurs="0" name="Coroner Indicator" type="ID"/>
      </segmentDef>
    </segments>
  - <messages>
    - <message name="TESTE">
        <segment name="MSH" minOccurs="1" maxOccurs="1"/>
        <segment name="SFT" minOccurs="0" maxOccurs="unbounded"/>
        <segment name="EVN" minOccurs="1" maxOccurs="1"/>
        <segment name="PID" minOccurs="1" maxOccurs="1"/>
        <segment name="PD1" minOccurs="0" maxOccurs="1"/>
        <segment name="ROL" minOccurs="0" maxOccurs="unbounded"/>
        <segment name="NK1" minOccurs="0" maxOccurs="unbounded"/>
        <segment name="PV1" minOccurs="1" maxOccurs="1"/>
        <segment name="PV2" minOccurs="0" maxOccurs="1"/>
        <segment name="ROL" minOccurs="0" maxOccurs="unbounded"/>
```

Figura 5.15: Excerto de um ficheiro construído

Capítulo 6

Message Browser

Esta ferramenta permite visualizar a informação contida numa mensagem, de forma estruturada. Divide a informação de acordo com o ficheiro de definições especificado e exibe essa informação através de uma "árvore" que se pode analisar em pormenor até ao nível mais simples de dados permitido pelo HL7.

Um exemplo desta árvore pode ver-se nas figuras 6.1 e 6.2.

Figura 6.1: Message Browser Tree View

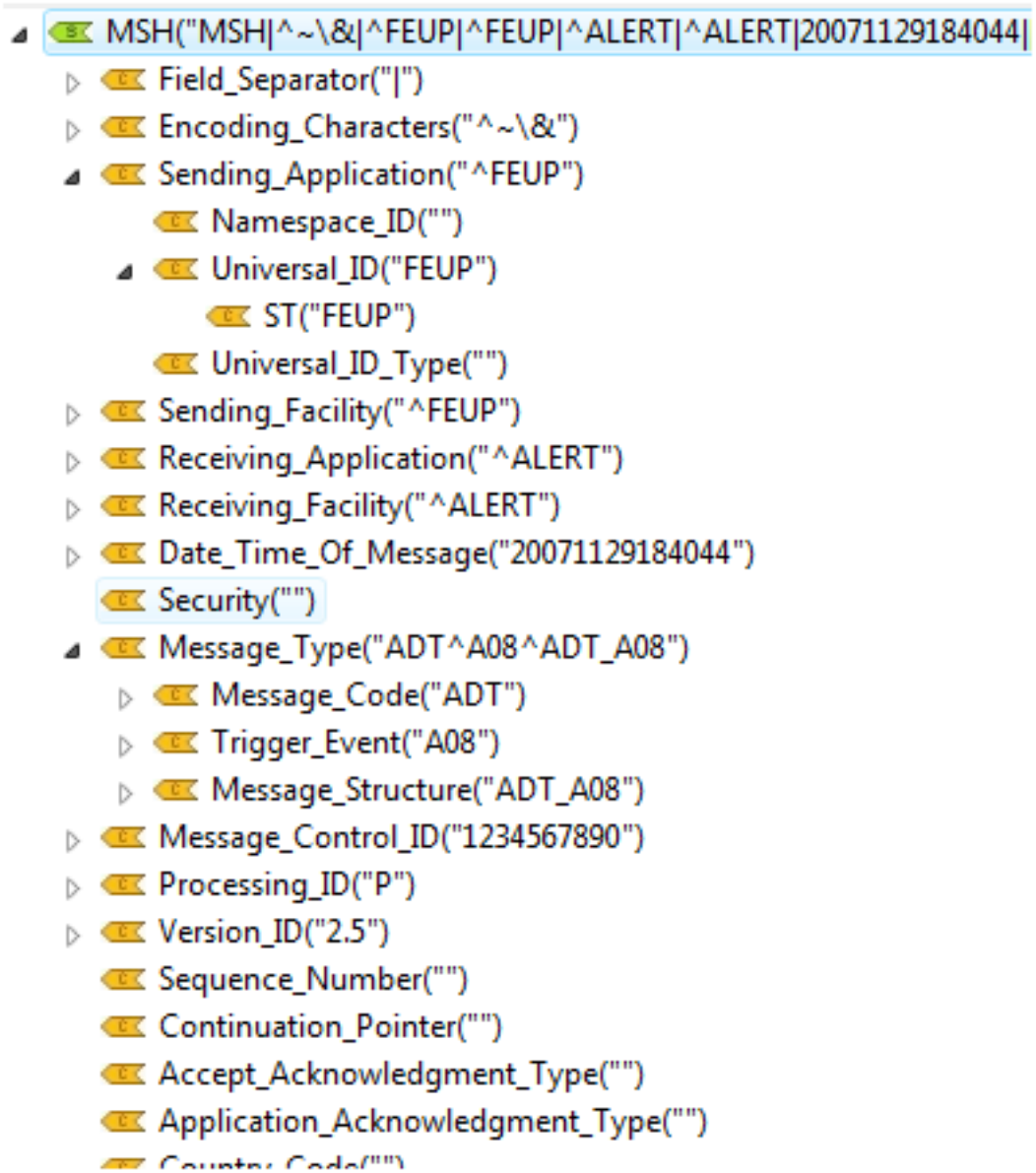

Figura 6.2: Message Browser Tree View 2

6.1 Estrutura de Classes

Esta aplicação é apenas constituída por 2 classes. Sendo uma a GUI e a outra uma classe com a lógica da aplicação. São elas:

- **MainWindow**, esta classe é a GUI da aplicação, foi utilizado o plug-in Visual Edit para o eclipse que permite manipular de forma gráfica os componentes visuais. Os componentes visuais utilizados foram os da Framework SWT;
- **MessageBrowserLogic**, esta classe contém a lógica da aplicação, qualquer interacção da classe GUI que não seja apenas visual é executada através desta classe;

Esta ferramenta utiliza um outro projecto, o *HL7 Parser* para validar e fazer a interpretação da mensagem a exibir.

6.2 Diagrama de sequência

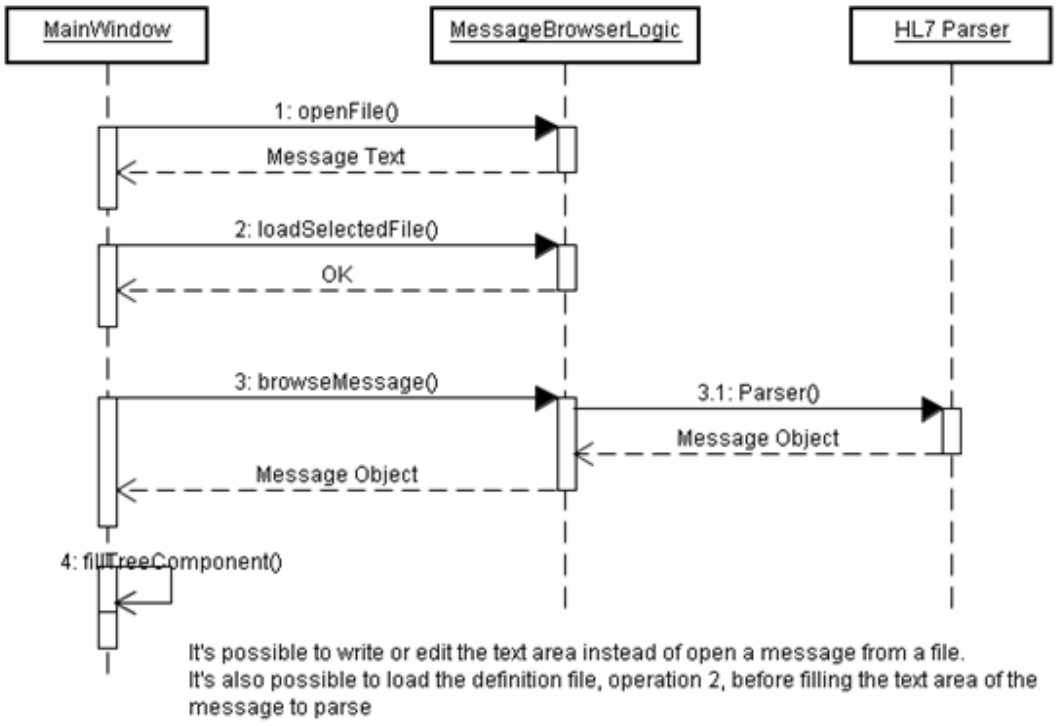

Figura 6.3: Diagrama de sequência

O diagrama de sequência mostrado na figura 6.3 é apenas um exemplo de execução onde inicialmente é possível realizar o passo 1 ou 2, em que o passo 1 pode ser substituído pela introdução manual do texto da mensagem na caixa de texto, em vez da leitura da mensagem a partir de um ficheiro.

Após ser carregado o ficheiro de definição (passo 2 na figura) o botão "Parse" fica acessível. No caso de ser executado o "parse", mas a caixa de texto de entrada estiver vazia não acontece nada. Caso exista informação será feita a validação da mensagem. Caso existam erros ou avisos estes serão visualizados e se possível a estrutura em árvore é criada. Para facilitar a identificação do elemento que originou um erro, na árvore esses elementos são assinalados com um ícone vermelho.

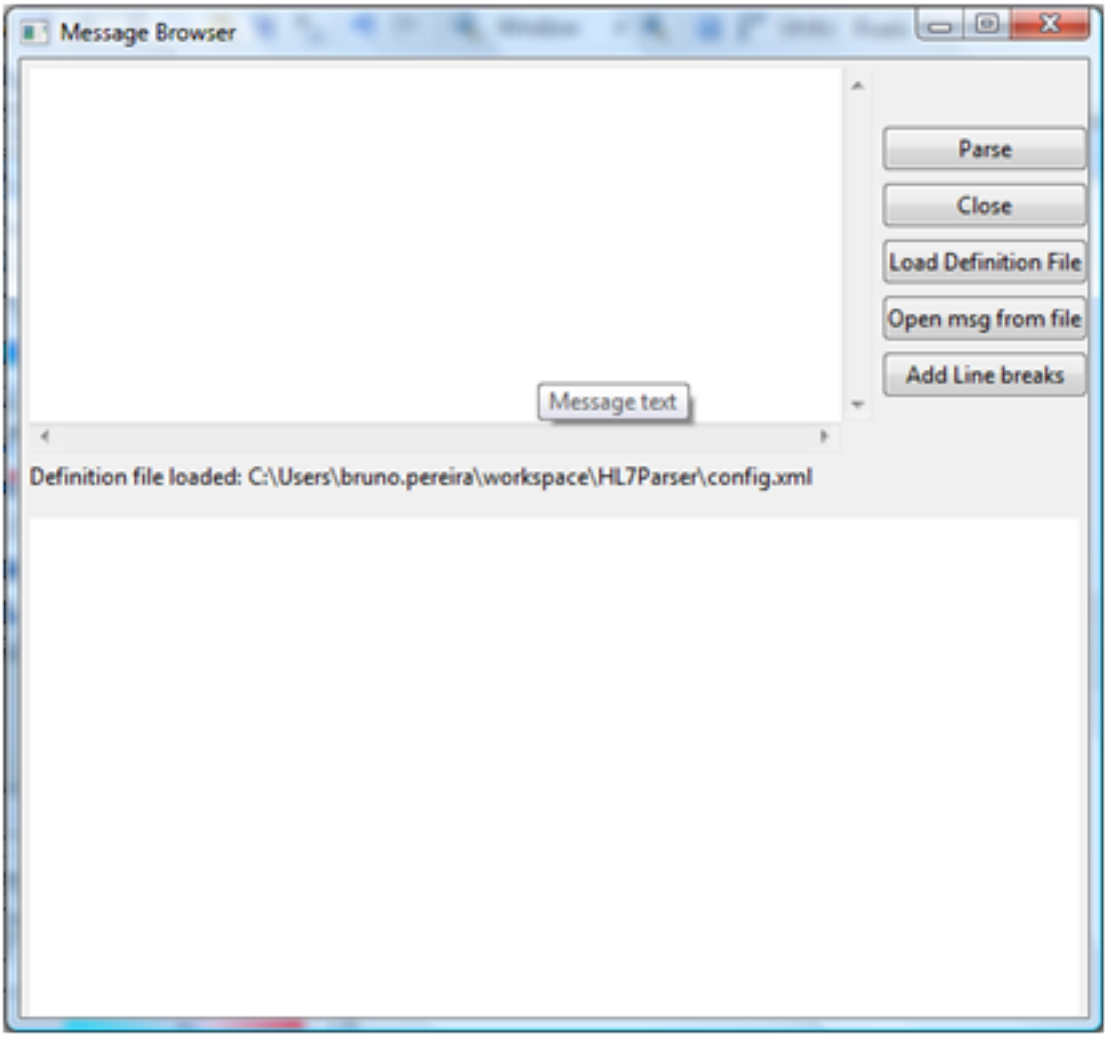

Figura 6.4: Message Browser GUI

6.3 Aspecto Gráfico

O intuito dos botões é o seguinte:

- **Parse**, exibe a informação de uma mensagem sobre a forma de uma árvore de acordo com o ficheiro de definição indicado;
- **Load Definition File**, permite indicar qual o ficheiro de definição de mensagem a utilizar;
- **Open msg from File**, carrega o conteúdo de um ficheiro para a caixa de texto que terá o texto da mensagem a analisar;
- **Close**, fecha a aplicação;

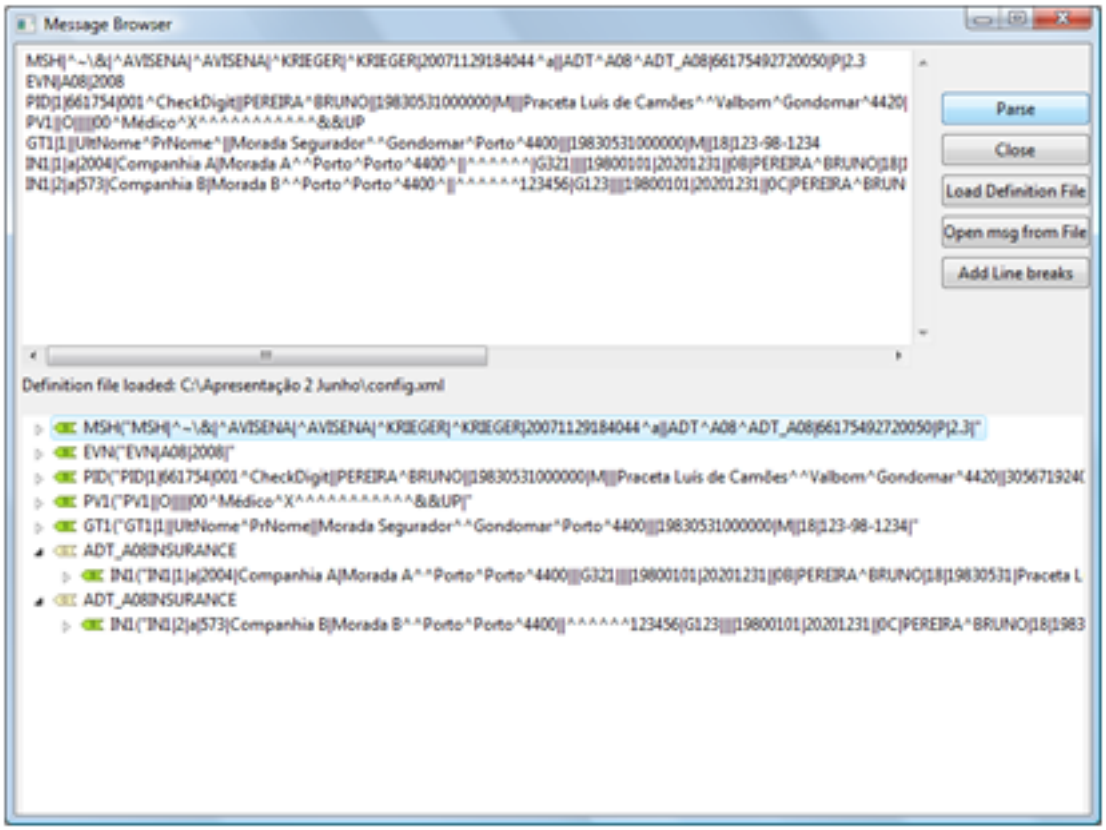

Figura 6.5: Message Browser GUI

- **Add line breaks**, a caixa de texto do SWT não insere uma quebra de linha quando identifica o carácter "carriage return", quando é feito o "copy/paste" de uma mensagem por vezes são estes os únicos caracteres que são colocados entre os segmentos. Quando este botão é pressionado identifica esses caracteres e insere os caracteres de "line feed";

A primeira área exibida nas imagens, corresponde a uma caixa de texto onde se deve colocar o texto da mensagem a analisar.

A segunda área é um "Tree component" e tal como se pode ver, é onde será exibida a mensagem após ter sido efectuada a sua interpretação.

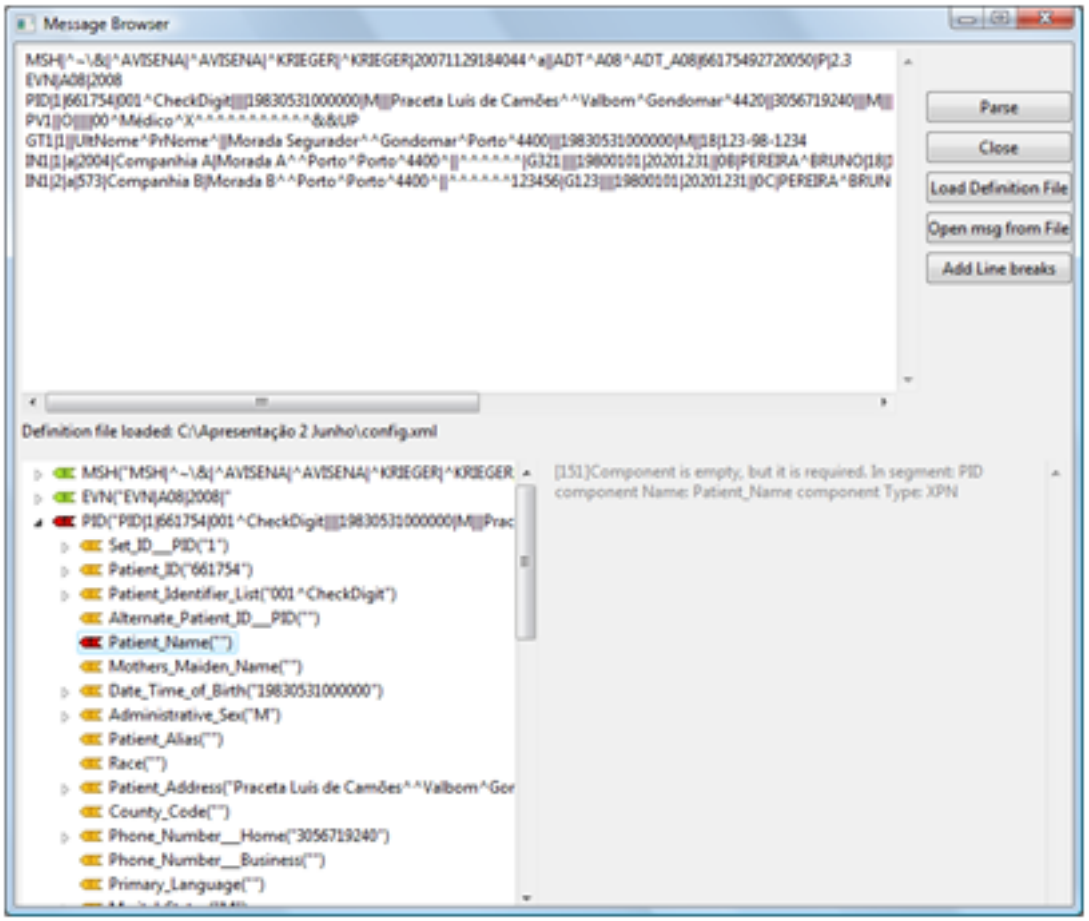

Figura 6.6: Erro durante o parse

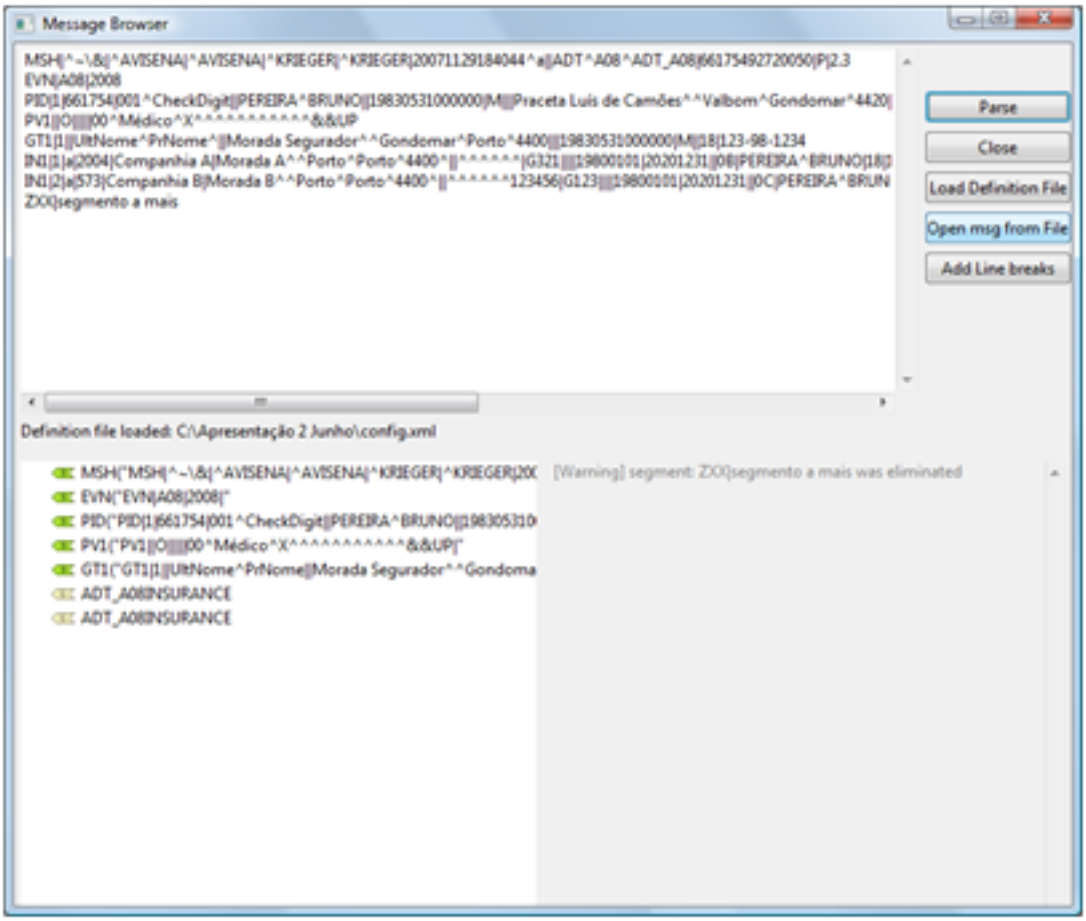

Figura 6.7: Aviso durante o parse

Capítulo 7

Cliente & Servidor LLP

Estas aplicações estendem o projecto *HL7 Parser*, acrescentando um novo meio de comunicação entre sistemas, que é neste caso a comunicação sobre TCP/IP para a recepção e envio de mensagens HL7. Quer o cliente quer o servidor foram desenvolvidos em projectos independentes, que encaixam e invocam métodos disponibilizados pelo *HL7 Parser*. Constituem, no entanto, um novo módulo daquela ferramenta.

O servidor LLP corresponde a um serviço que fica a correr numa máquina e à escuta num determinado porto. Quando um cliente tenta ligar-se ao servidor este cede o *socket* para escrita e escuta a mensagem. Quando a mensagem a receber é identificada como completa envia uma mensagem ACK.

Essa mensagem ACK indica se existem erros ou não. Caso existam erros é enviada uma descrição desses erros na mensagem ACK. No caso da validação se encontrar desligada apenas irá validar a mensagem quanto ao tipo.

O cliente LLP corresponde a uma aplicação que se liga ao servidor, enviando uma mensagem. Quando terminar de enviar a mensagem ficará à espera que lhe seja enviada uma resposta, que deverá ser uma mensagem ACK.

LLP significa *Lower Layer Protocol*, não especificando contudo qual o meio de comunicação a utilizar. Escolheram-se, no entanto, os sockects TCP/IP como meio de comunicação das ferramentas. Este protocolo apenas obriga a encapsular a mensagem a enviar da seguinte forma:

- Cabeçalho (carácter “vertical tab” 0x0B);
- Mensagem que se pretende enviar sem esquecer que entre cada segmento deverá ser enviado o carácter de “carriage return” 0x0D;
- Rodapé, onde deverão ser enviados os caracteres “file separator” 0x1C e “carriage return” 0x0D;

É através do cabeçalho que é identificada a recepção de uma nova mensagem, e é através do rodapé que é identificado o fim de mensagem.

7.1 Estrutura de Classes

Para cada uma das duas ferramentas descritas neste documento foi desenvolvida uma GUI em SWT com o auxílio do plugin Visual Editor para o eclipse IDE.

Estas ferramentas são simples e utilizam o projecto *HL7 Parser* para verificar se as mensagens que receberam ou estão prestes a enviar são válidas. No caso do cliente e caso seja detectada alguma anomalia exibe-se ao utilizador uma descrição da anomalia, sendo o utilizador questionado se quer enviar a mensagem ou não.

7.1.1 Servidor

- **GenerateACK**, classe que gera uma mensagem ACK com base numa mensagem recebida;
- **LLPBussinessLogic**, classe que contém as operações lógicas que não envolvam o socket;
- **LLPServerGUI**, classe com os componentes gráficos da aplicação;
- **SendInfoToGUI**, classe responsável por escrever a informação disponibilizada pelo socket do server na GUI da aplicação;
- **StreamTreatment**, classe que disponibiliza os métodos de tratamento dos streams a enviar e recebidos. Disponibiliza o método que encapsula a mensagem a enviar e faz o append dos streams recebidos, onde o conjunto destes streams constituem a mensagem;
- **TcpServer**, classe responsável pelo socket do servidor;

7.1.2 Cliente

- **LLPBussinessLogic**, classe que tem as operações lógicas que não envolvam o socket;
- **LLPClienteGUI**, classe com os componentes gráficos da aplicação;
- **StreamTreatment**, classe que disponibiliza os métodos de tratamento dos streams a enviar e recebidos. Disponibiliza o método que encapsula a mensagem a enviar e faz o append dos streams recebidos, onde o conjunto destes streams constituem a mensagem;

- **TcpClient**, classe responsável pelo socket do client;

7.2 Arquitectura

A arquitectura destas duas ferramentas é semelhante e ambas seguem uma arquitectura de 3 camadas:

- Visualização e interface com o utilizador, é a camada responsável pela interacção da aplicação com o utilizador. É responsável pela ilustração de toda a informação perceptível ao utilizador e pelo encaminhamento dos pedidos do utilizador para a camada de lógica da aplicação;
- Lógica de negócio, é a camada responsável pelo tratamento de informação e dados. É nesta camada que é realizada a comunicação com o módulo de *HL7 Parser*, a construção de mensagens ACK e encapsulamento/desencapsulamento das mensagens;
- Acesso aos dados, é a camada responsável pelo acesso aos dados. É nesta camada que são mantidos os canais e são realizadas as escritas e leituras para os mesmos;

7.2.1 Servidor

Para além da estrutura em 3 camadas referidas anteriormente, na figura 7.1 podemos identificar também as principais funcionalidades de cada camada e a camada que faz a interligação com o projecto *HL7 Parser*.

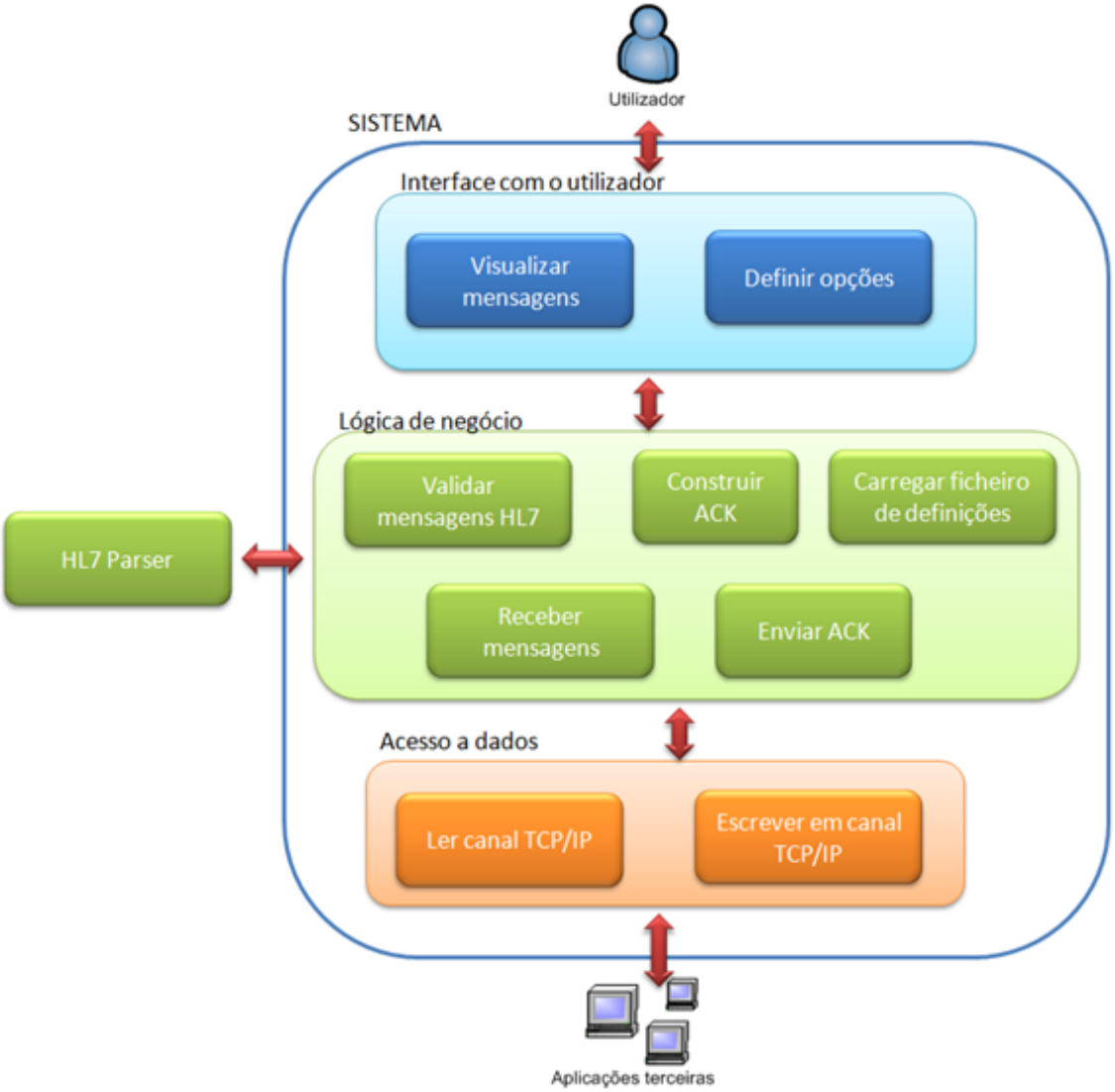

Figura 7.1: Arquitectura do Servidor LLP

7.2.2 Cliente

