DEPARTAMENTO DE INFORMÁTICA
Bloco C6 – Piso 3 – Campo Grande, 1749-016 Lisboa
Tel & Fax: +351 217500084

RELATÓRIO FINAL DE PROJECTO

sobre

Especificação de Protocolos para Injecção de Ataques

realizado na

Faculdade de Ciências da Universidade de Lisboa

por

Ana Luísa Batista Cotrim

06 de Novembro de 2006

Resumo

Vivemos num mundo onde o uso da Internet cada vez mais entra nas nossas rotinas diárias. Todos os dias milhões de pessoas fazem uso desta rede de tamanho mundial para fins de trabalho, educação, pesquisa, lazer ou assuntos pessoais: banca *online*, envio ou recepção de documentos para instituições do País, etc. Algumas destas utilizações implicam o armazenamento e troca de informação sensível. É portanto importante pensar na segurança não só dos protocolos de rede, mas também das aplicações utilizadas, dada a sua exposição a uma rede pública não segura. Muito trabalho tem sido desenvolvido no desenho de protocolos seguros, mas os esforços para tornar as aplicações seguras ainda não são suficientes na maioria dos casos. Com efeito, muitas das aplicações usadas por um utilizador comum apresentam vulnerabilidades graves com um potencial destrutivo não desprezável, especialmente se considerarmos a possibilidade de divulgação de informação sensível como detalhes pessoais ou de contas bancárias. Além dos óbvios prejuízos para o próprio, existem também casos em que a exploração de vulnerabilidades causa danos ou prejuízos a terceiros, como no caso em que a máquina é controlada remotamente pelo atacante e usada como *relay* de *spam* ou em ataques de negação de serviço.

Sendo impossível na prática desenhar e concretizar uma aplicação sem erros, torna-se necessário testar não só as funcionalidades desejadas, mas também o comportamento da aplicação na presença de faltas, numa tentativa de identificar problemas. Um método usado para atingir este objectivo é a injecção de faltas. Consiste em introduzir faltas (tanto de *hardware* como de *software*) num sistema e observar o seu efeito. No campo da segurança informática, a injecção de faltas é geralmente feita ao nível do software, mais particularmente nos pontos de execução de uma aplicação onde há *input* do exterior (leitura de um ficheiro, recepção de um pacote da rede, *input* do utilizador), visto serem esses pontos mais sensíveis a ataques. O presente trabalho vem na continuação de um projecto do CEPEI do ano passado, realizado por João Antunes, em que foi concretizada uma ferramenta de injecção de ataques chamada AJECT. Esta ferramenta simula a execução de um protocolo, no qual injecta pacotes que não obedecem à especificação: valores inválidos, campos demasiado grandes, etc. Na sua primeira encarnação, o AJECT

obtinha a especificação do protocolo em estudo a partir de classes Java escritas especificamente para cada protocolo a usar e que depois eram compiladas juntamente com a ferramenta. O objectivo do trabalho descrito neste relatório é eliminar a necessidade de escrever código para especificar um protocolo para uso no AJECT. Para isso foi desenvolvida uma ferramenta gráfica que produz um ficheiro XML com a especificação do protocolo.

Índice

1. Introdução ……...7

1.1 Organização do documento ……7

2. Objectivos do projecto e contexto do trabalho ……9

3. Metodologia e calendarização do trabalho ……12

3.1 Processo de desenvolvimento de software ……...12

3.2 Calendarização ……12

3.3 Análise de requisitos ……..13

3.3.1 Análise de requisitos da especificação ……14

3.3.2 Análise de requisitos da ferramenta ……14

3.4 Análise de riscos ……14

3.4.1 Identificação e gestão de riscos ……...15

4. Trabalho realizado ……17

4.1 Desenho ……...17

4.2 Package define ……17

4.3 Especificação de um protocolo usando o package define ……...18

4.3.1 ID ……19

4.3.2 Variable e SimpleVariable ……..19

4.3.3 Values e SimpleValues ……20

4.3.4 Field e SimpleField ……..21

4.3.5 Message e SimpleMessage ……..22

4.3.6 State e SimpleState ……..23

4.3.7 Transition e SimpleTransition ……24

4.3.8 TransitionMessage e SimpleTransitionMessage ……25

4.3.9 Event e SimpleEvent ……26

4.3.10 Condition e SimpleCondition ……27

4.3.11 Operator ……27

4.3.12 StartingValue e SimpleStartingValue ……28

4.3.13 ProtocolDetails ……...28

4.4.14 Protocol e SimpleProtocol ……..29

4.3.15 DefineGUI ……..30

4.4 Ferramenta DefineGUI ……30
5 Sumário, conclusões e trabalho futuro ……..37
5.1 Sumário e conclusões ……37
5.2 Trabalho futuro ……..37

Lista de figuras

Ilustração da classe Protocol ……29
Screenshot da ferramenta *DefineGUI* : Protocol Details……..31
Screenshot da ferramenta *DefineGUI* : Fields……..32
Screenshot da ferramenta *DefineGUI* : Messages……..……..33
Screenshot da ferramenta *DefineGUI* : States……..34
Screenshot da ferramenta *DefineGUI* :TransitionMessage……..35

Bibliografia ……..39

Anexos ……..40
Anexo A: Diagrama de Classes ……...40
Anexo B: Mapas de Gantt ……...47
Anexo C: Especificação do protocolo IMAP (versão simples) ……...49

Capítulo 1

Introdução

Este relatório final sobre o trabalho realizado na disciplina de Projecto de Engenharia Informática no âmbito do Curso de Especialização Profissional em Engenharia Informática pretende descrever os seus objectivos, contexto, métodos e a solução encontrada. Trata-se de um projecto na área da segurança informática com alguns elementos de teoria da computação, a realizado no ano lectivo 2005/2006, nas instalações do Departamento de Informática da FCUL, e coordenado pelo Professor Nuno Ferreira Neves.

É feito após o final da minha licenciatura em Informática nesta mesma Faculdade, onde sempre tive predilecção por assuntos da subárea da Informática que se ocupa com a arquitectura de sistemas e redes de computadores (tendo seguido o perfil com o mesmo nome), e mais particularmente com a segurança informática.

O campo da segurança informática está em franco crescimento, tanto em investigação como importância. Isto deve-se à cada vez maior digitalização da vida comum. Todos os dias são efectuadas operações sobre a Internet usando informação sensível: dados pessoais, bancários, empresariais e outros; e cada vez mais utilizadores se ligam à rede.

O presente projecto vem no seguimento de outro projecto do ano anterior, também no âmbito do CEPEI realizado por João Antunes. O seu projecto tratava da injecção de ataques em componentes de software e nele foi escrita uma ferramenta chamada AJECT que oferece essa funcionalidade. O projecto realizado por mim este ano ocupa-se com a especificação de protocolos para futuro uso em ferramentas como o AJECT.

1.1 Organização do documento

O relatório aqui apresentado está dividido em cinco partes distintas. Começa na primeira parte com uma introdução ao trabalho realizado. Na segunda parte do documento são descritos os objectivos do projecto e o contexto do trabalho. Na terceira parte é descrita e metodologia e calendarização do trabalho, bem como uma análise aos riscos do projecto. Na quarta parte o trabalho realizado é descrito ao pormenor. Por fim,

na quinta parte estão um sumário do trabalho, conclusões e uma discussão do trabalho futuro.

Capítulo 2

Objectivos do projecto e contexto do trabalho

Como foi referido na introdução, o contexto deste projecto é a segurança informática, em particular a descoberta de vulnerabilidades usando o método da injecção de ataques.

Um ataque a uma aplicação ou sistema tem como alvo uma vulnerabilidade, isto é, um erro no seu desenho, configuração ou codificação. É através dessa(s) vulnerabilidade(s) que um ataque tenta subverter a função e funcionamento da aplicação ou sistema. O seu objectivo pode ser simplesmente fazer com que a aplicação/sistema fique indisponível aos seus utilizadores através um *crash* (a isto chama-se um ataque de negação de serviço – *denial of service*), ou despoletar a execução de código, obter informação sensível, ou elevação de privilégios.

Nos dias que correm, cada vez mais aplicações (e alguns sistemas operativos) incorporam o uso da Internet nas suas funcionalidades, muitas vezes sem considerar que isso as expõe a uma rede pública e não segura. Isto significa que qualquer vulnerabilidade existente nas aplicações está potencialmente acessível, directa ou indirectamente, a qualquer pessoa com uma ligação à Internet. À primeira vista pode parecer que esta exposição de vulnerabilidades só é perigosa para sistemas nos quais uma intrusão pode ser lucrativa: instituições bancárias, grandes empresas, etc. Contudo, na prática isto não é verdade. A exploração de vulnerabilidades para usos criminosos é uma realidade e não se limita apenas a sistemas comerciais, empresariais ou bancários. Com efeito, a exploração de vulnerabilidades em larga escala tem vindo a aumentar. Um dos métodos mais comuns é através de *worms*, que ao contrário dos vírus de software, não requerem necessariamente de intervenção do utilizador para "infectar" a máquina. A maioria dos *worms* existentes tem como alvo vulnerabilidades do popular sistema operativo Windows da Microsoft. Após o *worm* se ter instalado, tenta propagar-se para outras máquinas. Geralmente o seu objectivo é apenas infectar o maior número de máquinas possível, mas algumas têm objectivos maliciosos, como roubar informação pessoal e bancária, executar um ataque de negação de serviço (participando nele **todas** as máquinas infectadas), ou

funcionar como um *backdoor* (um pequeno programa que permite controlar a máquina remotamente). Neste último caso, o verdadeiro objectivo é a extorsão, sob ameaça de um ataque de negação de serviço, ou o aluguer para distribuição de enormes quantidades de emails publicitários (o chamado *spam*) a comerciantes menos escrupulosos. Estas actividades causam prejuízos financeiros (no caso das negações de serviço) e tráfego excessivo na rede (no caso do *spam*) a terceiros. É portanto importante que, para além de testar o funcionamento correcto e normal de uma aplicação, também haja um esforço para detectar possíveis vulnerabilidades acessíveis pela rede. Este tipo de testes, infelizmente, tem sido inexistente em alguns casos e insuficiente em muitos outros.

Uma aplicação é geralmente atacada através dos dados que recebe como *input*: campos de texto, ficheiros, mensagens recebidas da rede. A injecção de ataques e este projecto focam-se neste ultimo tipo de *input*. Uma ferramenta de injecção de ataques executa o seu trabalho interagindo com uma aplicação específica através de um protocolo utilizado por ela. Os ataques são feitos durante a simulação do protocolo, através de pacotes maliciosos, contendo, por exemplo, valores inválidos ou campos trocados. O efeito destas mensagens é monitorizado, sendo registado o formato das mensagens maliciosas que perturbaram o funcionamento normal da aplicação.

Tanto na injecção de ataques como em outras simulações, é necessário especificar, isto é, descrever de uma maneira suficientemente detalhada, o protocolo em estudo para que este possa ser simulado correctamente. Em todas as ferramentas open source dedicadas à simulação de redes e protocolos que se encontram disponíveis na Internet, esta especificação é feita com módulos de código (geralmente escritos em C ou Java) que são carregados ou ligados com a ferramenta. Este método de especificação tem várias desvantagens. Além de o desenvolvimento ser lento e susceptível a erros, são também necessários conhecimentos sobre o API específico da ferramenta que se pretende usar.

A ferramenta AJECT usa um método semelhante ao descrito acima. O protocolo a usar é especificado através de classes Java escritas pelo utilizador, que contêm a semântica do protocolo e o formato das suas mensagens.

O objectivo deste projecto é encontrar um método de especificar protocolos para serem usados numa ferramenta como o AJECT sem que seja necessário escrever código. A

intenção é eliminar uma dificuldade das tentativas de descoberta de vulnerabilidades: a necessidade de escrever uma mini simulação de um protocolo e fazer o seu *debug* só para ser possível testar uma parte específica de uma aplicação.

Apesar da motivação principal deste projecto ser a injecção de ataques, as especificações produzidas também podem ser úteis para qualquer ferramenta que necessite de entender um protocolo, seja para o simular totalmente, para facilitar o seu estudo, ou para testar novos protocolos, sem haver necessidade de escrever código.

Capítulo 3

Metodologia e calendarização do trabalho

3.1 Processo de desenvolvimento de software

Existem vários modelos de desenvolvimento de software, cada um com diferentes sequências de fases de desenvolvimento. Os mais conhecidos são o modelo em espiral, incremental, *extreme programming*, protótipagem e unificado. O sucesso de um projecto depende muito do modelo escolhido, dada a necessidade de escolher um modelo que se adapte bem ao problema em questão. O trabalho presente, tratando-se de um projecto de pequena dimensão e com uma natureza criativa e experimental, seguiu o modelo em espiral. Este modelo de desenvolvimento de software contempla ciclos sucessivos de desenho, implementação, integração e testes, ao longo dos quais as funcionalidades vão sendo aperfeiçoadas e refinadas. Tratando-se de um projecto cujo objectivo é a especificação de protocolos, é lógico assumir que o método encontrado para o fazer vá sofrer pequenas alterações ao longo do desenvolvimento. Esta contínua refinação do método faz com que o modelo em espiral, com o seu gradual melhoramento e expansão de funcionalidades do software, seja o mais indicado para este projecto, tendo sido previstos três ciclos.

