Ajuda al desenvolupament de protocols de comunicació

Joan Romera Esteban
Grau d'Enginyeria Informàtica
Àrea de treball de Compiladors

Gerard Enrique Manonellas
Robert Clarisó Viladrosa

19 de juny de 2017

GNU Free Documentation License (GNU FDL)

Copyright

FITXA DEL TREBALL FINAL

Títol del treball:	*Ajuda al desenvolupament de protocols de comunicació*
Nom de l'autor:	*Joan Romera Esteban*
Nom del consultor/a:	*Gerard Enrique Manonellas*
Nom del PRA:	*Robert Clarisó Viladrosa*
Data de lliurament (mm/aaaa):	*06/2017*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Compiladors*
Idioma del treball:	*Català*
Paraules clau	*Compiladors, Protocols, Metallenguatges*

Resum del Treball:

La finalitat d'aquest treball és demostrar la viabilitat d'utilitzar eines genèriques de compiladors de compiladors per ajudar al desenvolupament de protocols de comunicació. El context d'aplicació és per a totes aquells desenvolupadors i companyies que tenen la necessitat continua d'integració amb aplicacions i sistemes d'altres companyies. La metodologia plantejada és basa en definir un metallenguatge que, per mitjà de metaprogramació, generi el codi amb la implementació de qualsevol protocol de comunicació. Els resultats han provat la viabilitat d'aquest plantejament i la conclusió final és que aquest ha demostrat ser un mètode de desenvolupament altament productiu.

Abstract:

The purpose of this study is to demonstrate the feasibility of using generic tools like compilers-compilers to help develop communication protocols. The application context is for all those developers and companies that have the need of continuous integration with applications and systems of other companies. The proposed methodology is based on a metalanguage, through metaprogramming, which generates the code to implement any communication protocol. The results have proved the feasibility of this approach and the conclusion is that this method has proven to be a highly productive development method.

Índex de continguts

Introducció

La motivació principal per a desenvolupar aquest projecte és que, per feina, porto força anys implementant molts protocols de comunicació de forma *ad hoc*, el que sol ser una tasca feixuga. Aquesta dificultat també inclou les proves, ja que els protocols de comunicació al ser quelcom dinàmic i que depèn del temps, també sol ser una etapa llarga i problemàtica. A més, al no ser una programació genèrica i que tampoc és orientada a objectes, comporta molta duplicació de codi.

La majoria de protocols amb que em trobo acostumen a ser protocols textuals propietaris. Dins del sector on programo, hostaleria i restauració, es troben pocs estàndards. De fet, els estàndards solen ser grans, complexes, i, mols cops, són protocols industrials i binaris. Això, provoca que es descartin per ésser inviables, econòmicament parlant, degut a la seva dificultat d'implementació. Encara que també és cert que, atès que són estàndards, es poden trobar disponibles implementacions ja existents, però no sempre es poden usar per qüestions del seu llenguatge de programació o per llicències privatives amb un alt cost.

Aleshores, per accelerar la implementació d'aquests protocols, la idea principal es basa en la semblança dels protocols de comunicació amb els llenguatges de programació. De fet, aquesta analogia evident, també es menciona a la Viquipèdia en diferents idiomes: Protocols and programming languages. És necessari parsejar i validar una entrada, tal i com es realitza en la compilació de programes, És a dir, existeixen unes regles lèxiques, sintàctiques i semàntiques.

I, per tant, basant-me en aquest fet, la meva proposta és utilitzar eines genèriques de compilació de llenguatges, com JFlex i CUP o Xtext en Java.

L'objectiu principal del projecte és demostrar que aquesta idea és possible i econòmicament viable. I per a demostrar-ho, caldrà una primera implementació real o prototip amb uns certs mínims, la qual ha de consistir només en automatitzar amb una eina de compilació la majoria de característiques dels protocols, principalment textuals.

D'aquí, se n'extreu un altre objectiu implícit, el qual és obtenir aquest primer exemple d'implementació, però insisteixo que l'objectiu real ha de ser arribar a una conclusió definitiva sobre la certesa d'aquesta teoria, sigui positiva o negativa.