Para além da estrutura em 3 camadas referidas anteriormente, na figura 7.2 podemos identificar também as principais funcionalidades de cada camada e a camada que faz a interligação com o projecto *HL7 Parser*.

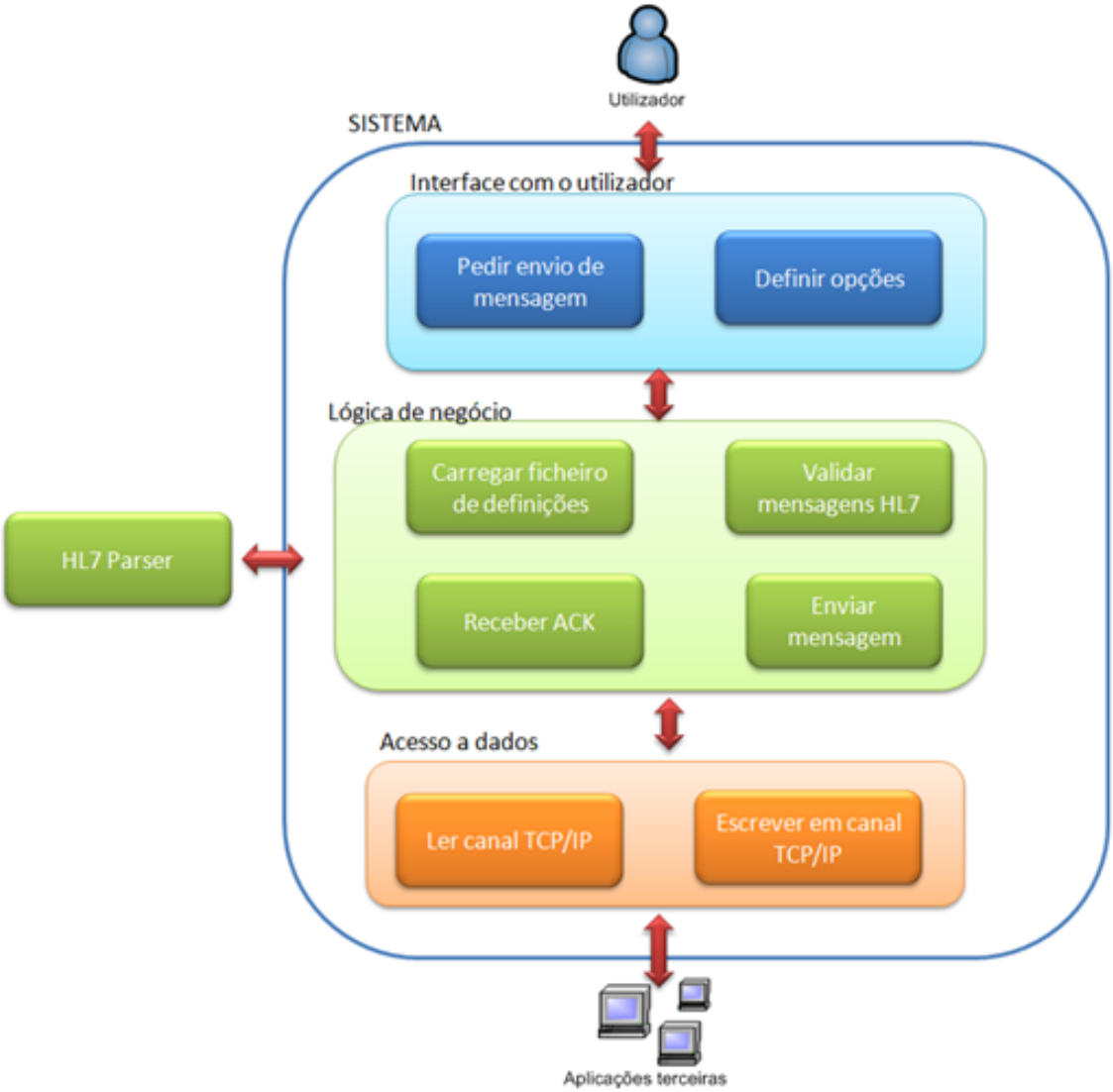

Figura 7.2: Arquitectura do Cliente LLP

7.3 Diagrama de sequência

7.3.1 Servidor

A aplicação servidora tem um conjunto de operações/tarefas que executa sequencialmente. Estas tarefas e a sua ordem podem ser consultadas na figura 7.3.

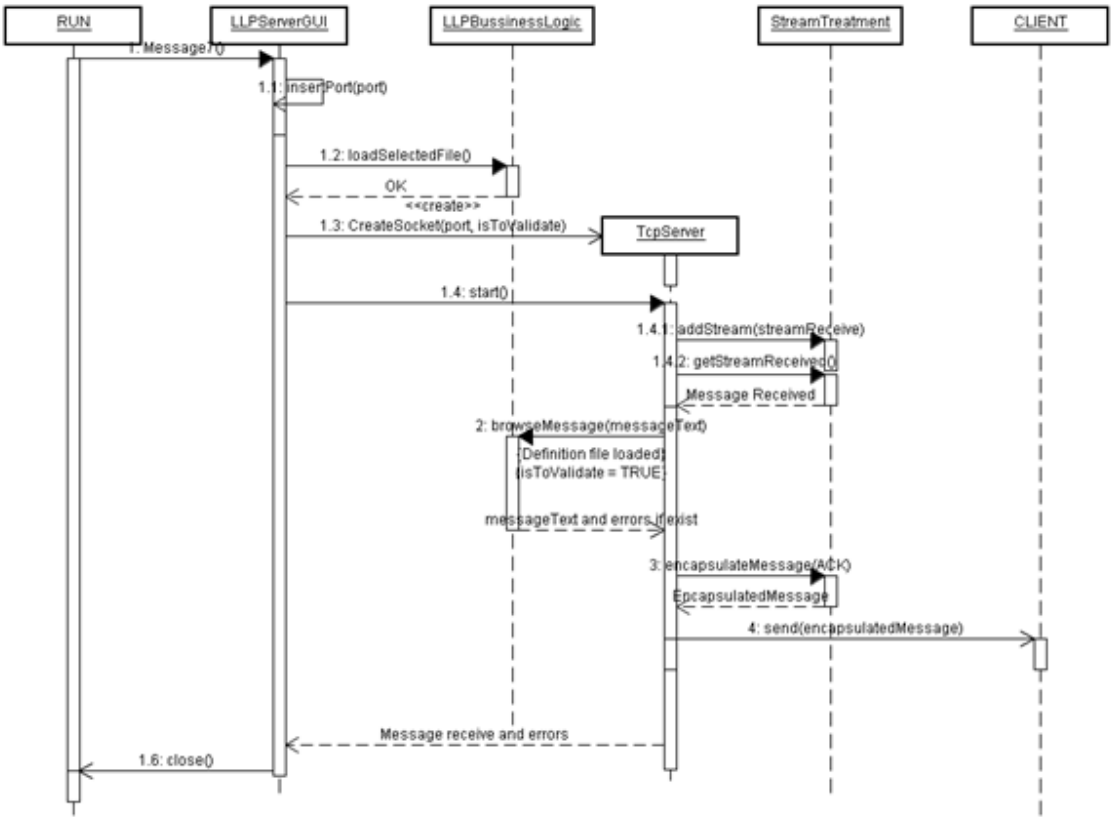

Figura 7.3: Diagrama de sequência da aplicação Server LLP

7.3.2 Cliente

Tal como a aplicação servidora, a aplicação cliente também tem um conjunto de operações/tarefas que executa sequencialmente. Estas tarefas e a sua ordem podem ser consultadas na figura 7.4.

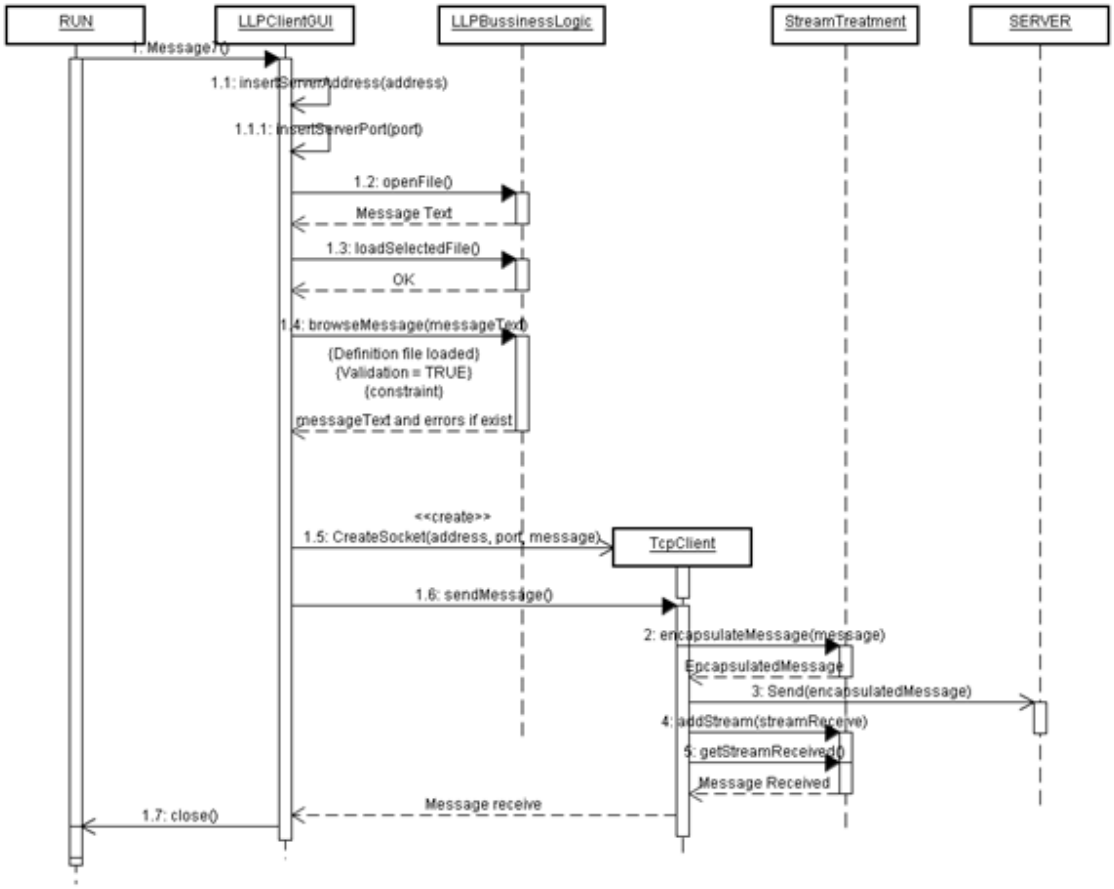

Figura 7.4: Diagrama de sequência da aplicação Client LLP

7.4 Aspecto Gráfico

No servidor e no cliente as mensagens recebidas ou a enviar apenas são validadas se o utilizador tiver especificado qual o ficheiro de definição de mensagens a utilizar e assinalar que deseja fazer essa validação.

7.4.1 Servidor

As opções do servidor resumem-se a:

- **Start Listening**, permite iniciar o socket servidor e ficar à escuta de mensagens;
- **Stop Listening**, permite parar a leitura do socket e libertar os seus recursos;
- **Load** Definition File, permite carregar um ficheiro de definição de mensagens;

Para alterar o modo de validação de mensagens é necessário que o servidor se encontre em modo STOP.

Na caixa de texto, não editável, são exibidas as mensagens recebidas assim como informação relevante. Caso a validação se encontre assinalada será exibida também nesta caixa de texto se a mensagem recebida é válida ou inválida e, caso existam, são exibidos os erros ou avisos.

É também possível especificar o tamanho máximo de clientes que ficam em fila, assim como o número máximo de comunicações simultâneas suportadas pelo porto.

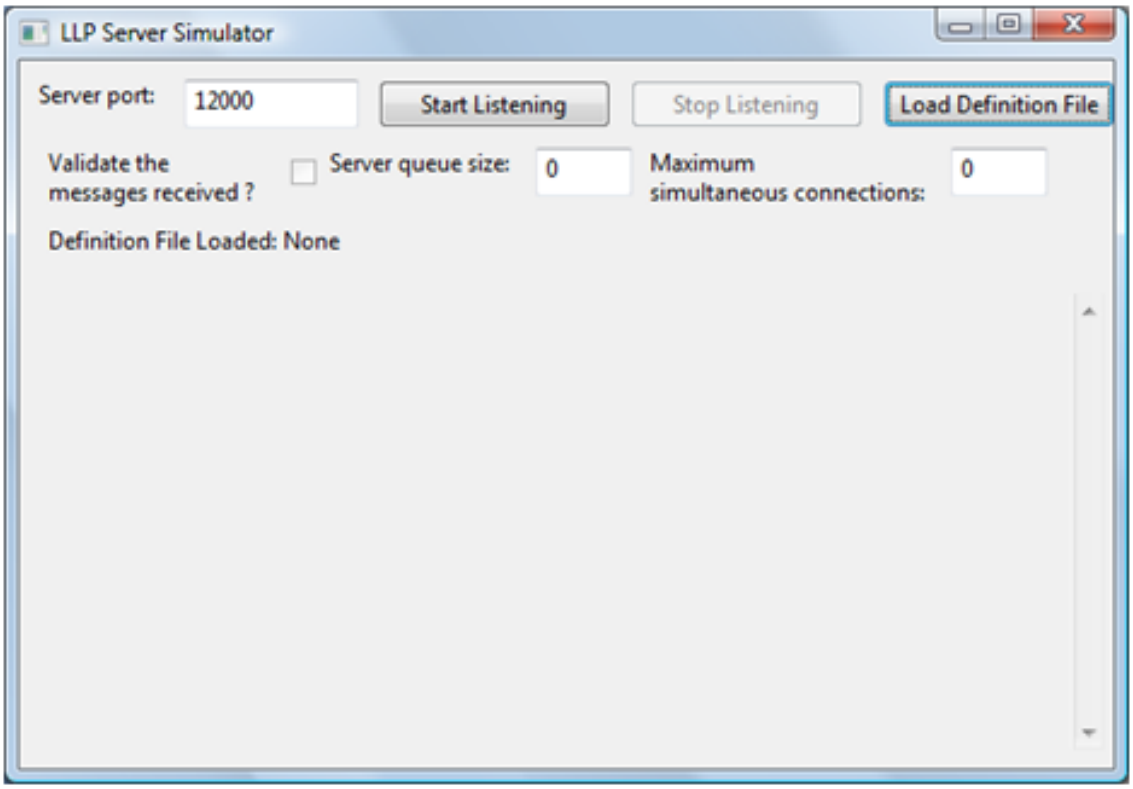

Figura 7.5: Server LLP GUI

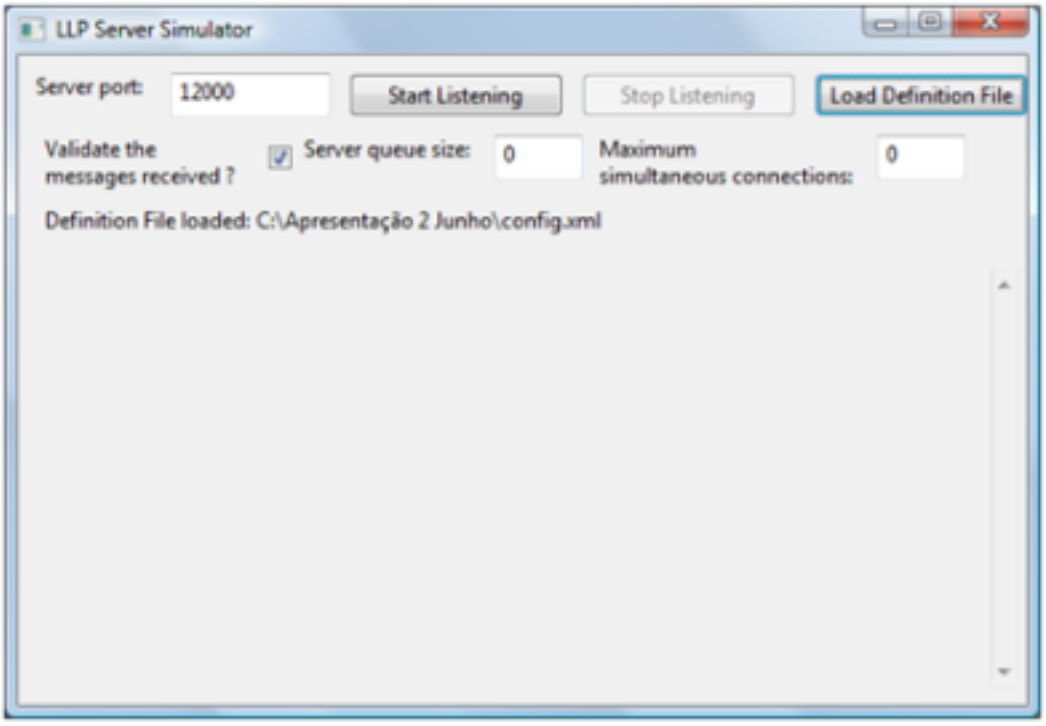

Figura 7.6: Server LLP GUI

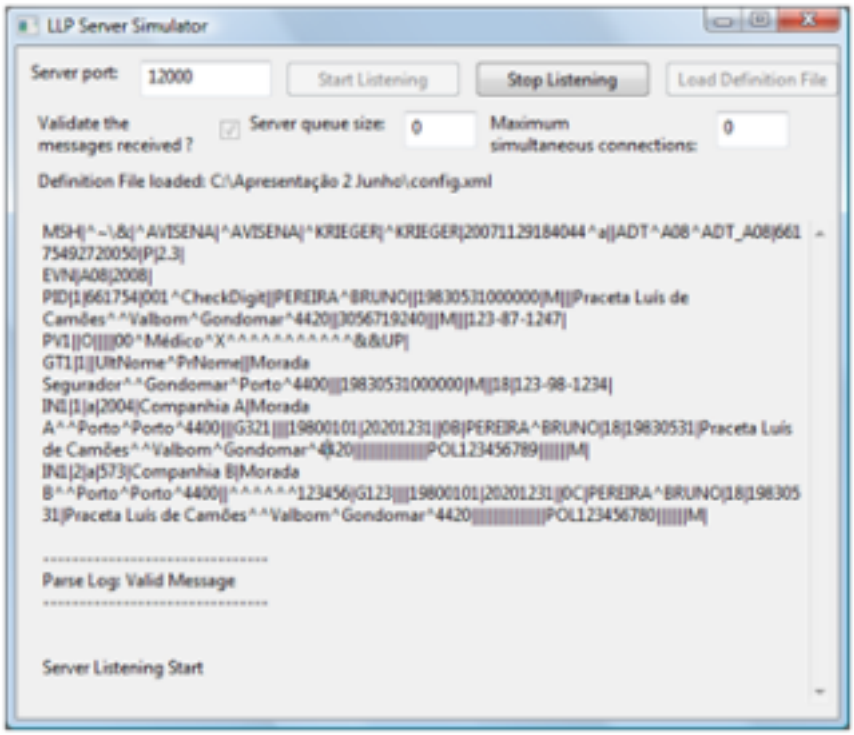

Figura 7.7: Server LLP mensagem recebida válida

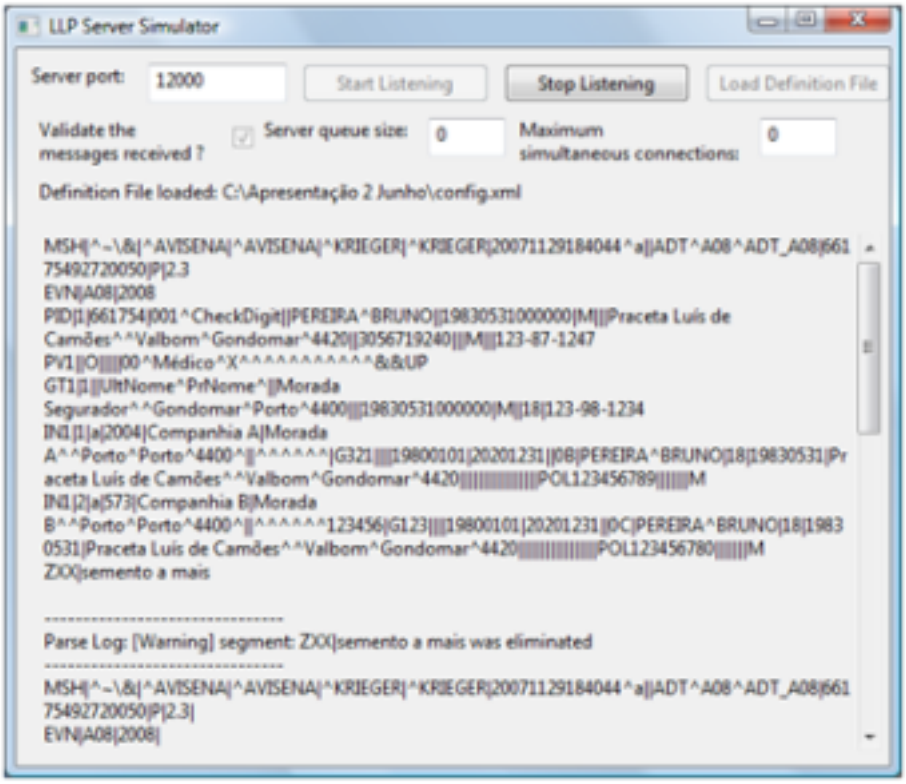

Figura 7.8: Server LLP mensagem recebida com avisos

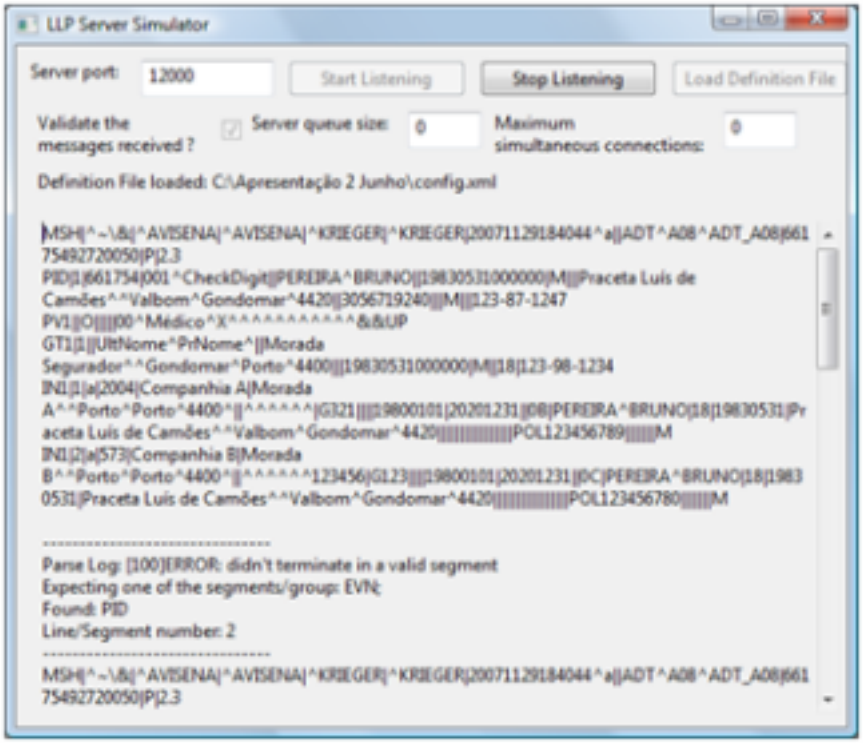

Figura 7.9: Server LLP mensagem recebida com erro

7.4.2 Cliente

Opções do Cliente:

- **Load Definition File**, permite carregar um ficheiro de definição de mensagens;
- **Open Message from File**, permite copiar o conteúdo de um ficheiro para a caixa de texto que contém a informação a enviar;
- **Send**, permite abrir o socket cliente e enviar a informação contida na caixa de texto, ou caso a opção de validar se encontre activa, enviar o texto da mensagem após ter sido validado;
- **Close**, permite fechar a aplicação;

A caixa de texto mais à esquerda contém a informação a enviar para o servidor, enquanto a caixa de texto não editável, mais à direita, contém as mensagens recebidas e outras informações relevantes.

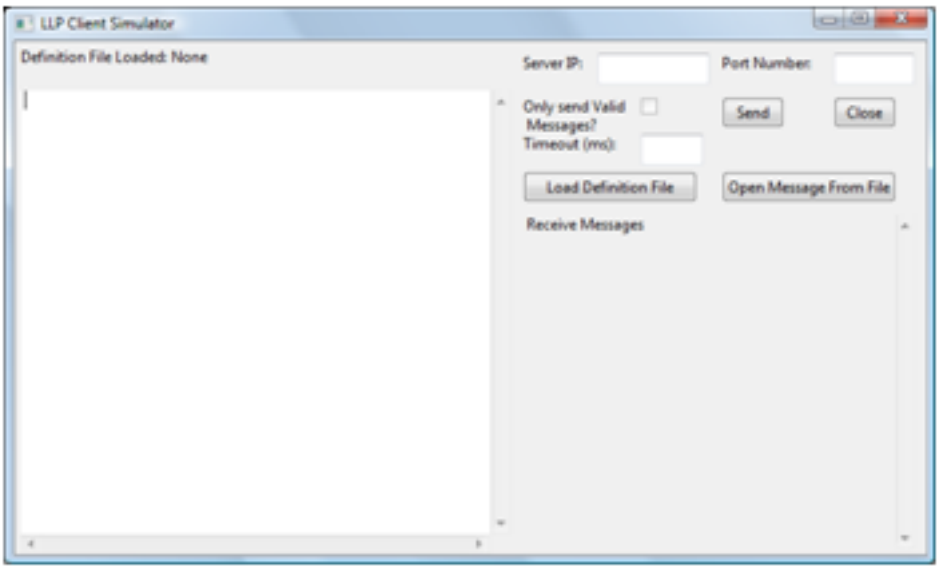

Figura 7.10: Cliente LLP GUI

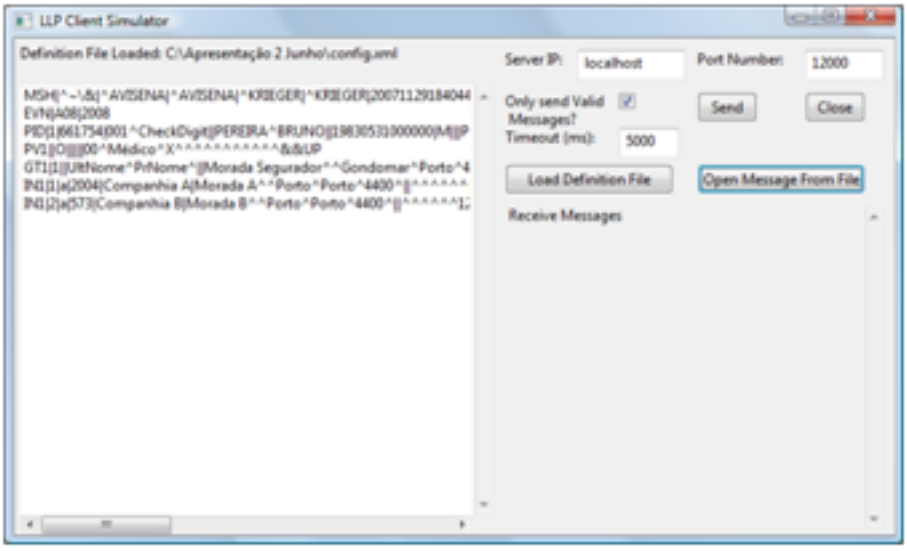

Figura 7.11: Cliente LLP GUI

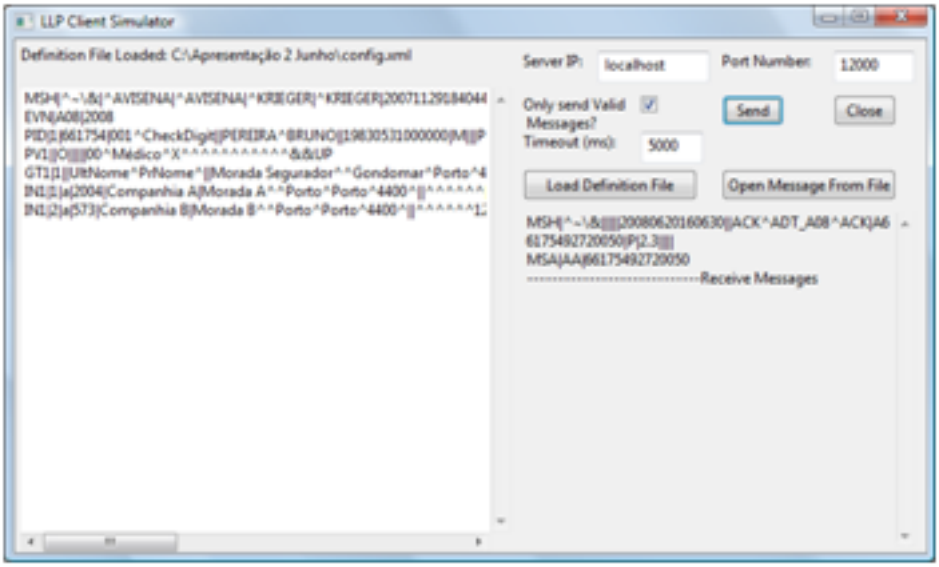

Figura 7.12: Cliente LLP GUI

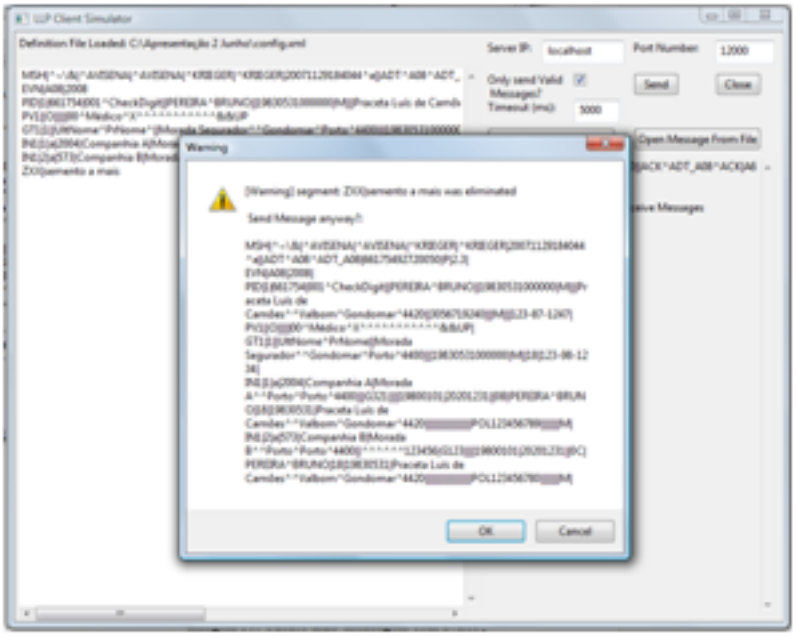

Figura 7.13: Cliente LLP mensagem com avisos

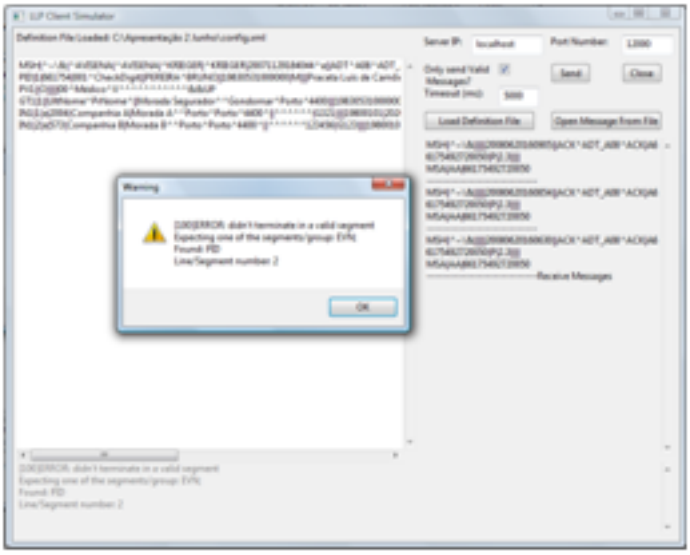

Figura 7.14: Cliente LLP mensagem com erro

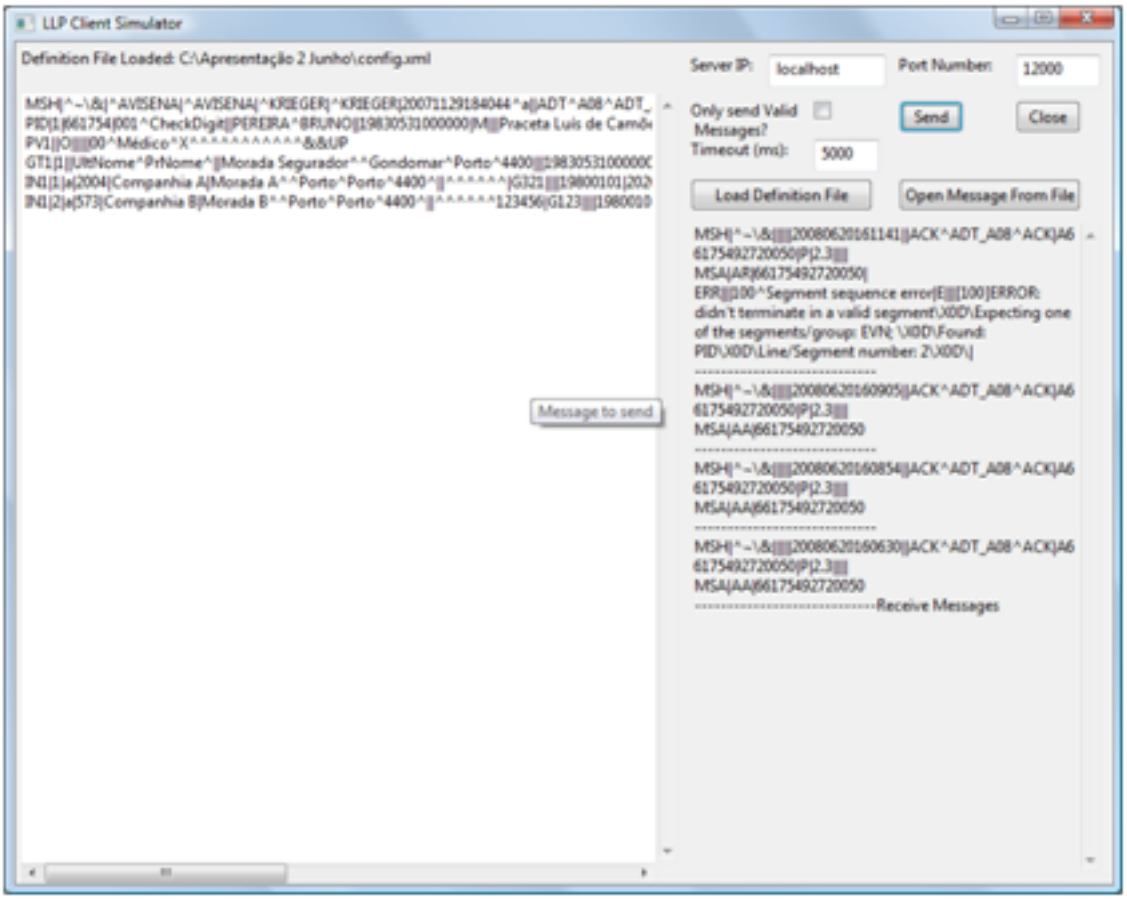

Figura 7.15: Cliente LLP ACK com erro recebido

Capítulo 8

Testes

Das várias aplicações descritas neste relatório a mais relevante é sem dúvida o motor de HL7 desenvolvido, ou seja, o projecto denominado neste relatório como *HL7 Parser*. As restantes são auxiliares ao desenvolvimento e implementação de interfaces pelas equipas da empresa ALERT®, ou são complementares como o caso da aplicação servidor LLP.

A aplicação servidor LLP desenvolvida foi aproveitada e desenvolvida tendo em vista o seu funcionamento como o receptor das comunicações recebidas por TCP/IP. Como este já se encontra desenvolvido os testes de performance realizados ao *HL7 Parser* passam pela recepção de mensagens através deste socket, que posteriormente valida e interpreta a mensagem e envia o ACK para o cliente. A aplicação cliente LLP foi utilizada para alimentar o sistema testado.

Ao contrário dos testes de performance os testes unitários realizados e aqui exibidos apenas se focam no projecto *HL7 Parser*.

8.1 Testes Unitários

Listagem dos testes unitários realizados:

1. Validação de mensagens
    (a) Mensagens válidas
    (b) Mensagens inválidas
        i. Falta de Segmentos
        ii. Terminar num segmento inválido
        iii. Segmentos incompletos, falta de campos
        iv. Tipos de dados errados

2. Alterar uma mensagem previamente validada
    (a) Alterar um segmento
    (b) Acrescentar um segmento
    (c) Alterar um grupo
        i. Alterar todo o conteúdo de um grupo
        ii. Acrescentar um grupo a um outro grupo
        iii. Alterar um tipo de dados, ver mensagem a falhar
3. Criar uma mensagem de raiz e validá-la
4. Testar os métodos que permitem no futuro efectuar a fragmentação de mensagens
    (a) Saber se uma mensagem esta fragmentada
        i. Mensagem não fragmentada
        ii. Início de uma mensagem fragmentada e obter o código apontador de continuação de mensagem
    (b) Obter o apontador de continuação de mensagem caso exista no segmento MSH
5. Testar o mecanismo de apagar segmentos que não são necessários

O código desenvolvido para a realização destes testes unitários encontram-se em anexo e o resultado pode-se consultar na figura 8.1, onde é possível verificar que todos os testes unitários passaram.

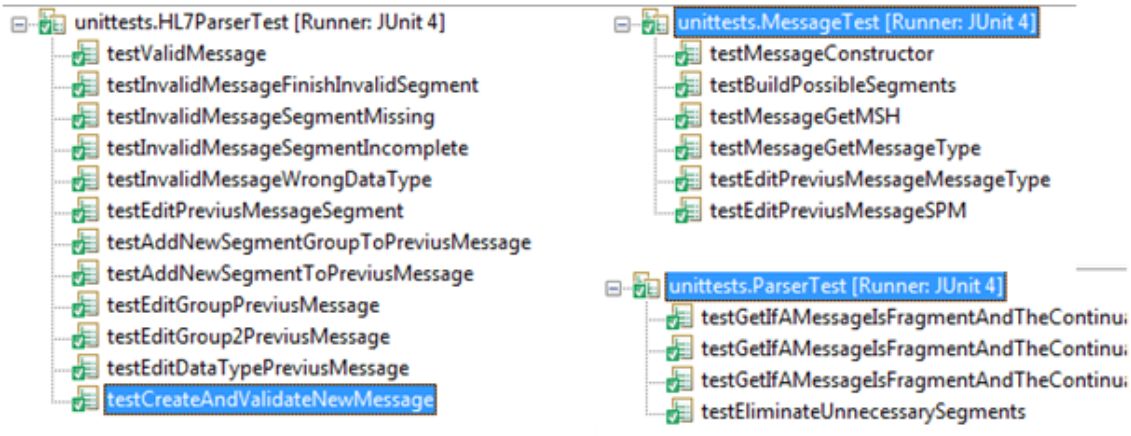

Figura 8.1: Resultado dos testes unitários

8.2 Testes de Performance

Máquina utilizada para os testes: **Sistema Operativo:** Windows Server 2003 R2 64 bits **Memória:** 16GB **Processador:** Intel® XEON® CPU E5405 @ 2.00Ghz

A máquina foi alimentada localmente, ou seja o servidor e os clientes estão a correr na mesma máquina. Desta forma os atrasos da comunicação são minimizados, mas tem o inconveniente de utilizar capacidade de processador, memória e acesso a disco da máquina que está a correr o processo a testar.

Numa única instância da JVM foram arrancados N sockets, ou seja temos N servidores à escuta de mensagens em diferentes portos.

Os ficheiros de definição de mensagens utilizados representam as definições em execução em algumas instituições. Ou seja, são ficheiros que representam a real necessidade da ALERT®

8.2.1 Teste 1

Parâmetros
Número de instâncias de servidor: 1
Número de sockets por instância: 8
Número de aplicações que alimentam um socket: 10
Memória JVM: 128M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 10m15s
Número de mensagens: 82939
Performance geral (mensagens / segundo): 134,8
Performance por socket (mensagens / segundo): 16,8

8.2.2 Teste 2

Parâmetros
Número de instâncias de servidor: 1
Número de sockets por instância: 8
Número de aplicações que alimentam um socket: 20
Memória JVM: 128M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 9m53s
Número de mensagens: 93474
Performance geral (mensagens / segundo): 157,5
Performance por socket (mensagens / segundo): 19,7

8.2.3 Teste 3

Parâmetros
Número de instâncias de servidor: 2
Número de sockets por instância: 4
Número de aplicações que alimentam um socket: 10
Memória JVM: 64M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 13m48s
Número de mensagens: 139139
Performance geral (mensagens / segundo): 168
Performance por socket (mensagens / segundo): 21

8.2.4 Teste 4

Parâmetros
Número de instâncias de servidor: 2
Número de sockets por instância: 4
Número de aplicações que alimentam um socket: 20
Memória JVM: 64M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 19m40s
Número de mensagens: 196688
Performance geral (mensagens / segundo): 166,6
Performance por socket (mensagens / segundo): 20,8

8.2.5 Teste 5

Parâmetros
Número de instâncias de servidor: 1
Número de sockets por instância: 8
Número de aplicações que alimentam um socket: 10
Memória JVM: 256M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 12m40s
Número de mensagens: 99806
Performance geral (mensagens / segundo): 131,2
Performance por socket (mensagens / segundo): 16,4

8.2.6 Teste 6

Parâmetros
Número de instâncias de servidor: 1
Número de sockets por instância: 8
Número de aplicações que alimentam um socket: 20
Memória JVM: 256M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 14m25s
Número de mensagens: 116826
Performance geral (mensagens / segundo): 135
Performance por socket (mensagens / segundo): 16,9

8.2.7 Teste 7

Parâmetros
Número de instâncias de servidor: 2
Número de sockets por instância: 4
Número de aplicações que alimentam um socket: 10