3.2 Calendarização

O projecto teve as seguintes fases:

- Análise do problema
- Estudo sobre os vários métodos e linguagens de especificação
- Criação e descrição de um método de especificação
- Desenho de uma ferramenta gráfica de especificação que utilize o método descrito
- Concretização e teste dessa mesma ferramenta
- Análise das especificações produzidas
- Elaboração de um relatório final de projecto

Ao longo de todo o projecto houve uma análise contínua sobre o funcionamento de protocolos, de modo a aperfeiçoar a especificação proposta.

Devido ao tamanho reduzido da equipa do projecto (1 pessoa) e às mudanças ao método proposto ao longo do desenvolvimento, a concretização da ferramenta de especificação foi algo flexível.

A calendarização foi cumprida sem grandes desvios para todas as tarefas menos a última, a elaboração do relatório final. Tal deveu-se a problemas de saúde que embora não encapacitantes, resultaram num estado depressivo que diminuiu significativamente a produtividade e adiaram a entrega por uma quantidade não desprezável de tempo.

Encontram-se em anexo dois mapas de Gantt com a calendarização prevista inicialmente e a que se realizou efectivamente.

3.3 Análise de requisitos

O presente projecto tem dois objectivos principais, o desenvolvimento de um método de especificação de protocolos para uso na injecção de ataques, e o desenvolvimento de uma ferramenta gráfica que permita efectuar a especificação de protocolos segundo o método desenvolvido na primeira parte do projecto. Assim, teremos dois conjuntos de requisitos a identificar, os de especificação e os da ferramenta.

3.3.1 Análise de requisitos da especificação

Objectivo: Definir um método para especificação de protocolos
Requisitos:

- Deve ser possível definir o funcionamento completo de um protocolo (estados (incluindo inicial e final), comandos, formatos das mensagens, valores válidos, etc.)
- Deve ser possível acrescentar informação relevante para a injecção de ataques

3.3.2 Análise de requisitos da ferramenta

Objectivo: Concretizar uma aplicação gráfica para efectuar a especificação de protocolos
Requisitos:

- A ferramenta deve ser simples e de uso fácil
- Deve suportar totalmente o método de especificação definido
- Deve ser possível guardar a especificação para um ficheiro, num ou mais formatos conhecido

3.4 Análise de riscos

Um projecto de desenvolvimento de software tem sempre alguns riscos envolvidos durante a sua realização. Estes riscos podem ser categorizados em duas categorias principais: técnicos e de projecto. Riscos técnicos referem-se, como nome indica, a problemas do foro técnico, tais como: pouca familiaridade com a linguagem escolhida, escolha de ferramentas inadequadas, etc. Riscos de projecto referem-se a possiveis problemas de gestão. Qualquer um destes riscos tem associado uma probabilidade e um impacto, que exprimem respectivamente a probabilidade da situação descrita pelo risco acontecer, e o impacto que essa situação terá no projecto. Existem quatro niveis de impacto com decrescente ordem de severidade. Segue-se uma pequena tabela ilustrando-os e exemplificando-os.

Nivel de Impacto	Risco técnico	Risco de projecto	Efeito no projecto
Catastrófico	Escolha de uma plataforma de hardware inadequada	Problemas legais (como infricção de patentes)	Cancelamento do projecto ou enorme adiamento no lançamento, custos monetários astronómicos
Crítico	Escolha de uma linguagem inadequada	Perda de membros chave da equipa	Grande adiamento do lançamento do projecto, grandes custos monetários
Médio	Escolha de bibliotecas	Falta de comunicação entre membros da equipa	Adiamento moderado, alguns custos monetários
Insignificante	Pouca familiaridade com a(s) ferramenta(s) escolhida(s)	Indisponibilidade de alguns membros da equipa para uma reunião	Pequeno adiamento, pequenos custos monetários

3.4.1 Identificação e gestão de riscos

	Tipo	Impacto	Probabilidade
Risco: Escolha de uma linguagem inadequada	Técnico	Crítico	Baixa
Gestão: Analisar aprofundadamente os requisitos e as linguagens disponíveis antes de selecionar a linguagem a usar.			

	Tipo	Impacto	Probabilidade
Risco: Pouca familiaridade com as ferramentas e/ou linguagens usadas	Técnico	Insignificante	Média
Gestão: Consulta de tutoriais e manuais *online* ou nas bibliotecas da faculdade, consulta de especialistas.			

	Tipo	Impacto	Probabilidade
Risco: Identificação de requisitos deficiente	Projecto	Crítico	Baixa – média
Gestão: Analisar aprofundadamente o problema, analisar métodos de especificação já existentes. Comunicar frequentemente com o orientador.			

	Tipo	Impacto	Probabilidade
Risco: Escolha de um formato de ficheiro para a especificação que iniba o seu uso em outras aplicações	Técnico	Crítico	Baixa
Gestão: Pesquisar *standards* de armazenamento de informação, limitar as opções a formatos conhecidos e adoptados.			

	Tipo	Impacto	Probabilidade
Risco: Falha ou indisponíbilidade no equipamento necessário	Técnico	Crítico	Baixa
Gestão: Fazer *backups* regularmente, assegurar acesso a equipamento alternativo.			

Capítulo 4

Trabalho realizado

4.1 Desenho

O trabalho realizado tem três componentes principais: a interface gráfica, o package Java de nome *define*, e a biblioteca *open source XStream* (http://xstream.codehaus.org).

A interface gráfica, de nome *DefineGUI*, foi programada em Java e permite especificar um protocolo e guardar a sua especificação em XML num ficheiro. A representação em Java da especificação é feita através de instâncias de classes do *package define*, que são usadas para descrever o protocolo como uma máquina de estados. A biblioteca XStream é depois utilizada para converter os objectos do package define que representam o protocolo a especificar para XML. Podem ser produzidas duas versões da especificação: uma com referências a objectos e estruturas de dados, e outra com referências e estruturas simples. A primeira opção destina-se a aplicações que também utilizem a biblioteca XStream para a leitura e *demarshalling* do XML. A segunda opção destina-se a aplicações que não usem a biblioteca XStream, podendo também não usar nenhuma classe do package define se tal for desejado.

4.2 *Package define*

A definição de um protocolo usando as classes do*package define* utiliza conceitos da teoria da computação, de redes e da própria injecção de ataques. Os conceitos mais importantes e essenciais para a especificação básica de um protocolo são as mensagens, os campos, as transições e os estados. Para além desses estão também presentes os conceitos de variáveis (locais e globais), eventos e condições. Cada um destes conceitos tem uma classe no *package define* que lhe corresponde e o representa. Existem também classes auxiliares utilizadas por outras classes, e uma classe onde está programada a aplicação gráfica *DefineGUI*.

4.3 Especificação de um protocolo usando o package define

A especificação de um protocolo com a profundidade necessária para a injecção de ataques involve uma grande quantidade de informação. Como tal, o método definido pelo package define tenta evitar ao máximo a repetição de definições. Para isso, todos os objectos passiveis de serem reutilizados têm um atributo de tipo ID, que contém a sua identificação (única) definida pelo utilizador. As referências a estes objectos são feitas aos seus atributos ID. Este método é bastante eficaz em conjunto com estruturas de dados como diccionários ou Hashtables, em que os objectos ID são chaves para o pedaço de informação reutilizável em questão. A utilização deste sistema é mais óbvia na representação de campos e mensagens.

A definição de um protocolo começa, conceptualmente, pelo elemento mais pequeno: os campos das mensagens. É possivel definir o seu tipo, tamanho, valores válidos, etc. Em seguida, a construção de mensagens. No package define, mensagens são apenas uma sequência de referências a campos numa ordem especifica. Depois da definição de campos e mensagens, são definidos os estados. As transições são definidas no estado de origem e podem transitar para o mesmo estado (usado para descrever situações como, por exemplo, que comandos podem ser dados num determinado estado sem que causem uma transição para outro estado). As transições contêm estruturas denominadas por mensagens de transição. Tratam-se de objectos com uma referência ao ID da mensagem a usar, seguida de uma lista de pares (ID, valor). Esta lista define que campos têm que ter um valor especifico, e qual é esse valor, para que a transição seja possivel. As referências ID nesta lista referem-se a campos da mensagem escolhida. Os campos não presentes nesta lista mas presentes na definição da mensagem são assumidos como tendo um valor válido de acordo com a especificação do campo. É também possivel especificar uma lista vazia para uma mensagem de transição, significando que qualquer dos valores possiveis para todos os campos é válido para esta transição.

Além destes conceitos principais, existem também funcionalidades como variáveis locais (aos estados), váriaveis globais, e eventos simples (guardar variável, iniciar temporizador, enviar/receber mensagem). Estes eventos podem ser despoletados aquando da entrada ou saída de um estado ou por outros eventos.

Segue-se uma descrição mais aprofundada das classes do package define e do XML produzido a partir delas. Serão dados exemplos baseados numa especificação do protocolo IMAP produzida pela ferramenta, ou exemplos genéricos quando a funcionalidade em questão não foi utilizada na especificação produzida..

4.3.1 ID.java

A classe ID tem a função de identificador de componentes. Todos os objectos do *package define* que podem ser referenciados por outros apresentam um atributo do tipo ID. Este objecto tem apenas uma String como seu unico atributo, e é substituido na versão simples pelo atributo que contém.

Normal	Simples
<id>um valor</id>	N/A

4.3.2 Variable e SimpleVariable.java

Estas classes representam uma variável simples. Os seus atributos são um objecto ID que a identifica, e uma String representando o tipo de dados desta variável. É usada para representar variáveis globais e locais em outros objectos do package.

Normal	Simples
<define.Variable> <name> <id>variavel_exemplo</id> </name> <type>Integer</type> </define.Variable>	<define.SimpleVariable> <name>variavel_exemplo</name> <type>Integer</type> </define.SimpleVariable>

4.3.3 Values e SimpleValues.java

A classe Values tem duas funções: define um intervalo numérico através dos seus atributos minValue e maxValue, e contém uma lista (no atributo values) de valores específicos não necessáriamente numéricos. É usada apenas por e em outras classes, não sendo necessário especificar o tipo dos valores contidos, estando essa informação explicita nas classes que fazem uso de objectos Values. É principalmente usada para definir valores válidos para um campo. Tem também um atributo booleano chamado reverse, que caso seja *true* significa que todos os valores excepto os especificados são válidos. Como objectos desta classe estão sempre associados a outros objectos e nunca são acedidos directamente, as suas versões normal e simples são idênticas.

Normal	Simples
<values> <minValue>0.0</minValue> <maxValue>0.0</maxValue> <values> <string>SELECT</string> <string>EXAMINE</string> <string>CREATE</string> <string>DELETE</string> <string>SUBSCRIBE</string> <string>UNSUBSCRIBE</string> </values> <reverse>false</reverse> </values>	<values> <minValue>0.0</minValue> <maxValue>0.0</maxValue> <values> <string>SELECT</string> <string>EXAMINE</string> <string>CREATE</string> <string>DELETE</string> <string>SUBSCRIBE</string> <string>UNSUBSCRIBE</string> </values> <reverse>false</reverse> </values>

4.3.4 Field e SimpleField.java

Os campos são o componente mais básico da especificação de um protocolo. São representados pela classe Field e descrevem, como o nome indica, um campo de um pacote de dados. Cada objecto desta classe contém nos seus atributos informação básica sobre o campo, bem como pormenores interessantes do ponto de vista da injecção de ataques. A classe Field extende a classe Variable com os seguintes atributos:

- int size, tamanho em bits, ou bytes se o tipo for string
- ID isSizeOf, serve para indicar que o campo presente representa o tamanho de outro campo cuja ID é o atributo isSizeOf. Este será null se isto não acontecer.
- boolean isFileName, indica se o campo representa o nome de um ficheiro.
- boolean isCommand, indica se o campo representa um comando.
- char delimiter, representa o caracter delimitador deste campo (opcional).
- Values values, valores válidos para este campo.

Normal	Simples
<define.Field> <size>0</size> <isFileName>false</isFileName> <isCommand>true</isCommand> <delimiter> </delimiter> <values> <minValue>0.0</minValue> <maxValue>0.0</maxValue> <values> <string>SELECT</string> <string>EXAMINE</string> <string>CREATE</string> <string>DELETE</string> <string>SUBSCRIBE</string> <string>UNSUBSCRIBE</string> </values> <reverse>false</reverse>	<define.SimpleField> <name>select ... mailbox</name> <type>String</type> <size>0</size> <isFileName>false</isFileName> <isCommand>true</isCommand> <delimiter> </delimiter> <values> <minValue>0.0</minValue> <maxValue>0.0</maxValue> <values> <string>SELECT</string> <string>EXAMINE</string> <string>CREATE</string> <string>DELETE</string> <string>SUBSCRIBE</string>

```
      </values>
      <name>
        <id>select ...
mailbox</id>
      </name>
      <type>String</type>
    </define.Field>
```

```
<string>UNSUBSCRIBE</string>
        </values>
        <reverse>false</reverse>
      </values>
    </define.SimpleField>
```

4.3.5 Message e SimpleMessage.java

Uma mensagem no *package define* representa o formato de um pacote de dados. É formada por uma sequência de campos numa ordem específica. A sua classe correspondente é a classe Message. É identificada por um atributo ID, e a definição da estrutura da mensagem é feita no atributo ArrayList fieldlist. Foi usado um objecto ArrayList devido à sua facilidade de manipulação e outra importante propriedade: o conceito de ordem. Como a ordem em que os campos aparecem numa mensagem é crucial para a sua definição, a vantagem de usar esta estrutura de dados é óbvia.