Memória JVM: 128M

Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 17m45s
Número de mensagens: 317186
Performance geral (mensagens / segundo): 297,7
Performance por socket (mensagens / segundo): 37,2

8.2.8 Teste 8

Parâmetros
Número de instâncias de servidor: 2
Número de sockets por instância: 4
Número de aplicações que alimentam um socket: 20
Memória JVM: 128M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 13m38s
Número de mensagens: 249447
Performance geral (mensagens / segundo): 304,8
Performance por socket (mensagens / segundo): 38,1

8.2.9 Teste 9

Parâmetros
Número de instâncias de servidor: 1
Número de sockets por instância: 8
Número de aplicações que alimentam um socket: 10
Memória JVM: 128M
Número máximo de Threads activos: 10

Resultados
Tempo: 36m47s
Número de mensagens: 149146
Performance geral (mensagens / segundo): 67,5
Performance por socket (mensagens / segundo): 8,4

8.2.10 Teste 10

Parâmetros
Número de instâncias de servidor: 1
Número de sockets por instância: 8
Número de aplicações que alimentam um socket: 10
Memória JVM: 128M
Número máximo de Threads activos: 20

Resultados
Tempo: 13m10s
Número de mensagens: 104913
Performance geral (mensagens / segundo): 132,7
Performance por socket (mensagens / segundo): 16,6

8.2.11 Teste 11

Parâmetros
Número de instâncias de servidor: 4
Número de sockets por instância: 2
Número de aplicações que alimentam um socket: 10
Memória JVM: 128M
Número máximo de Threads activos: 1000 (tentativa de simular sem limite)

Resultados
Tempo: 9m49s
Número de mensagens: 343191
Performance geral (mensagens / segundo): 582,4
Performance por socket (mensagens / segundo): 72,8

8.2.12 Teste 12

Parâmetros
Número de instâncias de servidor: 4
Número de sockets por instância: 2
Número de aplicações que alimentam um socket: 20
Memória JVM: 128M

Número máximo de Threads activos: 20

Resultados
Tempo: 9m10s
Número de mensagens: 340891
Performance geral (mensagens / segundo): 619
Performance por socket (mensagens / segundo): 77,4

8.3 Conclusão dos testes de performance

Como é possível analisar nos valores apresentados para cada teste, existem alguns factores que influenciam a performance deste sistema.

Manter vários canais numa única JVM prejudica a performance de cada canal, mas a memória consumida é consideravelmente inferior a manter todos os sockets em JVM separadas. É necessário encontrar um equilíbrio entre o número de JVM a utilizar e o número de canais por cada uma.

A limitação do número de comunicações activas, no caso de ser um valor muito baixo, em relação ao número de mensagens por segundo que recebemos, não consegue dar resposta imediata às mensagens recebidas e como tal o desempenho é penalizado. Se por outro lado for definido um limite máximo de comunicações activas superiores ao que a máquina consegue suportar, e esse máximo suportado for atingido, a performance também será penalizada, uma vez que a máquina irá passar a maior parte do tempo a tentar limpar a memória da JVM. Como é possível ver pelos testes realizados, encontrar um equilíbrio para estes valores não será complicado, uma vez que a memória consumida não é excessiva e é proporcional ao número de comunicações activas.

A atribuição de memória a mais à JVM não traz mais-valias para a performance, uma vez que essa memória possivelmente não será utilizada. É possível monitorizar a memória que está a ser utilizada pela JVM se arrancarmos a mesma com a opção -Xloggc:log.log onde a informação da memória antes e depois de realizado o Garbage Collection será guardado no ficheiro log.log. A memória foi monitorizada durante os testes e verificou-se que se encontra estável e sem fugas. A memória apenas aumenta ligeiramente quando o número de comunicações activas aumenta, ou seja está a processar mais mensagens em paralelo.

Como segurança para a JVM não atingir o máximo de memória possível é possível configurar a aplicação servidora para não aceitar mais que N comunicações simultâneas.

Capítulo 9

Conclusões

9.1 Síntese do trabalho desenvolvido

O projecto desenvolvido centra-se na validação e interpretação de mensagens HL7 para permitir a troca de informação clínica entre a aplicação ALERT® e outros sistemas de informação.

O projecto desenvolvido teve ir de encontro às necessidades levantadas pela ALERT® e sem os problemas existentes noutras aplicações analisadas. A performance e a fiabilidade foram outros aspectos considerados para o desenvolvimento deste motor HL7.

Durante o desenvolvimento deste projecto foram identificadas funcionalidades que trariam valor para este projecto e uma mais-valia para a análise de mensagens HL7 por um utilizador. Estas funcionalidades foram desenvolvidas como módulos independentes do motor HL7 para não carregar, nem prejudicar a performance do mesmo.

Para ser possível ilustrar um ciclo completo de troca de mensagens HL7 foi também desenvolvido um cliente e servidor para a troca de mensagens. Estas aplicações permitem executar todos os passos de um ciclo de mensagens: envio de uma mensagem (cliente), recepção (servidor), validação e interpretação (servidor), construção e envio do ACK (servidor), recepção do ACK (cliente).

Os objectivos propostos para este projecto foram todos atingidos, e no entender do aluno, superados. O trabalho desenvolvido cumpre todas as especificações feitas e foram realizadas ainda algumas ferramentas extra. Uma delas permite uma fácil definição da estrutura de mensagens e permitindo uma outra que o utilizador realize uma fácil e rápida validação e interpretação de mensagens.

Além disso efectuou-se o desenvolvimento de um cliente e servidor que comunicam através do protocolo LLP sobre TCP/IP, em que a aplicação servidora irá ser incorporada na próxima versão do INTER-ALERT®, tal como o *HL7 Parser*.

9.2 Trabalho futuro

Com a entrada deste projecto em produção existe a necessidade de acrescentar novas funcionalidades. Estas novas funcionalidades centram-se no desenvolvimento de novos módulos que vão permitir a monitorização e configuração da aplicação de uma forma mais produtiva e fácil. Os desenvolvimentos previstos são:

- Disponibilizar uma interface gráfica para o controlo e configuração de vários sockets numa única instância da JVM, com a finalidade de poupar alguma memória;
- Disponibilizar uma interface gráfica para visualização do conteúdo do ficheiro de log;
- Disponibilizar um método que permita guardar e retirar as mensagens que chegaram a um determinado socket;
- Enviar alertas quando surge um erro numa mensagem recebida, por exemplo através do e-mail;

Referências

[Apa] Apache. Ant. Disponível em http://ant.apache.org/, acedido a última vez em 30 de Junho de 2008.

[Com] SWT Community. The standard widget toolkit. Disponível em http://www.eclipse.org/swt/, acedido a última vez em 30 de Junho de 2008.

[Des] Desconhecido. Javascript api. Disponível em http://krook.org/jsdom/, acedido a última vez em 30 de Junho de 2008.

[Foua] Apache Software Foundation. Log4j, logging services. Disponível em http://logging.apache.org/log4j/1.2/index.html, acedido a última vez em 30 de Junho de 2008.

[Foub] Apache Software Foundation. Struts. Disponível em http://struts.apache.org/index.html, acedido a última vez em 30 de Junho de 2008.

[Fouc] Apache Software Foundation. Struts 2. Disponível em http://struts.apache.org/2.0.11.1/index.html, acedido a última vez em 30 de Junho de 2008.

[Fou01] Eclipse Foundation. Eclipse ide, 2001. Disponível em http://www.eclipse.org/, acedido a última vez em 30 de Junho de 2008.

[Fuc] Thomas Fuchs. script.aculo.us. Disponível em http://script.aculo.us/, acedido a última vez em 30 de Junho de 2008.

[Har01] Elliotte Rusty Harold. Processing xml with java, 2001. Disponível em http://www.cafeconleche.org/books/xmljava/, acedido a última vez em 30 de Junho de 2008.

[HLS] Inc. Health Level Seven. Health level seven. Disponível em http://www.hl7.org/, acedido a última vez em 30 de Junho de 2008.

[HM] Jason Hunter and Brett McLaughlin. Jdom. Disponível em http://www.jdom.org/, acedido a última vez em 30 de Junho de 2008.

[Ind] Rose India. Struts 2 tutoriais. Disponível em http://www.roseindia.net/struts/struts2/index.shtml, acedido a última vez em 30 de Junho de 2008.

[SUN] SUN. Java developer. Disponível em http://java.sun.com/, acedido a última vez em 30 de Junho de 2008.

[VV06] Ajay Vohra and Deepak Vohra. *Pro XML Development with Java Technology*. Apress, 2006.

[W3C] W3C. Document object model. Disponível em http://www.w3.org/DOM/, acedido a última vez em 30 de Junho de 2008.

[W3C00] W3C. Xml schema, 2000. Disponível em http://www.w3.org/XML/Schema, acedido a última vez em 30 de Junho de 2008.

Anexo A

Código dos testes unitários

Os testes unitários foram agrupados consoante o tipo de teste executado. O conteúdo desses ficheiros encontra-se transcrito a seguir.

HL7ParserTest.java

```
1  package unittests;
2
3  import java.util.ArrayList;
4  import junit.framework.TestCase;
5  /**
6   * Unit tests about message validation
7   *
8   * @author Bruno Pereira
9   *
10  */
11 public class HL7ParserTest extends TestCase{
12      public static int MESSAGEOK = 1;
13
14      public static int MESSAGESEGMENTERROR = 0;
15
16      public static int MESSAGECONTENTERROR = -1;
17
18      public static void main(String[] args){
19          junit.textui.TestRunner.run(MessageTest.class);
20      }
21
22      /**
23       * Test a Valid message with a ADD segment Test id: 1.a
24       *
25       */
26      public void testValidMessage(){
27          String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
28                  + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||||\r"
29                  + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||||1645675^^^GH
                        |||||||||||||||||||||||||||||||\r"
30                  + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
31                  + "OBR|\r"
32                  + "ADD|1|\r"
33                  + "ADD|comp1^comp2||a|\r"
```

```
34                  + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
35                  + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
36                  + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
37                  + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
38          Parser val = new Parser(msg);
39          assertEquals(MESSAGEOK, val.validate());
40      }
41
42      /**
43       * Test a invalid message Required Segment missing (SPM) Test id: 1.b.i
44       *
45       */
46      public void testInvalidMessageFinishInvalidSegment(){
47          String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
48                  + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||\r"
49                  + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||16456675^^^GH
                        |||||||||||||||||||||||||||||\r"
50                  + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r";
51          Parser val = new Parser(msg);
52          assertEquals(MESSAGESEGMENTERROR, val.validate());
53      }
54
55      /**
56       * Test a invalid message Finish in a invalid Segment Test id: 1.b.ii
57       *
58       */
59      public void testInvalidMessageSegmentMissing(){
60          String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
61                  + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||\r"
62                  + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||16456675^^^GH
                        |||||||||||||||||||||||||||||\r"
63                  + "OBR|1|comp1^comp2||\r"
64                  + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
65                  + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
66                  + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
67                  + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
68          Parser val = new Parser(msg);
69          assertEquals(MESSAGESEGMENTERROR, val.validate());
70      }
```