O atributo fieldlist contém uma sequência de referências para objectos ID de campos. Assim sendo, a definição de uma mensagem é uma lista de referências para campos. Isto torna possivel a reutilização de campos em várias mensagens sem ser necessário redefini-los para cada uma delas. A reutilização dos campos usando as suas referências também se traduz numa poupança de memória.

Normal	Simples

`<define.Message>`
`  <name>`
`  <id>AUTH com mailbox arg</id>`
`  </name>`
`  <fieldList>`
`  <define.ID reference="../../../../fields /define.Field[26]/name"/>`
	`<define.SimpleMessage>`
	`  <name>AUTH com mailbox arg</name>`
	`  <fieldList>`
	
	`<string>status/append</string>`
	`  <string>mailbox</string>`
	`  <string>status/append arg</string>`

```
        <define.ID
reference="../../../../fields
/define.Field[14]/name"/>
        <define.ID
reference="../../../../fields
/define.Field[3]/name"/>
      </fieldList>

<initDelimiter>A02</initDelimiter>

<endDelimiter>\r\n</endDelimiter>
      <sendOnly>false</sendOnly>

<receiveOnly>false</receiveOnly>
    </define.Message>
```

```
      </fieldList>

<initDelimiter>A02</initDelimiter>

<endDelimiter>\r\n</endDelimiter>
    </define.SimpleMessage>
```

4.3.6 State e SimpleState.java

O importante conceito de estado é representado pela classe State. É identificado por um atributo ID, e define eventos a serem lançados aquando da entrada e saída deste estado. Contém o atributo localVariables, que é uma lista de objectos Variable e representa as variáveis locais a este estado. O atributo transitions é o mais interessante desta classe, e define que transições são possíveis e como são feitas. Trata-se de uma lista de objectos Transition.

Normal	Simples
<define.State> <name> <id>selected state</id> </name> <auth>false</auth> <connected>false</connected> <endState>false</endState> <localVariables/> <transitions> <define.Transition>	<define.SimpleState> <name>selected state</name> <auth>false</auth> <connected>false</connected> <endState>false</endState> <localVariables/> <transitions> <define.SimpleTransition> (…) </transitions>

(…) </transitions> </define.State>	</define.SimpleState>

4.3.7 Transition e SimpleTransition.java

Esta classe define uma transição entre estados. Contém um atributo ID que referencia o estado de destino da transição, uma lista de objectos TransitionMessage que definem as mensagens necessárias para efectuar esta transição, uma lista de objectos Condition que representam as condições necessárias para esta transição ser possivel, e um atributo boolean error, que indica se esta transição indica um erro.

As transições são definidas assumindo que é o cliente a iniciá-las. Assim sendo, a primeira mensagem na lista transitions é mandada pelo cliente, sendo a segunda recebida do servidor (seguindo o mesmo critério se forem necessárias mais que duas mensagens). Se for o servidor a iniciar a transição, é necessário definir uma mensagem vazia para ocupar o primeiro lugar na lista antes de definir a mensagem a receber.

Normal	Simples
<define.Transition> <toState reference="../../../name"/> <messages> <define.TransitionMessage> <name reference="../../../../../../../messages/define.Message[7]/name"/> <fieldValues/> <launchEvent/> </define.TransitionMessage> </messages> <conditions/> <error>false</error> </define.Transition>	<define.SimpleTransition> <toState>selected state</toState> <messages> <define.SimpleTransitionMessage> <name>SEL com</name> <fieldValues/> <launchEvent/> </define.SimpleTransitionMessage> </messages> <conditions/> <error>false</error> </define.SimpleTransition>

4.3.8 TransitionMessage e SimpleTransitionMessage.java

A classe TransitionMessage contém a informação necessária para construir uma mensagem de transição. Para ser possível efectuar uma transição, é necessário que alguns campos assumam valores especificos, por exemplo: para passar do estado inicial para o estado autenticado é necessário transitar até ele através duma mensagem com o campo comando com o valor "AUTH". Esta classe representa este conceito de mensagem com valores especificos. Para isso, contém uma referência ID para a mensagem a utilizar, e uma Hashtable com IDs de campos (Field) como chaves, e objectos Value indexados por essas chaves. A construção de uma mensagem a partir de um objecto TransitionMessage é feita da depois de obter o ID da mensagem a usar da seguinte forma: percorre os campos definidos na mensagem; por cada campo verifica se a sua ID existe na Hashtable como chave; se existir, obter da Hashtable o objecto Values correspondente a essa chave e usar o(s) valor(es) lá contido(s) como valor desse campo na mensagem; se não existir, dar um valor segundo a definição do campo e/ou critérios do programa a usar a especificação.

Existem também um atributo launchEvent, que é uma lista de eventos a lançar ou executar aquando do envio ou recepção desta mensagem.

Normal	Simples
<define.TransitionMessage> <name reference="../../../../../../../ messages/define.Message[7]/name"/> <fieldValues/> <launchEvent/> </define.TransitionMessage>	<define.SimpleTransitionMessage> <name>SEL com</name> <fieldValues/> <launchEvent/> </define.SimpleTransitionMessage>

4.3.9 Event e SimpleEvent.java

A classe Event representa o dificil conceito de eventos. Na especificação de protocolos, os eventos que vão ser definidos são a atribuição de valores a variáveis, o envio ou recepção de mensagens, e o lançamento de temporizadores. Para além dos atributos que representam estes casos, a classe apresenta também um atributo ID que a identifica.

Para a atribuição de valores são usados os atributos saveTo e saveField. São ambos do tipo ID e representam respectivamente os Ids da variável onde o valor vai ser guardado e a origem desse valor.

Para o envio e/ou recepção de mensagens são usados os atributos send e receive, sendo ambos do tipo TransitionMessage e representando respectivamente o envio e a recepção de uma mensagem específica.

Por fim, o lançamento de temporizadores é feito através dos atributos timerSeconds e timerExpireEvent. O primeiro é um float e representa em segundos o tempo que deverá passar até o temporizador expirar. O segundo é do tipo ID e referencia o evento que deve ser lançado quando o temporizador expirar.

Normal	Simples
<define.Event> <eventName> <id>example_event</id> </eventName> <timerSeconds>5.0</timerSeconds> <timerExpireEvent> <id>event</id> </timerExpireEvent> </define.Event>	<define.SimpleEvent> <eventName>example_event</eventName> <timerSeconds>5.0</timerSeconds> <timerExpireEvent>event</timerExpireEvent> </define.SimpleEvent>

4.3.10 Condition e SimpleCondition.java

Esta classe representa condições. Uma condição é uma expressão cujo valor de verdade é avaliado, e é formada por dois argumentos e um operador. Isto é reflectido nesta classe pelos atributos ID firstArg, secondArg, e pelo atributo int operator. Existe também o Object value, cuja função é tomar o lugar do secondArg na condição quando a comparação pretendida é feita com um valor especifico e não entre campos ou variáveis. Neste caso, o atributo secondArg terá o valor null. Os operadores são definidos na classe Operator.

Normal	Simples
<define.Condition> <firstArg reference="../../../../../../../globalVariables/define.Variable/name"/> <secondArg reference="../../../../../../../fields/define.Field/name"/> <operator>4</operator> </define.Condition>	<define.SimpleCondition> <firstArg>example variable</firstArg> <secondArg>example field</secondArg> <operator>4</operator> </define.SimpleCondition>

4.3.11 Operator.java

Esta classe representa operadores usados por outras classes. Estão definidos os seguintes: adição, subtracção, multiplicação, divisão, igual, não igual, maior que, menor que, maior ou igual que, menor ou igual que.

4.3.12 StartingValue e SimpleStartingValue.java

A classe StartingValue tem como objectivo definir valores iniciais para a execução do protocolo. Extende a classe Variable adicionando um atributo Object value, contendo o valor inicial pretendido.

Normal	Simples
<define.StartingValue> <value class="string">abc12345</value> <name> <id>password</id> </name> <type>String</type> </define.StartingValue>	<define.SimpleStartingValue> <name>password</name> <type>String</type> <value class="string">abc12345</value> </define.SimpleStartingValue>

4.3.13 ProtocolDetails.Java

Esta classe, tal como a anterior, contém informação importante para a execução do protocolo, e não tanto para a sua definição. Nos seus atributos especifica o nome do protocolo, o porto e a morada IP do servidor alvo, o tipo do protocolo (TCP ou UDP) e o estado inicial. Na versão simples, a informação contida neste objecto está integrada nos atributos do objecto SimpleProtocol.

Normal	Simples
<details> <name>IMAP</name> <connectToIP>10.10.10.10</connectToIP> <connectToPort>143</connectToPort> <type>TCP</type> <startingState>not authenticated state</startingState> </details>	N/A

4.3.14 Protocol e SimpleProtocol.java

A classe Protocol contém todas as estruturas descritas anteriormente. A sua função é encapsular todos os componentes do protocolo definido. Assim sendo, nos seus atributos estão presentes um objecto ProtocoDetails, listas de StartingValue (os valores iniciais), Field (os campos do protocolo), Message (as mensagens), Variable (variáveis globais), State (os estados) e Event (os eventos).

Normal	Simples
<define.Protocol> <details> <name></name> <connectToIP></connectToIP> <connectToPort>0</connectToPort> <type></type> <startingState></startingState> </details> <startingValues/> <fields/> <messages/> <globalVariables/> <states/> <events/> </define.Protocol>	<define.SimpleProtocol> <name></name> <connectToIP></connectToIP> <connectToPort>0</connectToPort> <type></type> <startingState></startingState> <startingValues/> <fields/> <messages/> <globalVariables/> <states/> <events/> </define.SimpleProtocol>

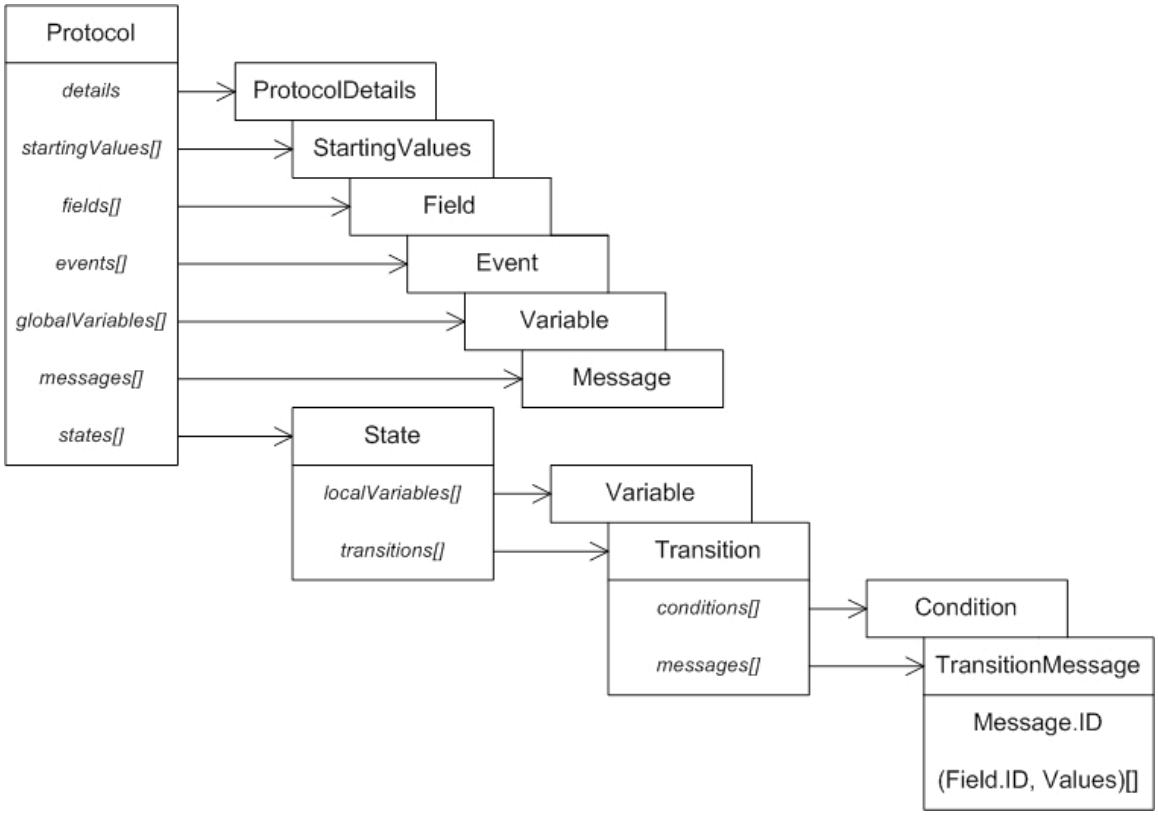


Figura 4.1 Ilustração das classes (conceptualmente) encapsuladas pela classe Protocol

4.3.15 DefineGUI.java

Esta classe contém o programa Define. Trata-se de uma aplicação gráfica que permite definir um protocolo usando as classes e conceitos descritos acima. É este programa que juntamente com a biblioteca Xstream faz o marshalling de objectos Protocol e SimpleProtocol para XML.