```
71
72      /**
73       * Test a invalid message Segment incomplete, requried field missing (OBR
             without field 4) Test id: 1.b.iii
74       *
75       */
76      public void testInvalidMessageSegmentIncomplete(){
77          String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
78                  + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||||\r"
79                  + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||||1645675^^^GH
                        |||||||||||||||||||||||||||||||\r"
80                  + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
81                  + "OBR|1|comp1^comp2|\r"
82                  + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
83                  + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
84                  + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
85                  + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
86          Parser val = new Parser(msg);
87          assertEquals(MESSAGECONTENTERROR, val.validate());
88      }
89
90      /**
91       * Test a invalid message Worng data type (Set ID - OBR - type SI) Test id:
             1.b.iv Este teste assume que a validação
92       * de tipos esta activa
93       *
94       */
95      public void testInvalidMessageWrongDataType(){
96          String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
97                  + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||||\r"
98                  + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||||1645675^^^GH
                        |||||||||||||||||||||||||||||||\r"
99                  + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
100                 + "OBR|1|comp1^comp2||\r"
101                 + "OBR|wrongType|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                        Sangue␣Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
102                 + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
103                 + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
104                 + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
```

```
105         Parser val = new Parser(msg);
106         assertEquals(MESSAGECONTENTERROR, val.validate());
107     }
108 
109     /**
110      * Test edit a message Edit one segment from a previus valid message Test
             id: 2.a
111      *
112      */
113     public void testEditPreviusMessageSegment(){
114         String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
115                 + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||\r"
116                 + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||16456 75^^^GH
                        |||||||||||||||||||||||||||||\r"
117                 + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
118                 + "OBR|1|comp1^comp2||a|\r"
119                 + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||\r"
120                 + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||\r"
121                 + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
122                 + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
123         Parser val = new Parser(msg);
124         assertEquals(MESSAGEOK, val.validate());
125         Message validMessage = val.getCorrectMsg();
126 
127         OUL_R22 oulr22 = new OUL_R22();
128         oulr22.setMessage(validMessage);
129         SPM spm = oulr22.getSPM();
130         spm.setSet_ID___SPM(new SI("5"));
131         EIP eip = new EIP();
132         EI ei = new EI(), ei2 = new EI();
133         ei.setEntity_Identifier(new ST("Placer"));
134         ei.setNamespace_ID(new IS("NameSpaceNameSpaceNa"));
135         eip.setPlacer_Assigned_Identifier(ei);
136         ei2.setEntity_Identifier(new ST("Filler"));
137         eip.setFiller_Assigned_Identifier(ei2);
138         spm.setSpecimen_ID(eip);
139         oulr22.setSPM(spm);
140         assertEquals(MESSAGEOK, oulr22.validate());
141         assertEquals(
142                 "SPM|5|Placer&NameSpaceNameSpaceNa^Filler||^SANGUE␣TOTAL^
                        CLINIDATAXXI||||||||||||20080310115548|",
143                 oulr22.getSPM().toString(oulr22.getDelimiters()));
144     }
145 
146     /**
147      * Test edit a message Add one segment to a previus valid message, in this
             case one group Test id: 2.b
148      *
149      */
```