4.4 Ferramenta *DefineGUI*

A ferramenta DefineGUI é constituída por duas janelas: uma janela principal, onde o utilizador introduz a maior parte da informação, e uma pequena janela que apresenta os estados definidos e que permite selecionar um estado a visualizar na janela principal.
A janela principal está dividida também em duas partes. No topo, uma *toolbar* com botões que permitem carregar uma especificação previamente guardada, criar uma nova especificação, guardar a especificação já feita, guardar a especificação em XML, carregar

uma especificação em XML, exportar uma especificação em XML simples, e mostrar/esconder a mini janela dos estados. Em baixo da toolbar, está a area principal de especificação. Existe aí um conjunto de *tabs*, cada um dedicado a um conceito do protocolo a especificar:

- Protocol Details – detalhes como nome do protocolo a especificar, IP e porto alvo, com ligação ou não (TCP vs UDP), e estado inicial.
- Starting Values – valores necessários para uma simulação do protocolo a especificar (opcional).
- Fields – descrição de todos os campos usados nas mensagens do protocolo.
- Messages – definição das mensagens usadas pelo protocolo, usando os campos definidos na *tab* Fields.
- Global Variables – definição de variáveis globais necessárias à simulação do protocolo (opcional).
- Events – definição de eventos (envio/recepção de mensagens, temporizadores, salvaguarda de informação recebida).
- States – definição de um estado. Esta tab apresenta apenas um estado de cada vez. Para visualizar um estado nesta tab, é necessário selecioná-lo na mini janela de estados e carregar no botão "Show". É nesta tab que são definidas as transições entre estados.
-

Seguem-se alguns *screenshots* para melhor ilustrar o funcionamento da ferramenta.

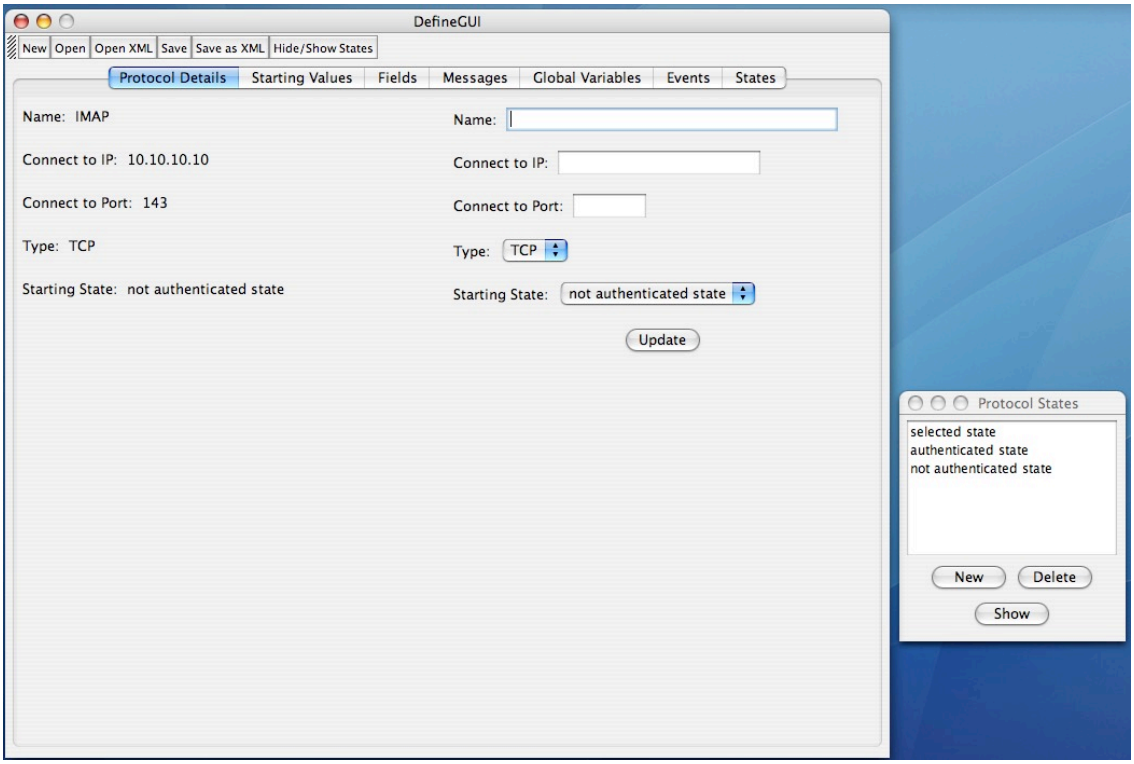


Figura 4.2 Protocol Details e mini janela de estados.

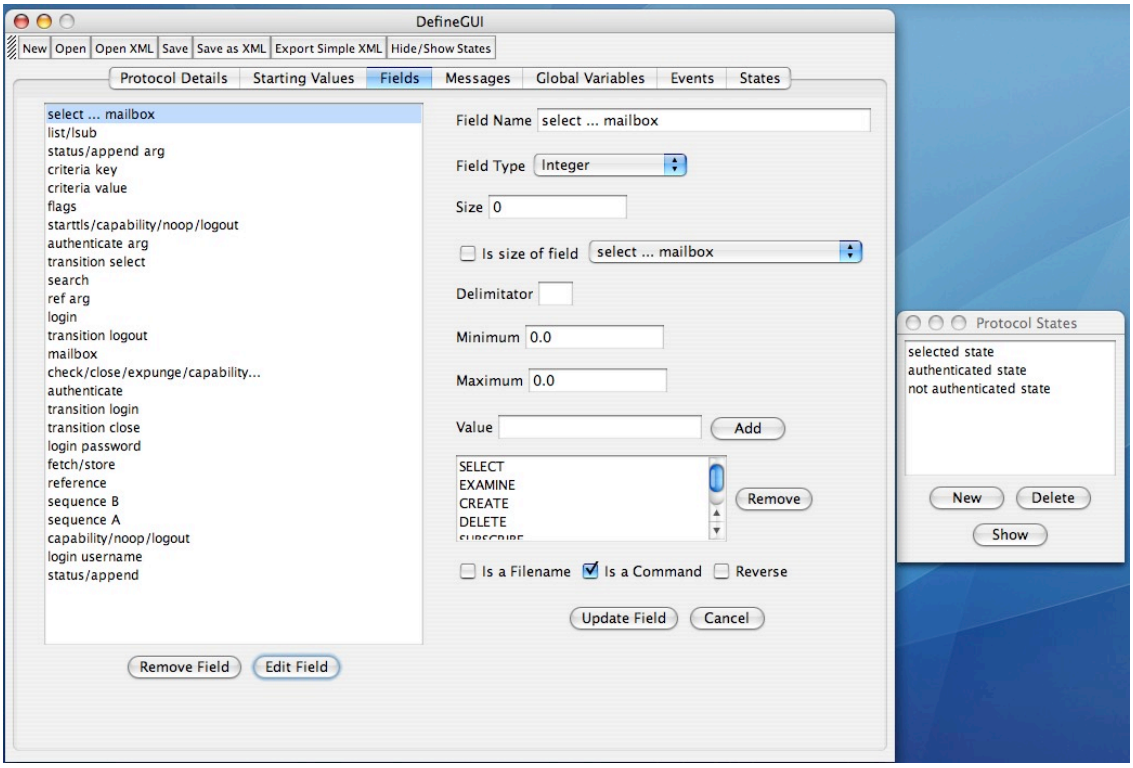


Figura 4.3 Aspecto da tab Fields aquando da modificação de um campo já definido. O botão "Update Field" seria "Add Field" se o campo em questão ainda não tivesse sido definido.

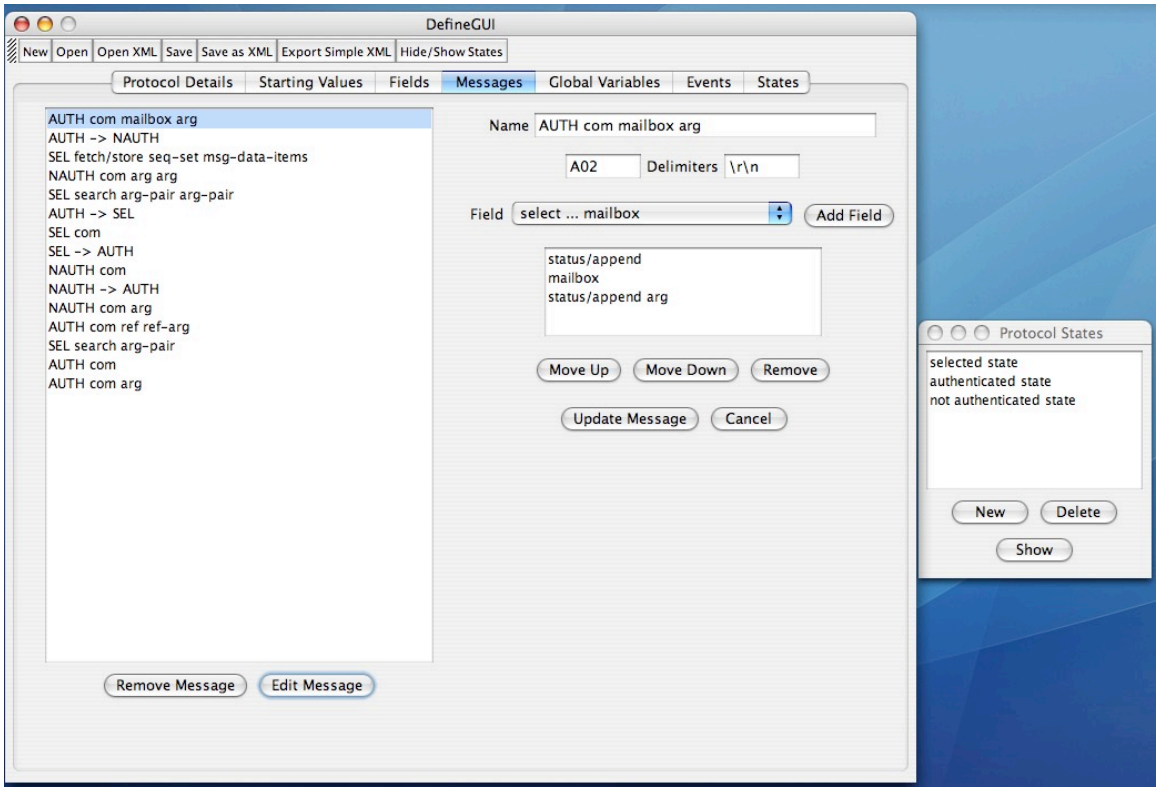


Figura 4.4 Aspecto da tab Message aquando da modificação de uma mensagem. Tal como na tab Fields, o botão "Update Message" tem o aspecto "Add Message" quando a mensagem está a ser definida.

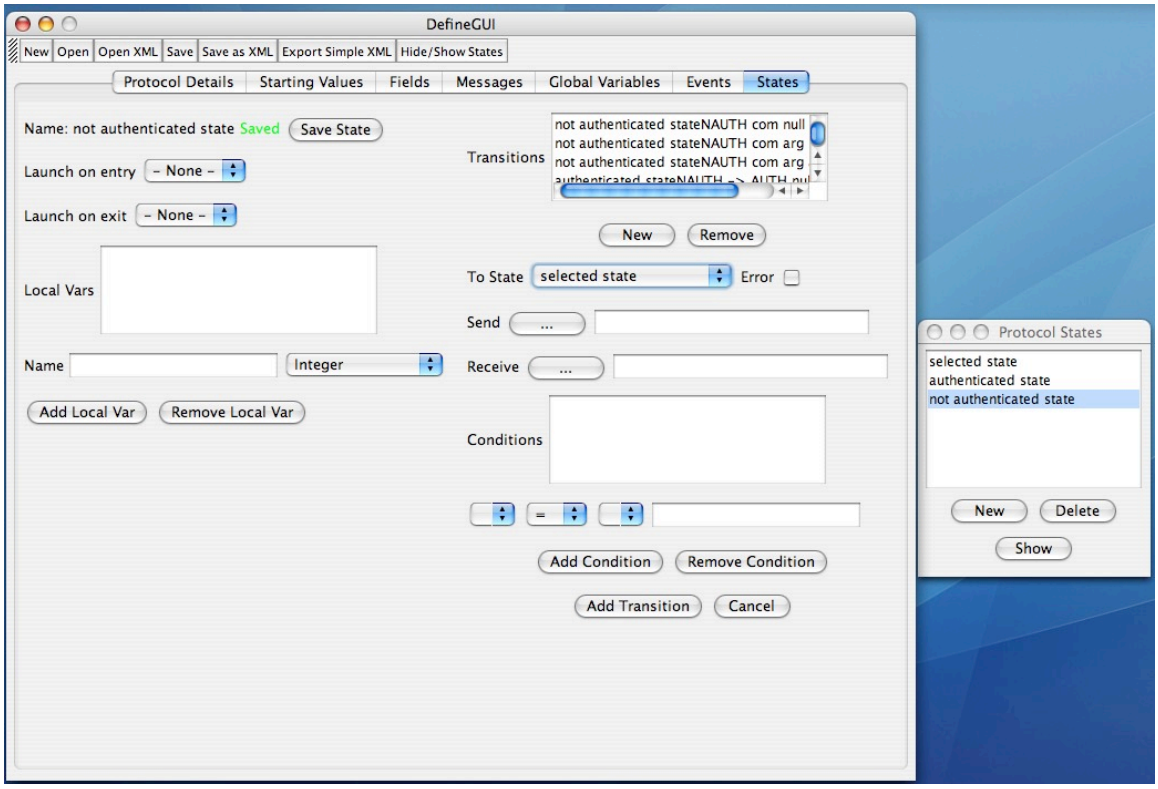


Figura 4.5 Aspecto da tab States com o estado "non authenticated state" selecionado. Neste caso existem já várias transições definidas, que aparecem nesta janela no formato "<estado destino> <identificador da mensagem a enviar> <identificador da mensagem a receber>".