```
150     public void testAddNewSegmentGroupToPreviusMessage(){
151         String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
152                 + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||||\r"
153                 + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||1645675^^^GH
                        |||||||||||||||||||||||||||||||\r"
154                 + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
155                 + "OBR|1|comp1^comp2||a|\r"
156                 + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
157                 + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
158                 + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
159                 + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
160         Parser val = new Parser(msg);
161         assertEquals(MESSAGEOK, val.validate());
162         Message temp = val.getCorrectMsg();
163 
164         OUL_R22 oulr22 = new OUL_R22();
165         oulr22.setMessage(temp);
166         OBR obr = new OBR();
167         obr.setSet_ID___OBR(new SI("4"));
168         EI ei = new EI(), ei2 = new EI();
169         ei.setEntity_Identifier(new ST("Placer"));
170         obr.setPlacer_Order_Number(ei);
171         ei2.setEntity_Identifier(new ST("Filler"));
172         obr.setFiller_Order_Number(ei2);
173         OUL_R22ORDER oulOrder = new OUL_R22ORDER();
174         ArrayList<OUL_R22ORDER> list = oulr22.getOUL_R22ORDERList();
175         oulOrder.setOBR(obr);
176         list.add(oulOrder);
177         oulr22.setOUL_R22ORDERList(list);
178         oulr22.update();
179         assertEquals(MESSAGEOK, oulr22.validate());
180         assertEquals("OBR|4|Placer|Filler||", oulr22
181                 .getOUL_R22ORDERList().get(3).getOBR().toString(oulr22.
                        getDelimiters()));
182     }
183 
184     /**
185      * Test edit a message Add one segment to a previus valid message Test id:
                2.b
186      *
187      */
188     public void testAddNewSegmentToPreviusMessage(){
189         String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
190                 + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||||\r"
191                 + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||1645675^^^GH
                        |||||||||||||||||||||||||||||||\r"
```