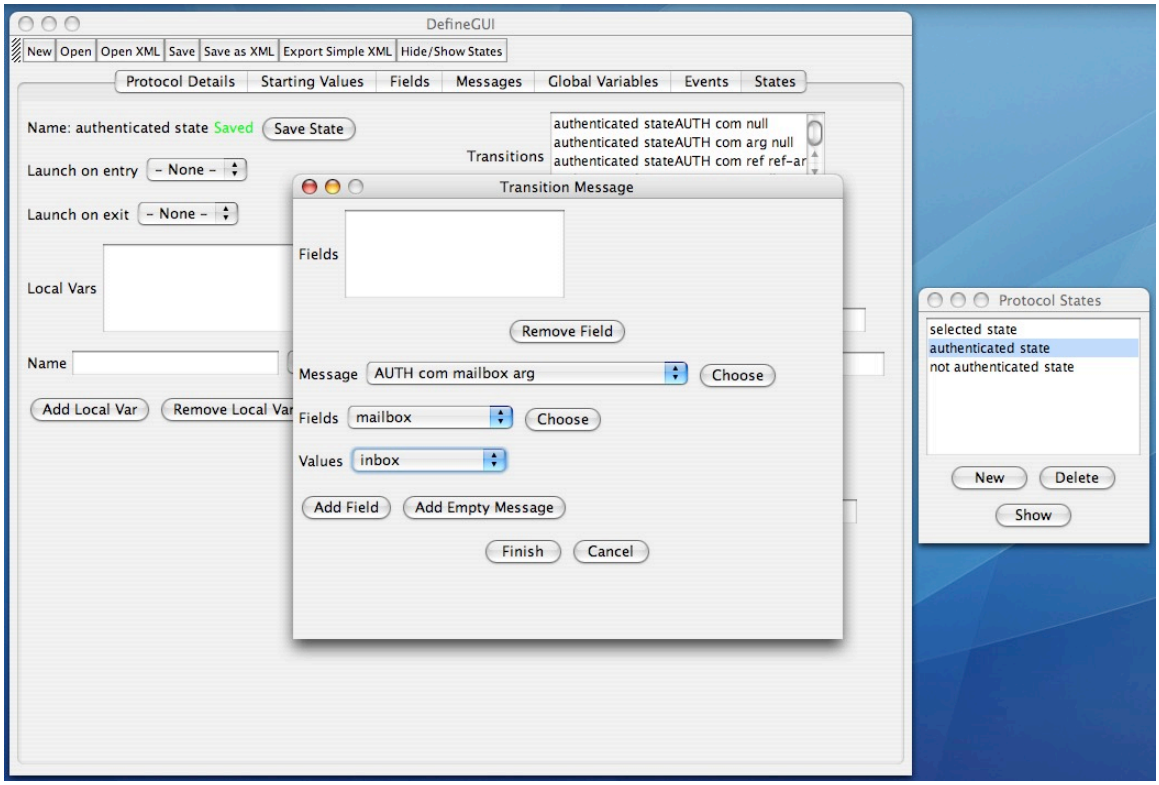


Figura 4.6 Subjanela de definição de uma mensagem de transição.

Capítulo 5

Sumário, conclusões e trabalho futuro

5.1 Sumário e conclusões

O trabalho realizado neste projecto definiu um método de especificação de protocolos para uso na injecção de ataques, bem como produziu uma ferramenta gráfica para auxiliar a especificação. O método desenvolvido consiste na representação de um protocolo como uma máquina de estados, e representa praticamente toda a informação necessária não só à injecção de ataques, mas também a uma simulação correcta de um protocolo.

A especificação em si é feita usando a ferramenta DefineGUI, com objectos Java pertencentes ao package *define*, ambos desenvolvido neste projecto. A especificação é depois convertida em XML pela biblioteca Open Source *Xstream* em um de dois modos: normal ou simples. A versão normal tira partido de estruturas de dados não triviais (como ArrayLists e Hashtables) e a referências a objectos, suportadas pela biblioteca *XStream*. A versão simples não contém estruturas de dados complexas (a não ser simples arrays unidimensionais) nem referências a objectos. Isto permite que especificações produzidas em modo simples possam ser lidas e interpretadas por quaisquer aplicações que usem outra biblioteca de *demarshalling* de XML além do *XStream*, sendo até possivel não usar qualquer classe do package define se desejado.

5.2 Trabalho futuro

O método de especificação de protocolos apresentado neste relatório contempla as funções típicas de um protocolo, necessárias à injecção de ataques. Contudo, existem duas excepções: os casos da ambiguidade e do paralelismo.

A ambiguidade refere-se ao facto de em alguns protocolos, algumas series de comandos resultarem em ambiguidades. Estas ambiguidades vêm da própria forma como o protocolo foi construido, e são dificeis de especificar, visto que a(s) mensagem(s) a

receber fruto de um comando ambiguo vão depender da codificação/configuração do servidor.

O paralelismo neste contexto refere-se a mais do que um canal de comunicação entre o cliente e o servidor. Um exemplo deste comportamento é o protocolo FTP (*File Transfer Protocol*), em que é iniciado uma nova ligação TCP quando o cliente executa o comando para *download* de um ficheiro. Este comportamento é por enquanto impossivel de descrever na presente encarnação do define.

A falta de expressão de situações como ambiguidade e paralelismo, sendo partes importantes de alguns protocolos, serão foco do trabalho futuro deste projecto.

Bibliografia

[1] J. Postel e J. Reynolds. *RFC 959 - File Transfer Protocol*, Outubro 1985.

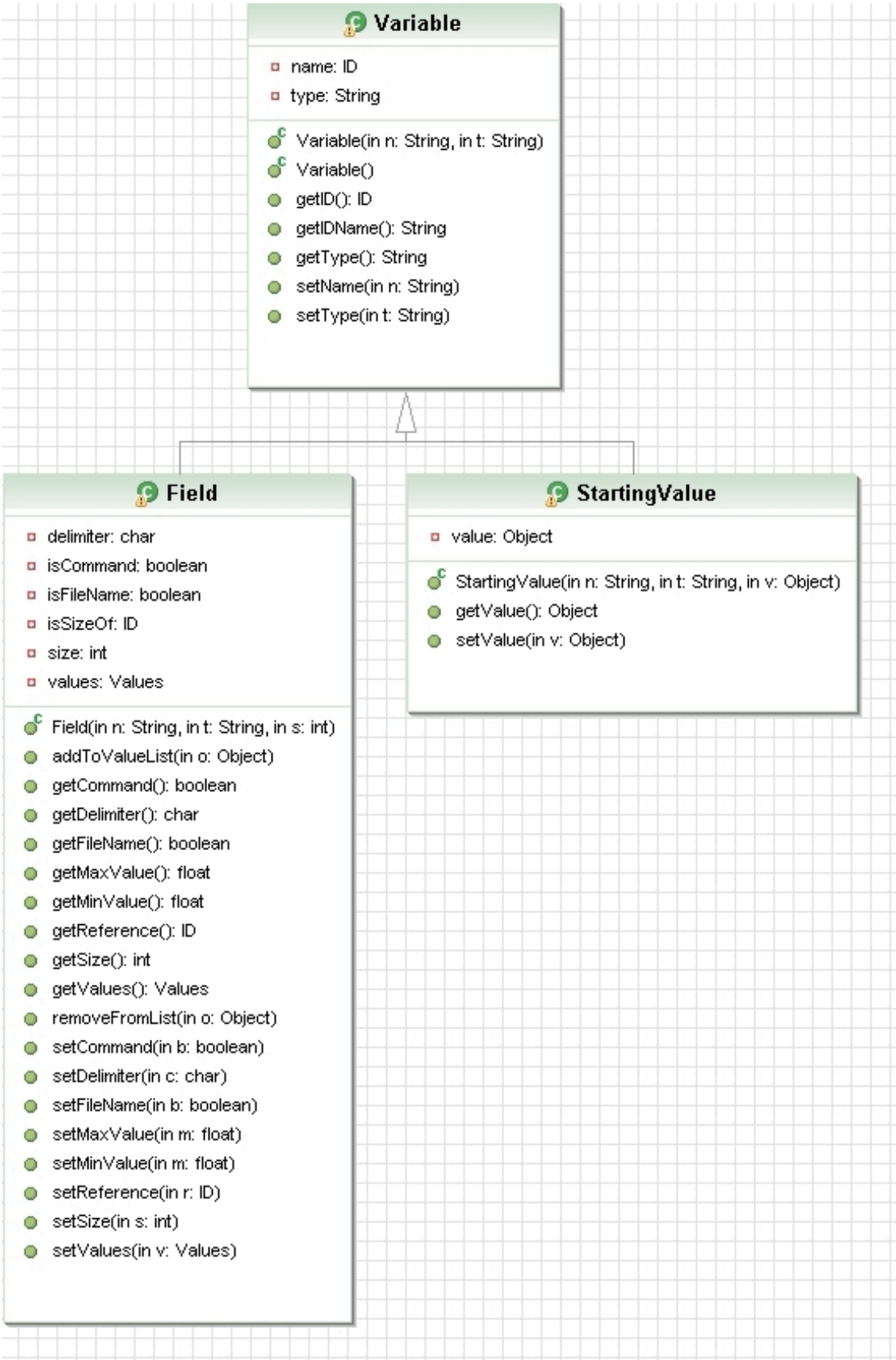
[2] M. Crispin, *RFC 3501 - INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*, Março 2003.

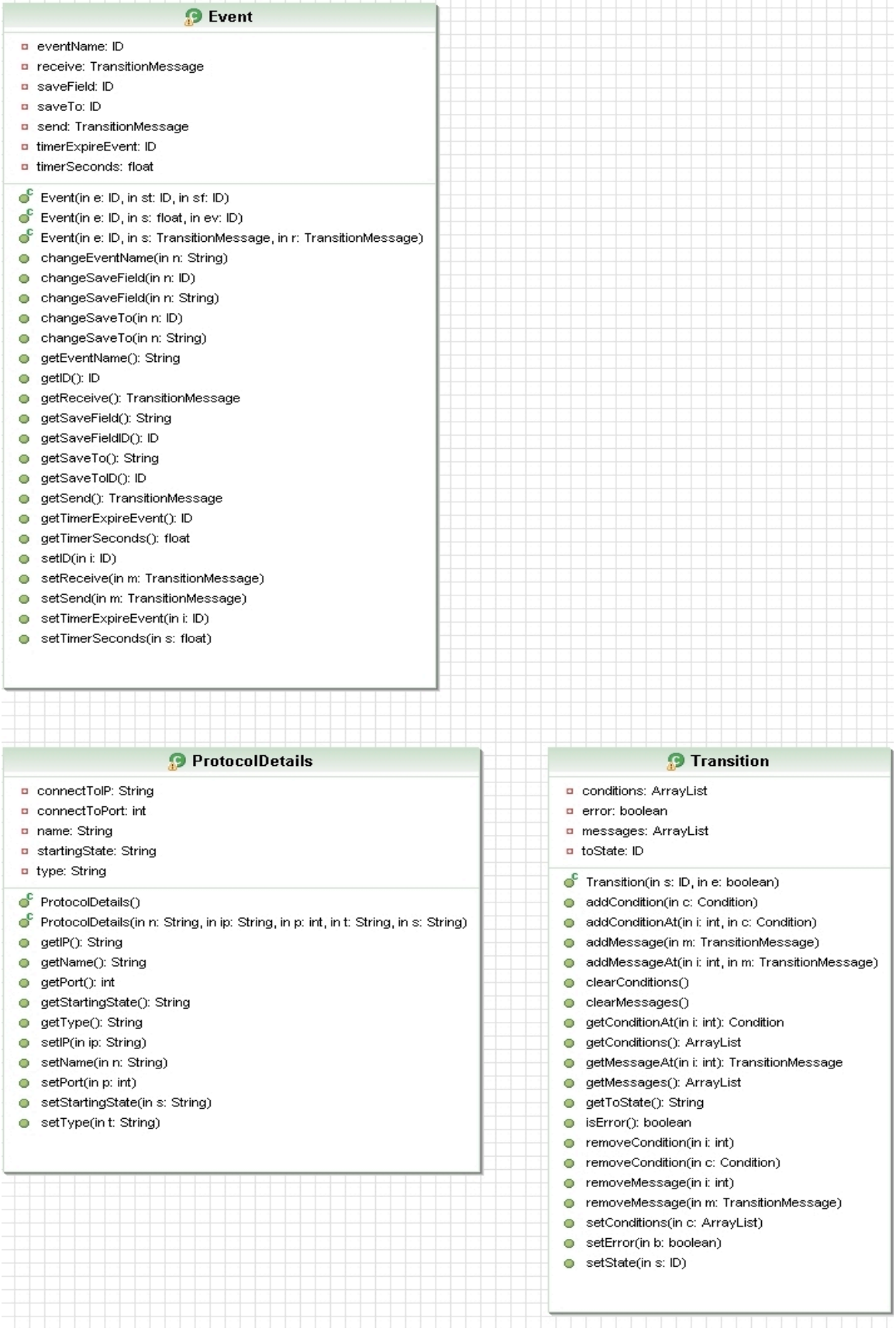
[3] D. Crocker e P. Overell. *RFC 2234 - Augmented BNF for Syntax Specifications: ABNF*, Novembro 1997.