```
192                 + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
193                 + "OBR|1|\\R\\comp1^comp2||a|\r"
194                 + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
195                 + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB|||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
196                 + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
197                 + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
198         Parser val = new Parser(msg);
199         assertEquals(MESSAGEOK, val.validate());
200         Message temp = val.getCorrectMsg();
201
202         OUL_R22 oulr22 = new OUL_R22();
203         oulr22.setMessage(temp);
204         ORC orc = new ORC();
205         orc.setOrder_Control(new ID("1"));
206
207         EI ei = new EI(), ei2 = new EI();
208         ei.setEntity_Identifier(new ST("Placer"));
209         ei2.setEntity_Identifier(new ST("Filler"));
210         orc.setPlacer_Order_Number(ei);
211         orc.setFiller_Order_Number(ei2);
212         OUL_R22ORDER order = oulr22.getOUL_R22ORDERList().get(0);
213         order.setORC(orc);
214         order.update();
215         oulr22.getOUL_R22ORDERList().set(0, order);
216         oulr22.setOUL_R22ORDERList(oulr22.getOUL_R22ORDERList());
217         oulr22.update();
218         assertEquals(MESSAGEOK, oulr22.validate());
219         assertEquals("ORC|1|Placer|Filler|", oulr22.getOUL_R22ORDERList().get
                (0)
220                 .getORC().toString(oulr22.getMessage().getDelimiters()));
221         assertEquals("~comp1", oulr22.getOUL_R22ORDERList().get(0).getOBR().
                getPlacer_Order_Number()
222                 .getEntity_Identifier().getContent());
223     }
224
225     /**
226      * Test edit a message Edit a group (replace all segments inside one) Test
                id: 2.c.i
227      *
228      */
229     public void testEditGroupPreviusMessage(){
230         String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
                R22^OUL_R22|1967651|P|2.5|1|||||||\r"
231                 + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                        ||19640602000000|M|||||||||19640600552^^^HUC
                        |||||||||||||||||||\r"
232                 + "PV1|1|URGENCIAS|URGENCIA||||||1251||||||||||1645675^^^GH
                        |||||||||||||||||||||||||||||\r"
233                 + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                        ||||||||||||20080310115548\r"
234                 + "OBR|1|comp1^comp2||a|\r"
```

```
235                 + "OBR|2|Test^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                        Sangue␣Total)^LAB||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
236                 + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB||||||||||||||||||20080310115544|||F
                        |||||||||||||||||||||\r"
237                 + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        |||||||||||||||||||\r"
238                 + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
239         Parser val = new Parser(msg);
240         assertEquals(MESSAGEOK, val.validate());
241         Message temp = val.getCorrectMsg();
242
243         OUL_R22 oulr22 = new OUL_R22();
244         oulr22.setMessage(temp);
245         ORC orc = new ORC();
246         orc.setOrder_Control(new ID("1"));
247
248         EI ei = new EI(), ei2 = new EI();
249         ei.setEntity_Identifier(new ST("Placer"));
250         ei2.setEntity_Identifier(new ST("Filler"));
251         orc.setPlacer_Order_Number(ei);
252         orc.setFiller_Order_Number(ei2);
253         OBR obr = new OBR();
254         obr.setSet_ID___OBR(new SI("123"));
255         obr.setPlacer_Order_Number(ei);
256         obr.setFiller_Order_Number(ei2);
257         OUL_R22ORDER oulOrder = new OUL_R22ORDER();
258         ArrayList<OUL_R22ORDER> list = new ArrayList<OUL_R22ORDER>();
259         oulOrder.setOBR(obr);
260         oulOrder.setORC(orc);
261         list.add(oulOrder);
262         oulr22.setOUL_R22ORDERList(list);
263         oulr22.update();
264         assertEquals(MESSAGEOK, oulr22.validate());
265         assertEquals("OBR|123|Placer|Filler||", oulr22
266                 .getOUL_R22ORDERList().get(0).getOBR().toString(oulr22.
                        getDelimiters()));
267
268     }
269
270     /**
271      * Test edit a message Edit a group 2(add new group to a group) Test id: 2.
                c.ii
272      *
273      */
274     public void testEditGroup2PreviusMessage(){
275         String msg = "MSH|^~\\&|^AVISENA|^AVISENA|^ALERT|^ALERT
                |20070314115247||SIU^S12|37822943202007|P|2.3\r"
276                 + "SCH|3782294|3782294|1|AVISENA||VEIN␣TX^VEIN␣TX|VEIN␣TX^VEIN␣
                        TX|VEIN␣TX^VEIN␣TX
                        |||^^^20070320113000^20070320120000|||||2999^GONZALEZ^MARIA
                        |7866214400|||2999^GONZALEZ^MARIA|||||2000^NEW\r"
277                 + "PID|1||1240333|000103||RODRIGUEZ^JUANA||19771230000000|F
                        |||100␣SW␣23RD␣STREET^^MIAMI^FL^33125||3055558570|||D
                        ||000103|999-99-9999\r"
278                 + "PV1|1|O|^^^&2744^^^^^Practices␣Inst||||919^Jones^Indiana
                        ^^^^^^^^^^&&UPIN\r" + "RGS|1\r"
```

```
279                 + "AIL|||2744\r" + "AIP|1|||919|^PROVIDER\r";
280         Parser val = new Parser(msg);
281         assertEquals(MESSAGEOK, val.validate());
282         Message temp = val.getCorrectMsg();
283 
284         SIU_S12 sius12 = new SIU_S12();
285         sius12.setMessage(temp);
286         AIG aig = new AIG();
287         CE ce = new CE();
288         SIU_S12RESOURCES resource = sius12.getSIU_S12RESOURCESList().get(0);
289 
290         aig.setSet_ID___AIG(new SI("123"));
291         ce.setText(new ST("Test␣CE␣in␣General␣Resorce␣group"));
292         aig.setResource_Type(ce);
293         SIU_S12GENERAL_RESOURCE general = new SIU_S12GENERAL_RESOURCE();
294         general.setAIG(aig);
295         ArrayList<SIU_S12GENERAL_RESOURCE> listGeneral = new ArrayList<
              SIU_S12GENERAL_RESOURCE>();
296         listGeneral.add(general);
297         resource.setSIU_S12GENERAL_RESOURCEList(listGeneral);
298         resource.update();
299         ArrayList<SIU_S12RESOURCES> listResources = sius12.
              getSIU_S12RESOURCESList();
300         listResources.set(0, resource);
301         sius12.setSIU_S12RESOURCESList(listResources);
302         sius12.update();
303 
304         // after add one new group to a group check if everyting is OK
305         assertEquals(MESSAGEOK, sius12.validate());
306         assertEquals("AIG|123|||^Test␣CE␣in␣General␣Resorce␣group|", sius12.
              getSIU_S12RESOURCESList()
307                 .get(0).getSIU_S12GENERAL_RESOURCEList().get(0).getAIG().
                      toString(sius12.getDelimiters()));
308         assertEquals("AIL|||2744|", sius12.getSIU_S12RESOURCESList().get(0)
309                 .getSIU_S12LOCATION_RESOURCEList().get(0).getAIL().toString(
                      sius12.getDelimiters()));
310     }
311 
312     /**
313      * Test edit a message Edit one data type, message should be invalid (SI to
              String) Test id: 2.d Este teste assume
314      * que a validação de tipos esta activa
315      *
316      */
317     public void testEditDataTypePreviusMessage(){
318         String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC|20080310115548||OUL^
              R22^OUL_R22|1967651|P|2.5|1|||||||\r"
319                 + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                      ||19640602000000|M|||||||||19640600552^^^HUC
                      |||||||||||||||||||\r"
320                 + "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||1645675^^^GH
                      |||||||||||||||||||||||||||||\r"
321                 + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                      ||||||||||||20080310115548\r"
322                 + "OBR|1|comp1^comp2||a|\r"
323                 + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                      Total)^LAB|||||||||||||||||20080310115544|||F
                      |||||||||||||||||||\r"
```

```
324                 + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB||||||||||||||||||20080310115544|||F
                        ||||||||||||||||||||||\r"
325                 + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                        ||||||||||||||||||||\r"
326                 + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L
                        |0.50␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(
                        Médico)~Enfermeira|||\r";
327         Parser val = new Parser(msg);
328         assertEquals(MESSAGEOK, val.validate());
329         Message temp = val.getCorrectMsg();
330
331         OUL_R22 oulr22 = new OUL_R22();
332         oulr22.setMessage(temp);
333         SPM spm = oulr22.getSPM();
334         spm.setSet_ID___SPM(new SI("OI"));
335
336         assertEquals(MESSAGECONTENTERROR, oulr22.validate());
337     }
338
339     /**
340      * Test Build a message from root
341      * Build and validate a message
342      * Test id: 3
343      */
344     public void testCreateAndValidateNewMessage(){
345         if (!RepositoryList.getSingleTonInstance().loadNewFile("default2", "
                config.xml")) {
346             System.out.println("Error␣during␣load␣definition␣file");
347             return;
348         }
349         SIU_S12 sius12 = BuildMessage.GenerateSIU_S12();
350         sius12.update();
351         String text = "MSH|^~\\&|||||||SIU^S12^SIU_S12||||||OK|\r"
352             + "SCH||||||||||||||||||||^^^^^^^^^^^^^^^^^^&Universal_ID~^Salgueiro&&
                    Meireles||||||\r"
353             + "PID||^check␣Digit^^&Universal_ID||||||||||||||||||||||||CE␣1~CE␣
                    2|\r"
354             + "RGS||rgs␣algo|\r"
355             + "AIS|123|||\r"
356             + "AIL||||||||||||yes|\r";
357         assertEquals(text, sius12.toString());
358
359         sius12.update();
360         assertEquals(MESSAGEOK, sius12.validate());
361         assertEquals(text, sius12.toString());
362     }
363 }
```

MessageTest.java

```
1 package unittests;
2 import java.util.ArrayList;
3 import junit.framework.TestCase;
4 /**
5  * Units test about the generated objects and the mapping of this objects with
        the object Message
6  *
7  * @author Bruno Pereira
8  *
9  */
```

```
10
11 public class MessageTest extends TestCase{
12
13         private String[] delimiters = {"\r", "|", "^", "&", "~", "\\" };
14
15     /**
16      * Test message constructor
17      *
18      */
19         public void testMessageConstructor() {
20                 Message m = new Message("Teste");
21                 assertEquals("Teste", m.getName());
22         }
23         /**
24      * Test if
25      *
26          */
27         public void testBuildPossibleSegments() {
28                 Message m = new Message("Teste");
29                 SegmentDefinition segment = new SegmentDefinition("YYY");
30                 m.addStructure(new Structure("YYY", segment, 0, "unbounded"));
31                 ArrayList<Segment2> test = new ArrayList<Segment2>();
32                 test.add(new Segment2("YYY"));
33                 assertEquals(test.size(), m.buildPossibleSegments().size());
34         }
35         /**
36          * Test the get the MSH of the message, the return is equivalent to the
                1st segment of the message
37      *
38          */
39         public void testMessageGetMSH() {
40                 String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                    |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|||||||||\r"
                     +
41                         "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                            ||19640602000000|M|||||||||19640600552^^^HUC
                            |||||||||||||||||||||\r" +
42                         "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||||1645675^^^
                            GH|||||||||||||||||||||||||||||||||||\r" +
43                         "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                            ||||||||||||20080310115548\r" +
44                         "OBR|\r" +
45                         "ADD|1|\r" +
46                         "ADD|comp1^comp2||a|\r" +
47                         "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                            Sangue␣Total)^LAB
                            |||||||||||||||||20080310115544|||F
                            |||||||||||||||||||||\r" +
48                         "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                            Sangue␣Total)^LAB
                            |||||||||||||||||20080310115544|||F
                            |||||||||||||||||||||\r" +
49                         "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                            |||||||||||||||||||\r" +
50                         "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|
                            mmol/L|0.50␣-␣2.00|H|||F|||20080310115308||Filomena
                            ␣Martinho(Médico)~Enfermeira|||\r"
51                         ;
52                 Parser val = new Parser(msg);
53                 assertEquals(HL7ParserTest.MESSAGEOK, val.validate());
```

```
54
55                  Message temp = val.getCorrectMsg();
56
57                  OUL_R22 oulr22 = new OUL_R22();
58                  oulr22.setMessage(temp);
59
60                  assertEquals("MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                        |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|", oulr22.
                        getMSH().toString(delimiters));
61          }
62          /**
63           * Test the get of the message Type field
64           *
65           */
66          public void testMessageGetMessageType() {
67                  String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                        |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|||||||||\r"
                         +
68                          "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                            ||19640602000000|M|||||||||19640600552^^^HUC
                            |||||||||||||||||||||\r" +
69                          "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||||1645675^^^
                            GH|||||||||||||||||||||||||||||||||\r" +
70                          "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                            ||||||||||||20080310115548\r" +
71                          "OBR|\r" +
72                          "ADD|1|\r" +
73                          "ADD|comp1^comp2||a|\r" +
74                          "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                            Sangue␣Total)^LAB
                            |||||||||||||||||20080310115544|||F
                            |||||||||||||||||||||\r" +
75                          "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                            Sangue␣Total)^LAB
                            |||||||||||||||||20080310115544|||F
                            |||||||||||||||||||||\r" +
76                          "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                            |||||||||||||||||||\r" +
77                          "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|
                            mmol/L|0.50␣-␣2.00|H|||F|||20080310115308||Filomena
                            ␣Martinho(Médico)~Enfermeira|||\r"
78                          ;
79                  Parser val = new Parser(msg);
80                  assertEquals(HL7ParserTest.MESSAGEOK, val.validate());
81
82                  Message temp = val.getCorrectMsg();
83
84                  OUL_R22 oulr22 = new OUL_R22();
85                  oulr22.setMessage(temp);
86
87                  assertEquals("OUL^R22^OUL_R22", oulr22.getMSH().getMessage_Type
                        ().toString(delimiters, true));
88          }
89          /**
90           * Test the edit of the message type of a message (the message was not
                validate next)
91           *
92           */
93          public void testEditPreviusMessageMessageType() {
```