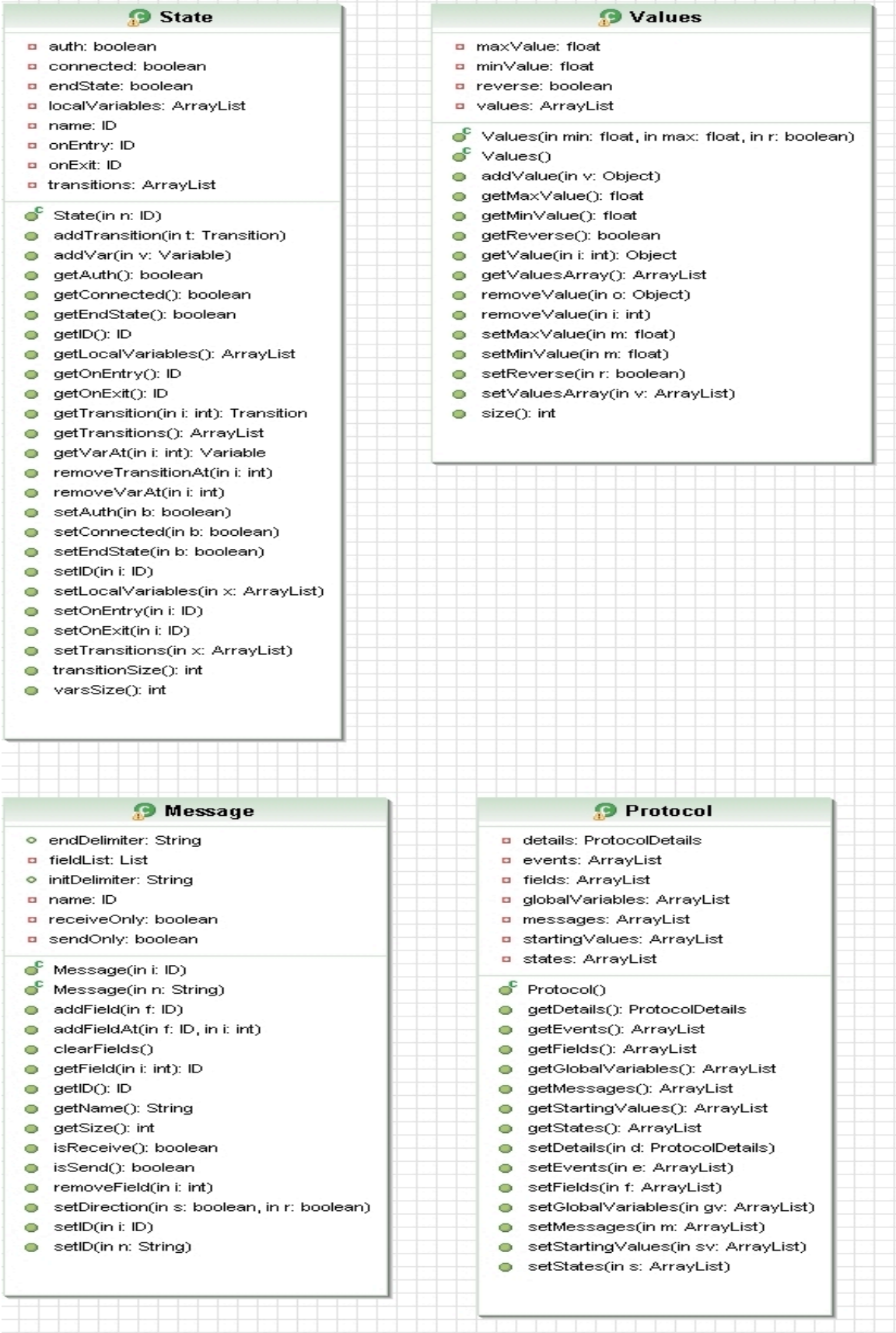
[4] J. Boyer. *RFC 3076 - Canonical XML Version 1.0*, Março 2001.

[5] J.M. Pires dos Santos, "Linguagens Formais e Autómatos" in *Colecção Textos de Apoio do DI-FCUL*, 2003.

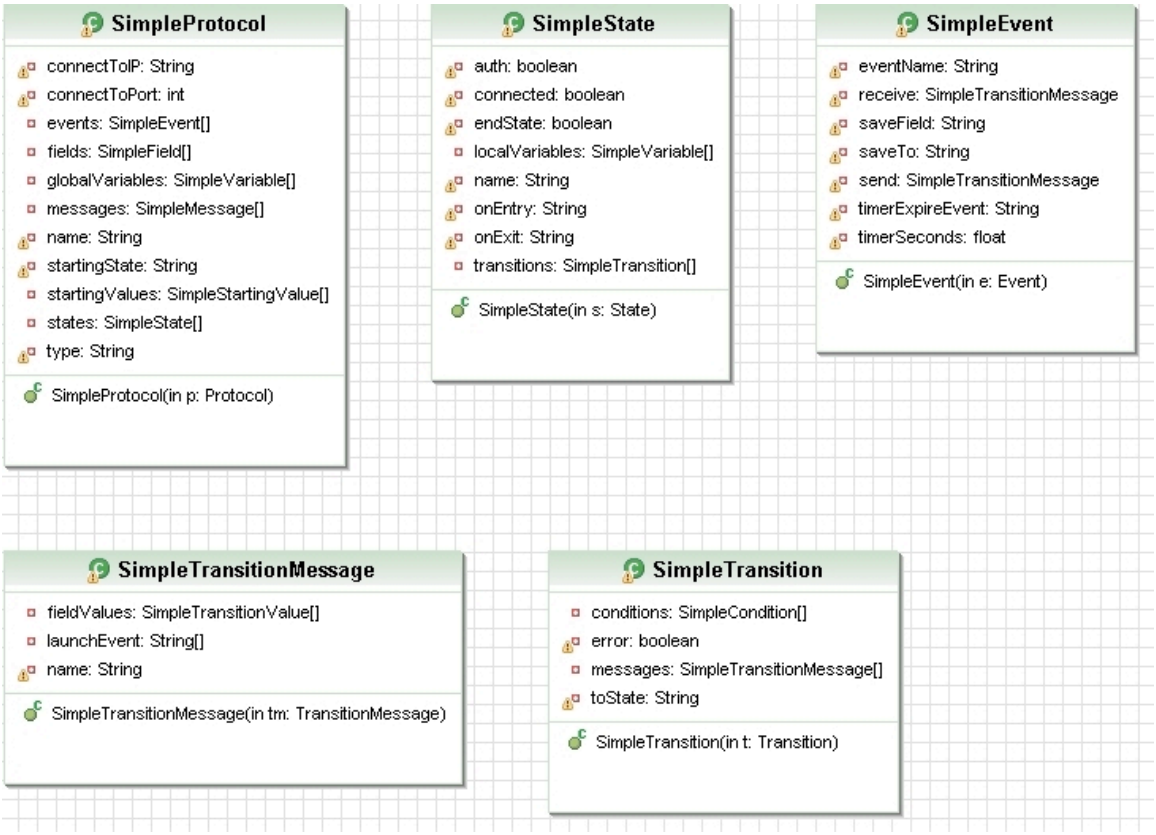
Anexo A: Diagrama de classes

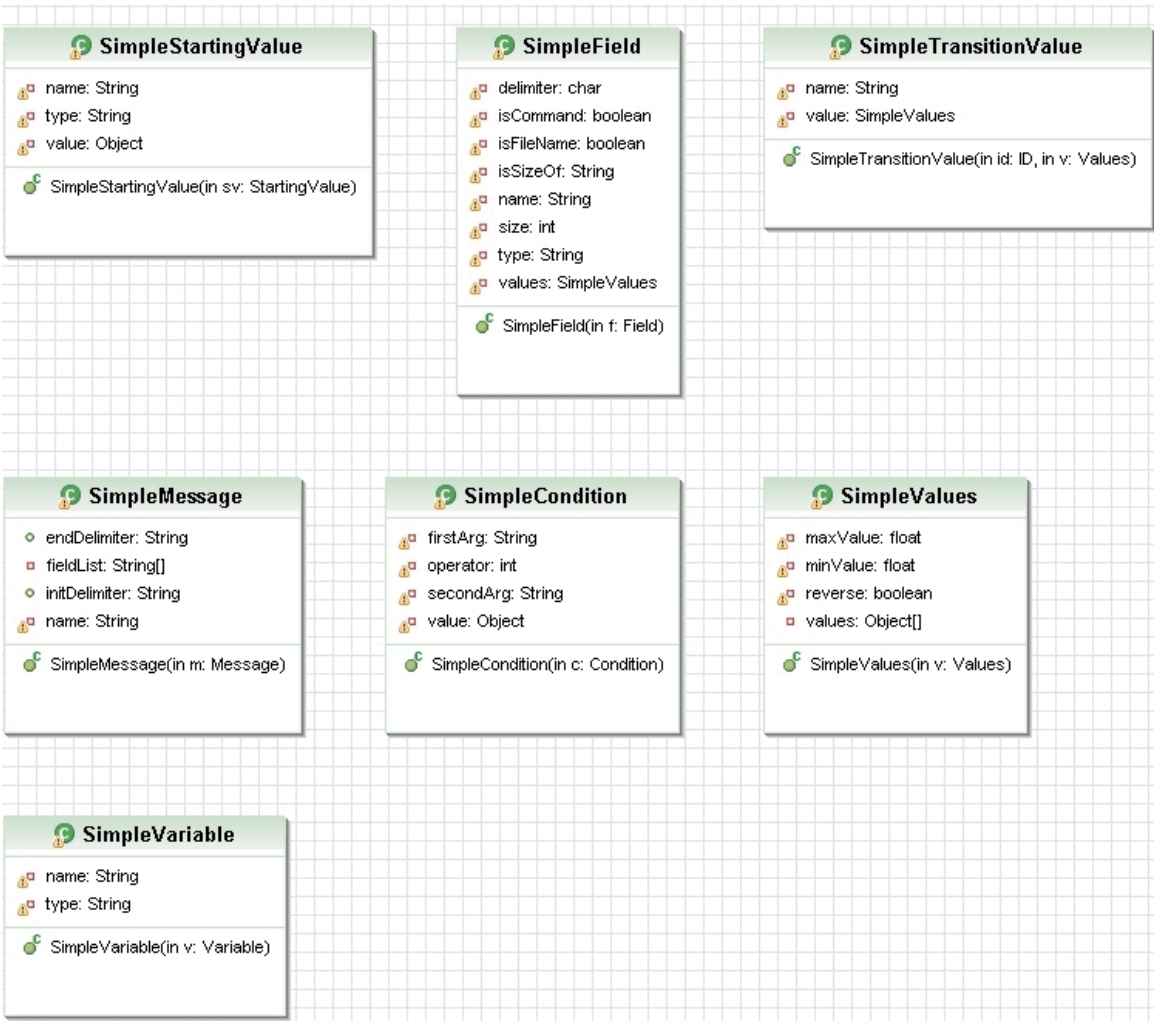

Variable
name: ID
type: String
Variable(in n: String, in t: String)
Variable()
getID(): ID
getIDName(): String
getType(): String
setName(in n: String)
setType(in t: String)
Field
delimiter: char
isCommand: boolean
isFileName: boolean
isSizeOf: ID
size: int
values: Values
Field(in n: String, in t: String, in s: int)
addToValueList(in o: Object)
getCommand(): boolean
getDelimiter(): char
getFileName(): boolean
getMaxValue(): float
getMinValue(): float
getReference(): ID
getSize(): int
getValues(): Values
removeFromList(in o: Object)
setCommand(in b: boolean)
setDelimiter(in c: char)
setFileName(in b: boolean)
setMaxValue(in m: float)
setMinValue(in m: float)
setReference(in r: ID)
setSize(in s: int)
setValues(in v: Values)
StartingValue
value: Object
StartingValue(in n: String, in t: String, in v: Object)
getValue(): Object
setValue(in v: Object)


Event
eventName: ID
receive: TransitionMessage
saveField: ID
saveTo: ID
send: TransitionMessage
timerExpireEvent: ID
timerSeconds: float
Event(in e: ID, in st: ID, in sf: ID)
Event(in e: ID, in s: float, in ev: ID)
Event(in e: ID, in s: TransitionMessage, in r: TransitionMessage)
changeEventName(in n: String)
changeSaveField(in n: ID)
changeSaveField(in n: String)
changeSaveTo(in n: ID)
changeSaveTo(in n: String)
getEventName(): String
getID(): ID
getReceive(): TransitionMessage
getSaveField(): String
getSaveFieldID(): ID
getSaveTo(): String
getSaveToID(): ID
getSend(): TransitionMessage
getTimerExpireEvent(): ID
getTimerSeconds(): float
setID(in i: ID)
setReceive(in m: TransitionMessage)
setSend(in m: TransitionMessage)
setTimerExpireEvent(in i: ID)
setTimerSeconds(in s: float)
ProtocolDetails
connectToIP: String
connectToPort: int
name: String
startingState: String
type: String
ProtocolDetails()
ProtocolDetails(in n: String, in ip: String, in p: int, in t: String, in s: String)
getIP(): String
getName(): String
getPort(): int
getStartingState(): String
getType(): String
setIP(in ip: String)
setName(in n: String)
setPort(in p: int)
setStartingState(in s: String)
setType(in t: String)
Transition
conditions: ArrayList
error: boolean
messages: ArrayList
toState: ID
Transition(in s: ID, in e: boolean)
addCondition(in c: Condition)
addConditionAt(in i: int, in c: Condition)
addMessage(in m: TransitionMessage)
addMessageAt(in i: int, in m: TransitionMessage)
clearConditions()
clearMessages()
getConditionAt(in i: int): Condition
getConditions(): ArrayList
getMessageAt(in i: int): TransitionMessage
getMessages(): ArrayList
getToState(): String
isError(): boolean
removeCondition(in i: int)
removeCondition(in c: Condition)
removeMessage(in i: int)
removeMessage(in m: TransitionMessage)
setConditions(in c: ArrayList)
setError(in b: boolean)
setState(in s: ID)


State
auth: boolean
connected: boolean
endState: boolean
localVariables: ArrayList
name: ID
onEntry: ID
onExit: ID
transitions: ArrayList
State(in n: ID)
addTransition(in t: Transition)
addVar(in v: Variable)
getAuth(): boolean
getConnected(): boolean
getEndState(): boolean
getID(): ID
getLocalVariables(): ArrayList
getOnEntry(): ID
getOnExit(): ID
getTransition(in i: int): Transition
getTransitions(): ArrayList
getVarAt(in i: int): Variable
removeTransitionAt(in i: int)
removeVarAt(in i: int)
setAuth(in b: boolean)
setConnected(in b: boolean)
setEndState(in b: boolean)
setID(in i: ID)
setLocalVariables(in x: ArrayList)
setOnEntry(in i: ID)
setOnExit(in i: ID)
setTransitions(in x: ArrayList)
transitionSize(): int
varsSize(): int
Values
maxValue: float
minValue: float
reverse: boolean
values: ArrayList
Values(in min: float, in max: float, in r: boolean)
Values()
addValue(in v: Object)
getMaxValue(): float
getMinValue(): float
getReverse(): boolean
getValue(in i: int): Object
getValuesArray(): ArrayList
removeValue(in o: Object)
removeValue(in i: int)
setMaxValue(in m: float)
setMinValue(in m: float)
setReverse(in r: boolean)
setValuesArray(in v: ArrayList)
size(): int
Message
endDelimiter: String
fieldList: List
initDelimiter: String
name: ID
receiveOnly: boolean
sendOnly: boolean
Message(in i: ID)
Message(in n: String)
addField(in f: ID)
addFieldAt(in f: ID, in i: int)
clearFields()
getField(in i: int): ID
getID(): ID
getName(): String
getSize(): int
isReceive(): boolean
isSend(): boolean
removeField(in i: int)
setDirection(in s: boolean, in r: boolean)
setID(in i: ID)
setID(in n: String)
Protocol
details: ProtocolDetails
events: ArrayList
fields: ArrayList
globalVariables: ArrayList
messages: ArrayList
startingValues: ArrayList
states: ArrayList
Protocol()
getDetails(): ProtocolDetails
getEvents(): ArrayList
getFields(): ArrayList
getGlobalVariables(): ArrayList
getMessages(): ArrayList
getStartingValues(): ArrayList
getStates(): ArrayList
setDetails(in d: ProtocolDetails)
setEvents(in e: ArrayList)
setFields(in f: ArrayList)
setGlobalVariables(in gv: ArrayList)
setMessages(in m: ArrayList)
setStartingValues(in sv: ArrayList)
setStates(in s: ArrayList)

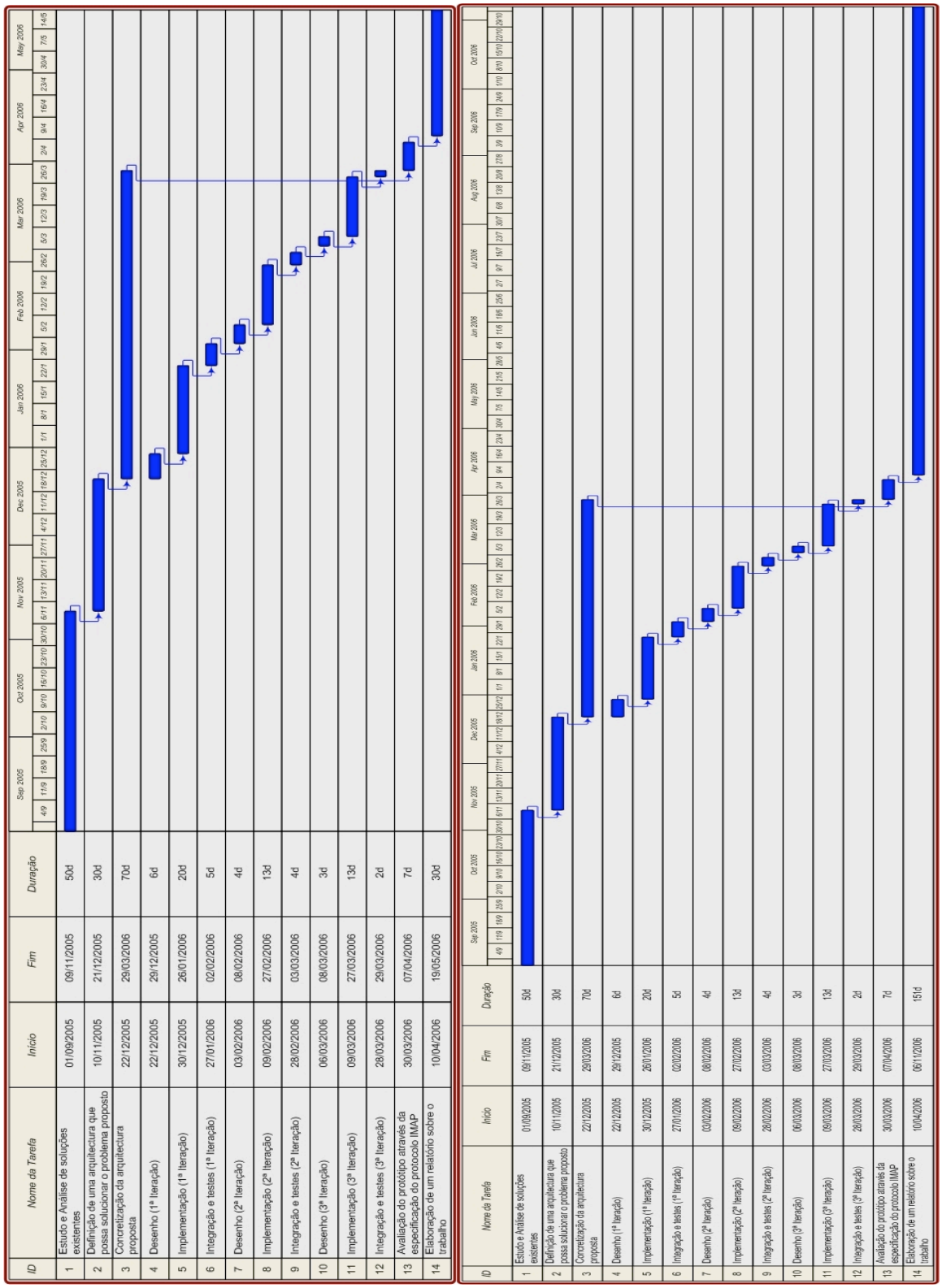

TransitionMessage
fieldValues: Hashtable
launchEvent: ArrayList
name: ID
TransitionMessage(in n: ID)
addEvent(in e: ID)
addEventAt(in e: ID, in i: int)
addValue(in n: ID, in v: Values)
getEvent(in i: int): ID
getEventList(): ArrayList
getID(): ID
getTable(): Hashtable
getValue(in n: ID): Values
removeEvent(in s: ID): boolean
removeValue(in n: String)
setEvent(in e: ArrayList)
setID(in i: ID)
setTable(in t: Hashtable)
Condition
firstArg: ID
operator: int
secondArg: ID
value: Object
Condition()
getFirstArg(): ID
getOperator(): int
getSecondArg(): ID
getValue(): Object
setFirstArgID(in i: ID)
setOperator(in operator: int)
setSecondArgID(in i: ID)
setValue(in v: Object)
Modification
argument: ID
operator: int
value: float
Modification()
getArgument(): ID
getOperator(): int
getValue(): float
setArgument(in argument: ID)
setOperator(in operator: int)
setValue(in value: float)
Operator
ADD: int
DIVIDE: int
EQUALS: int
GREATEREQUALTHAN: int
GREATERTHAN: int
LESSEREQUALTHAN: int
LESSERTHAN: int
MULTIPLY: int
NOTEQUAL: int
SUBTRACT: int
ID
id: String
ID()
ID(in i: String)
changeID(in i: String)
getID(): String


SimpleProtocol
connectToIP: String
connectToPort: int
events: SimpleEvent[]
fields: SimpleField[]
globalVariables: SimpleVariable[]
messages: SimpleMessage[]
name: String
startingState: String
startingValues: SimpleStartingValue[]
states: SimpleState[]
type: String
SimpleProtocol(in p: Protocol)
SimpleState
auth: boolean
connected: boolean
endState: boolean
localVariables: SimpleVariable[]
name: String
onEntry: String
onExit: String
transitions: SimpleTransition[]
SimpleState(in s: State)
SimpleEvent
eventName: String
receive: SimpleTransitionMessage
saveField: String
saveTo: String
send: SimpleTransitionMessage
timerExpireEvent: String
timerSeconds: float
SimpleEvent(in e: Event)
SimpleTransitionMessage
fieldValues: SimpleTransitionValue[]
launchEvent: String[]
name: String
SimpleTransitionMessage(in tm: TransitionMessage)
SimpleTransition
conditions: SimpleCondition[]
error: boolean
messages: SimpleTransitionMessage[]
toState: String
SimpleTransition(in t: Transition)


SimpleStartingValue
name: String
type: String
value: Object
SimpleStartingValue(in sv: StartingValue)
SimpleField
delimiter: char
isCommand: boolean
isFileName: boolean
isSizeOf: String
name: String
size: int
type: String
values: SimpleValues
SimpleField(in f: Field)
SimpleTransitionValue
name: String
value: SimpleValues
SimpleTransitionValue(in id: ID, in v: Values)
SimpleMessage
endDelimiter: String
fieldList: String[]
initDelimiter: String
name: String
SimpleMessage(in m: Message)
SimpleCondition
firstArg: String
operator: int
secondArg: String
value: Object
SimpleCondition(in c: Condition)
SimpleValues
maxValue: float
minValue: float
reverse: boolean
values: Object[]
SimpleValues(in v: Values)
SimpleVariable
name: String
type: String
SimpleVariable(in v: Variable)

Apêndice B: Mapas de Gantt

Calendarização prevista & realizada



ID	Nome da Tarefa	Início	Fim	Duração
1	Estudo e Análise de soluções existentes	01/09/2005	09/11/2005	50d
2	Definição de uma arquitectura que possa solucionar o problema proposto	10/11/2005	21/12/2005	30d
3	Concretização da arquitectura proposta	22/12/2005	29/03/2006	70d
4	Desenho (1ª Iteração)	22/12/2005	29/12/2005	6d
5	Implementação (1ª Iteração)	30/12/2005	26/01/2006	20d
6	Integração e testes (1ª Iteração)	27/01/2006	02/02/2006	5d
7	Desenho (2ª Iteração)	03/02/2006	08/02/2006	4d
8	Implementação (2ª Iteração)	09/02/2006	27/02/2006	13d
9	Integração e testes (2ª Iteração)	28/02/2006	03/03/2006	4d
10	Desenho (3ª Iteração)	06/03/2006	08/03/2006	3d
11	Implementação (3ª Iteração)	09/03/2006	27/03/2006	13d
12	Integração e testes (3ª Iteração)	28/03/2006	29/03/2006	2d
13	Avaliação do protótipo através da especificação do protocolo IMAP	30/03/2006	07/04/2006	7d
14	Elaboração de um relatório sobre o trabalho	10/04/2006	19/05/2006	30d