```
94              String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                    |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|||||||||\r"
                     +
95              "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                    ||19640602000000|M|||||||||19640600552^^^HUC
                    |||||||||||||||||||||\r" +
96              "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||16456 75^^^GH
                    |||||||||||||||||||||||||||||||||\r" +
97              "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                    ||||||||||||20080310115548\r" +
98              "OBR|\r" +
99              "ADD|1|\r" +
100             "ADD|comp1^comp2||a|\r" +
101             "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                    Total)^LAB|||||||||||||||||20080310115544|||F
                    |||||||||||||||||||\r" +
102             "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                    Total)^LAB|||||||||||||||||20080310115544|||F
                    |||||||||||||||||||\r" +
103             "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM|||||||||||||||||||\
                    r" +
104             "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L|0.50
                    ␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(Médico)~
                    Enfermeira|||\r"
105             ;
106             Parser val = new Parser(msg);
107             assertEquals(HL7ParserTest.MESSAGEOK, val.validate());
108             Message temp = val.getCorrectMsg();
109
110             OUL_R22 oulr22 = new OUL_R22();
111             oulr22.setMessage(temp);
112             MSH msh = oulr22.getMSH();
113             MSG msgfield = msh.getMessage_Type();
114             msgfield.setMessage_Structure(new ID("Test"));
115             assertEquals("OUL^R22^Test", oulr22.getMSH().getMessage_Type().
                    toString(delimiters, true));
116     }
117     /**
118      * Test editing the SPM segment and validate the message next
119      *
120      */
121     public void testEditPreviusMessageSPM() {
122             String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                    |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|||||||||\r"
                     +
123             "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                    ||19640602000000|M|||||||||19640600552^^^HUC
                    |||||||||||||||||||||\r" +
124             "PV1|1|URGENCIAS|URGENCIA||||||1251|||||||||16456 75^^^GH
                    |||||||||||||||||||||||||||||||||\r" +
125             "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                    ||||||||||||20080310115548\r" +
126             "OBR|\r" +
127             "ADD|1|\r" +
128             "ADD|comp1^comp2||a|\r" +
129             "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                    Total)^LAB|||||||||||||||||20080310115544|||F
                    |||||||||||||||||||\r" +
```

```
130                 "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(Sangue␣
                        Total)^LAB||||||||||||||||||20080310115544|||F
                        ||||||||||||||||||||||\r" +
131                 "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM||||||||||||||||||\
                        r" +
132                 "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|mmol/L|0.50
                        ␣-␣2.00|H|||F|||20080310115308||Filomena␣Martinho(Médico)~
                        Enfermeira|||\r"
133                 ;
134                 Parser val = new Parser(msg);
135
136                 assertEquals(HL7ParserTest.MESSAGEOK, val.validate());
137                 Message temp = val.getCorrectMsg();
138
139                 OUL_R22 oulr22 = new OUL_R22();
140                 oulr22.setMessage(temp);
141                 SPM spm = new SPM();
142                 spm.setSet_ID___SPM(new SI("69"));
143                 EIP eip = new EIP();
144                 EI ei = new EI(),
145                         ei2 = new EI();
146                 ei.setEntity_Identifier(new ST("Placer"));
147                 eip.setPlacer_Assigned_Identifier(ei);
148                 ei2.setEntity_Identifier(new ST("Filler"));
149                 eip.setFiller_Assigned_Identifier(ei2);
150                 spm.setSpecimen_ID(eip);
151                 oulr22.setSPM(spm);
152                 oulr22.update();
153                 assertEquals(1, oulr22.getMessage().validateMessage());
154                 assertEquals("SPM|69|Placer^Filler|", oulr22.getSPM().toString(
                        delimiters));
155         }
156
157         public static void main(String[] args) {
158         junit.textui.TestRunner.run(
159                         MessageTest.class);
160     }
161 }
```

ParserTest.java

```
1  package unittests;
2
3  import alsc.interalert.hl7parser.parser.validate.Parser;
4  import junit.framework.TestCase;
5
6  import alsc.interalert.hl7parser.grammar.*;
7
8  /**
9   * Unit test about fragmented messages
10  * @author Bruno Pereira
11  *
12  */
13 public class ParserTest extends TestCase{
14
15         public static void main(String[] args) {
16         junit.textui.TestRunner.run(
17                         MessageTest.class);
18     }
19
20         /**
```

```
21          * Test if this message is fragment and if returns the correct code and
                continuation code
22          *
23          * Test id: 4.a.i
24          */
25         public void testGetIfAMessageIsFragmentAndTheContinuationCode() {
26
27                 String text = "MSH|^~\\&|||||||SIU^S12^SIU_S12||||||OK|||||||\r
                       " +
28                         "SCH||||||||||||||||||||^^^^^^^^^^^^^^^& Universal_ID
                             &^^^^^^^^^~^Salgueiro&&Meireles
                             &&^^^^^^^^^^^^^^^^^^^^^^^^^|||||||\r" +
29                         "PID||^check Digit^^&Universal_ID
                             &^^^^^^||||||||||||||||||||||||CE 1^^^^^~CE
                             2^^^^^|||||||||||||\r" +
30                         "RGS||rgs algo||\r" +
31                         "AIS|123||||||||||||\r" +
32                         "AIL||||||||||yes||\r"
33                         ;
34
35
36                 Parser val = new Parser(text);
37
38                 assertEquals(null, val.isFragmentMessage());
39         }
40
41         /**
42          * Test if this message is fragment and if returns the correct code and
                continuation code
43          *
44          * Test id: 4.a.ii
45          */
46         public void testGetIfAMessageIsFragmentAndTheContinuationCode2() {
47                 String text = "MSH|^~\\&|||||||SIU^S12^SIU_S12||||||OK|||||||\r
                       " +
48                         "SCH||||||||||||||||||||^^^^^^^^^^^^^^^& Universal_ID
                             &^^^^^^^^^~^Salgueiro&&Meireles
                             &&^^^^^^^^^^^^^^^^^^^^^^^^^|||||||\r" +
49                         "PID||^check Digit^^&Universal_ID
                             &^^^^^^||||||||||||||||||||||||CE 1^^^^^~CE
                             2^^^^^|||||||||||||\r" +
50                         "RGS||rgs algo||\r" +
51                         "AIS|123||||||||||||\r" +
52                         "AIL||||||||||yes||\r" +
53                         "DSC|key"
54                         ;
55
56                 Parser val = new Parser(text);
57
58                 assertEquals("key", val.isFragmentMessage());
59                 assertEquals(null, val.getContinuationPointer());
60         }
61
62         /**
63          * Test if this message is fragment and if returns the correct code and
                continuation code
64          *
65          * Test id: 4.b
66          */
67         public void testGetIfAMessageIsFragmentAndTheContinuationCode3() {
```

```
68              String text = "MSH|^~\\&|||||||SIU^S12^SIU_S12|||||key|OK
                    |||||||\r" +
69                      "SCH||||||||||||||||||||^^^^^^^^^^^^^^^&Universal_ID
                            &^^^^^^^^^~^Salgueiro&&Meireles
                            &&^^^^^^^^^^^^^^^^^^^^^^^|||||||\r" +
70                      "PID||^check␣Digit^^&Universal_ID
                            &^^^^^^|||||||||||||||||||||||CE␣1^^^^^~CE␣
                            2^^^^^|||||||||||||||\r" +
71                      "RGS||rgs␣algo||\r" +
72                      "AIS|123|||||||||||\r" +
73                      "AIL||||||||||yes||\r"
74                      ;
75
76              Parser val = new Parser(text);
77              assertEquals(null, val.isFragmentMessage());
78              assertEquals("key", val.getContinuationPointer());
79      }
80      /**
81       * Test if the method to delete unnecessary segments is working
             correctly
82       * Test id: 5
83       *
84       */
85      public void testEliminateUnnecessarySegments() {
86              String msg = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                    |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|||||||||\r"
87                              + "PID|1||19640600552^^^HUC^NS||Queiros^
                                    Francisco||19640602000000|M
                                    |||||||||19640600552^^^HUC
                                    |||||||||||||||||||||\r"
88                              + "PV1|1|URGENCIAS|URGENCIA
                                    ||||||1251|||||||||16456 75^^^GH
                                    |||||||||||||||||||||||||||||||\r"
89                              + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^
                                    CLINIDATAXXI|||||||||||20080310115548\r"
90                              + "OBR|1|\\R\\comp1^comp2||\r"
91                              + "ZZZ|1|try|Delete|ExtraSegment\r"
92                              + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^
                                    Lactato␣(Sangue␣Total)^LAB
                                    |||||||||||||||||20080310115544|||F
                                    |||||||||||||||||||||\r"
93                              + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^
                                    Lactato␣(Sangue␣Total)^LAB
                                    |||||||||||||||||20080310115544|||F
                                    |||||||||||||||||||||\r"
94                              + "OPS|DeleteThissegment␣plz|yep\r"
95                              + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                                    |||||||||||||||||||\r"
96                              + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB
                                    ||5.65~9.99|mmol/L|0.50␣-␣2.00|H|||F
                                    |||20080310115308||Filomena␣Martinho(Médico
                                    )~Enfermeira|||\r";
97
98              String msg2 = "MSH|^~\\&|CLINIDATAXXI|HUC|ALERT|HUC
                    |20080310115548||OUL^R22^OUL_R22|1967651|P|2.5|1|||||||||\r"
99                      + "PID|1||19640600552^^^HUC^NS||Queiros^Francisco
                            ||19640602000000|M|||||||||19640600552^^^HUC
                            |||||||||||||||||||||\r"
```

```
100                         + "PV1||1|URGENCIAS|URGENCIA
                                ||||||1251||||||||||1645675^^^GH
                                |||||||||||||||||||||||||||||||\r"
101                         + "SPM|1|3^CLINIDATAXXI||^SANGUE␣TOTAL^CLINIDATAXXI
                                ||||||||||||20080310115548\r"
102                         + "OBR|1|\\R\\comp1^comp2||\r"
103                         + "OBR|2|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                                Sangue␣Total)^LAB
                                |||||||||||||||||20080310115544|||F
                                |||||||||||||||||||||\r"
104                         + "OBR|3|^ALERT|52551#ZLAC^CLINIDATAXXI|ZLAC^Lactato␣(
                                Sangue␣Total)^LAB
                                |||||||||||||||||20080310115544|||F
                                |||||||||||||||||||||\r"
105                         + "ORC|SC|^ALERT|52551#ZLAC^CLINIDATAXXI||CM
                                |||||||||||||||||||\r"
106                         + "OBX|1|NM|132^Lactato␣(Sangue␣Total)^LAB||5.65~9.99|
                                mmol/L|0.50␣-␣2.00|H|||F|||20080310115308||Filomena
                                ␣Martinho(Médico)~Enfermeira|||\r";
107
108                 Parser val = new Parser(msg);
109                 val.validate();
110
111                 assertEquals(msg2, val.getCorrectMsg().getContentText());
112
113         }
114 }
```

BuildMessage.java

```
 1 package unittests;
 2
 3 import java.util.ArrayList;
 4 /**
 5  * Create HL7 Message from scratch
 6  *
 7  * @author Bruno Pereira
 8  *
 9  */
10 public class BuildMessage{
11
12     /**
13      * Generate a SIU_S12 message
14      * @return object SIU_S12, that's one existing message
15      */
16     public static SIU_S12 GenerateSIU_S12(){
17         SIU_S12 siu_s12 = new SIU_S12();
18         /** MSH* */
19         MSH msh = new MSH("default2");
20         msh.setField_Separator(new ST("|"));
21         msh.setEncoding_Characters(new ST("^~\\&"));
22         msh.setAccept_Acknowledgment_Type(new ID("OK"));
23         MSG msg = new MSG();
24         msg.setMessage_Code(new ID("SIU"));
25         msg.setTrigger_Event(new ID("S12"));
26         msg.setMessage_Structure(new ID("SIU_S12"));
27         msh.setMessage_Type(msg);
28         /** SCH * */
29         SCH sch = new SCH();
30         ArrayList<XCN> xcns = new ArrayList<XCN>();
31         XCN xcn1 = new XCN(), xcn2 = new XCN();
```

```
32          HD hd = new HD();
33          hd.setUniversal_ID(new ST("Universal_ID"));
34          xcn1.setAssigning_Facility(hd);
35          FN fn = new FN();
36          fn.setSurname(new ST("Salgueiro"));
37          fn.setOwn_Surname(new ST("Meireles"));
38          xcn2.setFamily_Name(fn);
39          xcns.add(xcn1);
40          xcns.add(xcn2);
41          sch.setEntered_By_PersonList(xcns);
42          /** SIU_S12.PATIENT * */
43          PID pid = new PID();
44          CX cx = new CX();
45          cx.setAssigning_Authority(hd);
46          cx.setCheck_Digit(new ST("check␣Digit"));
47          pid.setPatient_ID(cx);
48          CE ce1 = new CE(), ce2 = new CE();
49          ce1.setIdentifier(new ST("CE␣1"));
50          ce2.setIdentifier(new ST("CE␣2"));
51          ArrayList<CE> ces = new ArrayList<CE>();
52          ces.add(ce1);
53          ces.add(ce2);
54          pid.setCitizenshipList(ces);
55          SIU_S12PATIENT siupatient = new SIU_S12PATIENT();
56          siupatient.setPID(pid);
57          ArrayList<SIU_S12PATIENT> siupatients = new ArrayList<SIU_S12PATIENT>()
                ;
58          siupatients.add(siupatient);
59          /** SIU_S12.RESOURCE * */
60          SIU_S12RESOURCES siuresource = new SIU_S12RESOURCES();
61          RGS rgs = new RGS();
62          rgs.setSegment_Action_Code(new ID("rgs␣algo"));
63          siuresource.setRGS(rgs);
64          /** SIU_S12.SERVICE * */
65          SIU_S12SERVICE siuservice = new SIU_S12SERVICE();
66          AIS ais = new AIS();
67          ais.setSet_ID___AIS(new SI("123"));
68          siuservice.setAIS(ais);
69          ArrayList<SIU_S12SERVICE> siuservices = new ArrayList<SIU_S12SERVICE>()
                ;
70          siuservices.add(siuservice);
71          /** SIU_S12-LOCATION * */
72          SIU_S12LOCATION_RESOURCE siulocation = new SIU_S12LOCATION_RESOURCE();
73          AIL ail = new AIL();
74          ail.setAllow_Substitution_Code(new IS("yes"));
75          siulocation.setAIL(ail);
76          ArrayList<SIU_S12LOCATION_RESOURCE> siulocations = new ArrayList<
                SIU_S12LOCATION_RESOURCE>();
77          siulocations.add(siulocation);
78          /** END SIU_S12-LOCATION &  SIU_S12.SERVICE * */
79          siuresource.setSIU_S12LOCATION_RESOURCEList(siulocations);
80          siuresource.setSIU_S12SERVICEList(siuservices);
81          /** END SIU_S12.RESOURCE * */
82          ArrayList<SIU_S12RESOURCES> siuresources = new ArrayList<
                SIU_S12RESOURCES>();
83          siuresources.add(siuresource);
84          /** set the information to the message * */
85          siu_s12.setMSH(msh);
86          siu_s12.setSCH(sch);
87          siu_s12.setSIU_S12PATIENTList(siupatients);
```

```
88          siu_s12.setSIU_S12RESOURCESList(siuresources);
89          /**
90           * "MSH|^~\\&|||||||SIU^S12^SIU_S12||||||OK|||||||\r"
91             "SCH||||||||||||||||||||||^^^^^^^^^^^^^^^&Universal_ID&^^^^^^^^^~^
                 Salgueiro&&Meireles&&^^^^^^^^^^^^^^^^^^^^^^^^^^^^|||||||\r"
92             "PID|||^check Digit^^&Universal_ID&^^^^^^|||||||||||||||||||||||||CE
                 1^^^^^~CE 2^^^^^|||||||||||||||\r"
93             "RGS|||rgs algo|||\r" + "AIS|123|||||||||||\r" + "AIL||||||||||yes
                 |||\r";
94           */
95          return siu_s12;
96      }
97  }
```