ID	Nome da Tarefa	Início	Fim	Duração
1	Estudo e Análise de soluções existentes	01/09/2005	09/11/2005	50d
2	Definição de uma arquitectura que possa solucionar o problema proposto	10/11/2005	21/12/2005	30d
3	Concretização da arquitectura proposta	22/12/2005	29/03/2006	70d
4	Desenho (1ª Iteração)	22/12/2005	29/12/2005	6d
5	Implementação (1ª Iteração)	30/12/2005	26/01/2006	20d
6	Integração e testes (1ª Iteração)	27/01/2006	02/02/2006	5d
7	Desenho (2ª Iteração)	03/02/2006	08/02/2006	4d
8	Implementação (2ª Iteração)	09/02/2006	27/02/2006	13d
9	Integração e testes (2ª Iteração)	28/02/2006	03/03/2006	4d
10	Desenho (3ª Iteração)	06/03/2006	08/03/2006	3d
11	Implementação (3ª Iteração)	09/03/2006	27/03/2006	13d
12	Integração e testes (3ª Iteração)	28/03/2006	29/03/2006	2d
13	Avaliação do protótipo através da especificação do protocolo IMAP	30/03/2006	07/04/2006	7d
14	Elaboração de um relatório sobre o trabalho	10/04/2006	06/11/2006	151d

Apêndice C: Expecificação do protocolo IMAP (versão simples)

```
<define.SimpleProtocol>
  <name>IMAP</name>
  <connectToIP>10.10.10.10</connectToIP>
  <connectToPort>143</connectToPort>
  <type>TCP</type>
  <startingState>not authenticated state</startingState>
  <startingValues/>
  <fields>
    <define.SimpleField>
      <name>select ... mailbox</name>
      <type>String</type>
      <size>0</size>
      <isFileName>false</isFileName>
      <isCommand>true</isCommand>
      <delimiter> </delimiter>
      <values>
        <minValue>0.0</minValue>
        <maxValue>0.0</maxValue>
        <values>
          <string>SELECT</string>
          <string>EXAMINE</string>
          <string>CREATE</string>
          <string>DELETE</string>
          <string>SUBSCRIBE</string>
          <string>UNSUBSCRIBE</string>
        </values>
        <reverse>false</reverse>
      </values>
    </define.SimpleField>
    <define.SimpleField>
      <name>list/lsub</name>
```

<type>String</type>
<size>0</size>
<isFileName>false</isFileName>
<isCommand>true</isCommand>
<delimiter> </delimiter>
<values>
<minValue>0.0</minValue>
<maxValue>0.0</maxValue>
<values>
<string>LIST</string>
<string>LSUB</string>
</values>
<reverse>false</reverse>
</values>
</define.SimpleField>
<define.SimpleField>
<name>status/append arg</name>
<type>String</type>
<size>0</size>
<isFileName>false</isFileName>
<isCommand>false</isCommand>
<delimiter null="true"/>
<values>
<minValue>0.0</minValue>
<maxValue>0.0</maxValue>
<values>
<string>(UIDNEXT MESSAGES)</string>
<string>(\\Seen) {310}</string>
</values>
<reverse>false</reverse>
</values>

```
</define.SimpleField>
<define.SimpleField>
  <name>criteria key</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>TEXT</string>
      <string>FROM</string>
      <string>TO</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>criteria value</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>jantunes@di.fc.ul.pt</string>
```

```
      <string>aject</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>flags</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>+FLAGS (\\Deleted)</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>starttls/capability/noop/logout</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
```

```
    <values>
      <string>STARTTLS</string>
      <string>CAPABILITY</string>
      <string>NOOP</string>
      <string>LOGOUT</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>authenticate arg</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>CRAM-MD5</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>transition select</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
```

```
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values/>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>search</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>SEARCH</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>ref arg</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter> </delimiter>
```

```
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>*</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>login</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>LOGIN</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>transition logout</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
```

```
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>A01 LOGOUT</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>mailbox</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>inbox</string>
      <string>Draft</string>
      <string>saved-messages</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>check/close/expunge/capability...</name>
  <type>String</type>
```

```
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter> </delimiter>
  <values>
   <minValue>0.0</minValue>
   <maxValue>0.0</maxValue>
   <values>
    <string>CHECK</string>
    <string>CLOSE</string>
    <string>EXPUNGE</string>
    <string>CAPABILITY</string>
    <string>NOOP</string>
    <string>LOGOUT</string>
   </values>
   <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>authenticate</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter> </delimiter>
  <values>
   <minValue>0.0</minValue>
   <maxValue>0.0</maxValue>
   <values>
    <string>AUTHENTICATE</string>
   </values>
```

```
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>transition login</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>A01 LOGIN aject-user aject-pass</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>login password</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>aject-pass</string>
```

```
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>transition close</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter null="true"/>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>A01 CLOSE</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>fetch/store</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
```

<string>FETCH</string>
<string>STORE</string>
</values>
<reverse>false</reverse>
</values>
</define.SimpleField>
<define.SimpleField>
<name>reference </name>
<type>String</type>
<size>0</size>
<isFileName>false</isFileName>
<isCommand>false</isCommand>
<delimiter> </delimiter>
<values>
<minValue>0.0</minValue>
<maxValue>0.0</maxValue>
<values>
<string>/imap-test/folder1</string>
<string>/imap-test/folder2</string>
</values>
<reverse>false</reverse>
</values>
</define.SimpleField>
<define.SimpleField>
<name>sequence B</name>
<type>String</type>
<size>0</size>
<isFileName>false</isFileName>
<isCommand>false</isCommand>
<delimiter> </delimiter>
<values>

```
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>1</string>
      <string>2</string>
      <string>3</string>
      <string>4</string>
      <string>5</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>capability/noop/logout</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>true</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>CAPABILITY</string>
      <string>NOOP</string>
      <string>LOGOUT</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
```

```
  <name>sequence A</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter>:</delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>1</string>
      <string>2</string>
      <string>3</string>
      <string>4</string>
      <string>5</string>
    </values>
    <reverse>false</reverse>
  </values>
</define.SimpleField>
<define.SimpleField>
  <name>login username</name>
  <type>String</type>
  <size>0</size>
  <isFileName>false</isFileName>
  <isCommand>false</isCommand>
  <delimiter> </delimiter>
  <values>
    <minValue>0.0</minValue>
    <maxValue>0.0</maxValue>
    <values>
      <string>aject-user</string>
```

```
      </values>
      <reverse>false</reverse>
    </values>
  </define.SimpleField>
  <define.SimpleField>
    <name>status/append</name>
    <type>String</type>
    <size>0</size>
    <isFileName>false</isFileName>
    <isCommand>true</isCommand>
    <delimiter> </delimiter>
    <values>
      <minValue>0.0</minValue>
      <maxValue>0.0</maxValue>
      <values>
        <string>STATUS</string>
        <string>APPEND</string>
      </values>
      <reverse>false</reverse>
    </values>
  </define.SimpleField>
</fields>
<messages>
  <define.SimpleMessage>
    <name>AUTH com mailbox arg</name>
    <fieldList>
      <string>status/append</string>
      <string>mailbox</string>
      <string>status/append arg</string>
    </fieldList>
    <initDelimiter>A02</initDelimiter>
```

```
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>SEL fetch/store seq-set msg-data-items</name>
  <fieldList>
    <string>fetch/store</string>
    <string>sequence A</string>
    <string>sequence B</string>
    <string>flags</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>AUTH -&gt; NAUTH</name>
  <fieldList>
    <string>transition logout</string>
  </fieldList>
  <initDelimiter></initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>NAUTH com arg arg</name>
  <fieldList>
    <string>login</string>
    <string>login username</string>
    <string>login password</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
```

```
<define.SimpleMessage>
  <name>SEL search arg-pair arg-pair</name>
  <fieldList>
    <string>search</string>
    <string>criteria key</string>
    <string>criteria value</string>
    <string>criteria key</string>
    <string>criteria value</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>AUTH -&gt; SEL</name>
  <fieldList>
    <string>transition select</string>
  </fieldList>
  <initDelimiter></initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>SEL com</name>
  <fieldList>
    <string>check/close/expunge/capability...</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>SEL -&gt; AUTH</name>
  <fieldList>
```

```
    <string>transition close</string>
  </fieldList>
  <initDelimiter></initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>NAUTH com</name>
  <fieldList>
    <string>starttls/capability/noop/logout</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>NAUTH -&gt; AUTH</name>
  <fieldList>
    <string>transition login</string>
  </fieldList>
  <initDelimiter></initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>AUTH com ref ref-arg</name>
  <fieldList>
    <string>reference </string>
    <string>ref arg</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
```

```
  <name>NAUTH com arg</name>
  <fieldList>
    <string>authenticate</string>
    <string>authenticate arg</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>SEL search arg-pair</name>
  <fieldList>
    <string>search</string>
    <string>criteria key</string>
    <string>criteria value</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>AUTH com</name>
  <fieldList>
    <string>capability/noop/logout</string>
  </fieldList>
  <initDelimiter>A02</initDelimiter>
  <endDelimiter>\r\n</endDelimiter>
</define.SimpleMessage>
<define.SimpleMessage>
  <name>AUTH com arg</name>
  <fieldList>
    <string>select ... mailbox</string>
    <string>mailbox</string>
```

```
    </fieldList>
    <initDelimiter>A02</initDelimiter>
    <endDelimiter>\r\n</endDelimiter>
  </define.SimpleMessage>
</messages>
<globalVariables/>
<states>
  <define.SimpleState>
    <name>selected state</name>
    <auth>false</auth>
    <connected>false</connected>
    <endState>false</endState>
    <localVariables/>
    <transitions>
      <define.SimpleTransition>
        <toState>selected state</toState>
        <messages>
          <define.SimpleTransitionMessage>
            <name>SEL com</name>
            <fieldValues/>
            <launchEvent/>
          </define.SimpleTransitionMessage>
        </messages>
        <conditions/>
        <error>false</error>
      </define.SimpleTransition>
      <define.SimpleTransition>
        <toState>selected state</toState>
        <messages>
          <define.SimpleTransitionMessage>
            <name>SEL search arg-pair</name>
```

```
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>selected state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>SEL search arg-pair arg-pair</name>
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>selected state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>SEL fetch/store seq-set msg-data-items</name>
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
```

```
    <define.SimpleTransition>
      <toState>authenticated state</toState>
      <messages>
        <define.SimpleTransitionMessage>
          <name>SEL -&amp;gt; AUTH</name>
          <fieldValues/>
          <launchEvent/>
        </define.SimpleTransitionMessage>
      </messages>
      <conditions/>
      <error>false</error>
    </define.SimpleTransition>
    <define.SimpleTransition>
      <toState>not authenticated state</toState>
      <messages>
        <define.SimpleTransitionMessage>
          <name>AUTH -&amp;gt; NAUTH</name>
          <fieldValues/>
          <launchEvent/>
        </define.SimpleTransitionMessage>
      </messages>
      <conditions/>
      <error>false</error>
    </define.SimpleTransition>
  </transitions>
</define.SimpleState>
<define.SimpleState>
  <name>authenticated state</name>
  <auth>false</auth>
  <connected>false</connected>
  <endState>false</endState>
```

```
<localVariables/>
<transitions>
 <define.SimpleTransition>
  <toState>authenticated state</toState>
  <messages>
   <define.SimpleTransitionMessage>
    <name>AUTH com</name>
    <fieldValues/>
    <launchEvent/>
   </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
 </define.SimpleTransition>
 <define.SimpleTransition>
  <toState>authenticated state</toState>
  <messages>
   <define.SimpleTransitionMessage>
    <name>AUTH com arg</name>
    <fieldValues/>
    <launchEvent/>
   </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
 </define.SimpleTransition>
 <define.SimpleTransition>
  <toState>authenticated state</toState>
  <messages>
   <define.SimpleTransitionMessage>
    <name>AUTH com ref ref-arg</name>
```

```
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>authenticated state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>AUTH com mailbox arg</name>
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>selected state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>AUTH -&amp;gt; SEL</name>
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
```

```
    <define.SimpleTransition>
      <toState>not authenticated state</toState>
      <messages>
        <define.SimpleTransitionMessage>
          <name>AUTH -&gt; NAUTH</name>
          <fieldValues/>
          <launchEvent/>
        </define.SimpleTransitionMessage>
      </messages>
      <conditions/>
      <error>false</error>
    </define.SimpleTransition>
  </transitions>
</define.SimpleState>
<define.SimpleState>
  <name>not authenticated state</name>
  <auth>false</auth>
  <connected>false</connected>
  <endState>false</endState>
  <localVariables/>
  <transitions>
    <define.SimpleTransition>
      <toState>not authenticated state</toState>
      <messages>
        <define.SimpleTransitionMessage>
          <name>NAUTH com</name>
          <fieldValues/>
          <launchEvent/>
        </define.SimpleTransitionMessage>
      </messages>
      <conditions/>
```

```
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>not authenticated state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>NAUTH com arg</name>
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>not authenticated state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>NAUTH com arg arg</name>
      <fieldValues/>
      <launchEvent/>
    </define.SimpleTransitionMessage>
  </messages>
  <conditions/>
  <error>false</error>
</define.SimpleTransition>
<define.SimpleTransition>
  <toState>authenticated state</toState>
  <messages>
    <define.SimpleTransitionMessage>
      <name>NAUTH -&gt; AUTH</name>
```

```
            <fieldValues/>
            <launchEvent/>
           </define.SimpleTransitionMessage>
          </messages>
          <conditions/>
          <error>false</error>
         </define.SimpleTransition>
       </transitions>
     </define.SimpleState>
   </states>
   <events/>
</define.SimpleProtocol>
```