Un cop acomplert, un segon objectiu desitjable és aconseguir dissenyar també un metallenguatge (DSL) per a la definició dels protocols, el qual hauria de generar de forma automàtica la implementació de la fita anterior. Així, un cop assolit, s'obté una eina que permet generar un component per a qualsevol aplicació que necessiti integrar-se amb un sistema extern qualsevol que implementi un protocol concret.

Però el que no es pretén és crear un protocol universal, al igual que no existeix un idioma universal per a les persones, sinó «*ensenyar a parlar i entendre*» un protocol específic, per part d'una màquina o aplicació.

Per altra part, també es manifesta la possibilitat d'un tercer objectiu, el qual és totalment opcional, depenent de l'evolució del projecte i de les seves necessitats. Aquesta possibilitat és crear una aplicació intermediària, a mode de capa façana, entre els diferents sistemes. Seria com una espècie de concentrador, on es gestionarien les comunicacions dels diferents sistemes heterogenis. Això permetria reduir l'acoblament entre aquests.

En quant a la viabilitat parcial o total del projecte, el propòsit és utilitzar el mètode MoSCoW per a delimitar els objectius i els seus requisits, tal com es podrà llegir en l'apartat següent.

Els beneficis de la solució proposada són, en ordre d'importància i en base només a les dues primeres fites:

- Ajudar i accelerar el desenvolupament de protocols, una tasca feixuga tant en la programació com en les proves posteriors i l'anàlisi prèvia.
- Estalviar costos d'implementació, proves i desplaçaments a casa del client.
- Evitar errors d'implementació.
- Tenir fàcil accés als dos costats de les comunicacions per a les proves d'integració.
- No estar subjecte als defectes i inconvenients de la documentació d'un protocol, relacionats, respectivament, amb les habilitats humanes de redacció i interpretació.
- Descripció formal dels protocols intel·ligible per a màquines.
- Independència de l'idioma de la documentació informal, sigui anglès, alemany, francès, italià, etc.
- Donar, a més de la documentació textual, una o vàries implementacions en diferents llenguatges de programació per a ús directe o com a exemple d'implementació.
- Comparació fàcil entre protocols suposadament diferents o les seves diferents versions, per a trobar la igualtat o les seves diferències, independentment del format de les seves documentacions textuals.
- Facilitar el disseny, documentació i desenvolupament de nous protocols. En algunes ocasions, nosaltres hem de subministrar el protocol. Això facilitaria la nostra feina (disseny, documentació, desenvolupament) com la del receptor de la documentació (interpretació, implementació i proves).

I els avantatges de la tercera opció com a aplicació independent són:

- Reduir l'acoblament entre sistemes.
- Cobrir la necessitat d'una aplicació que pot requerir comunicar-se amb més d'un sistema extern.
- Possibilitar que una implementació d'un protocol pugui ser usada en més d'una aplicació sense arribar a duplicar el codi font.
- Permetre que els processos de comunicació corrin en segon pla, sense dependre de cap execució de l'aplicació amfitriona.
- Subministrar utilitats comunes entre protocols i aplicacions com les següents:

    - Configuració del canal físic. Per exemple, l'adreça IP i port TCP del servidor del sistema extern, o els paràmetres de la comunicació sèrie com poden ser el número del port connectat, els bauds, la paritat, etc.
    - El visor del fitxer del registre de les comunicacions.
    - Control dels processos de comunicació: aturada, engegada, estat, canvi del detall del fitxer de *log*, etc.
    - Testeig fàcil en entorns de producció en cas de problemes de comunicació (i ajudar també a les proves en temps de desenvolupament).
- Solucionar problemes amb la deslocalització de les comunicacions. Deguts, per exemple, a la llargada màxima permesa del cable sèrie o arxius seqüencials de comunicació residents en altres màquines. Aquests inconvenients provoquen que la comunicació només sigui possible des de màquines diferents a la qual on resideix l'aplicació amfitriona.
- Possibilitar asincronia virtual mitjançant cues d'entrada i/o sortida.

Els protocols amb que es treballa pertanyen a les capes d'aplicació, presentació i sessió, segons el model OSI, o a la capa d'aplicació, segons el *protocol stack* d'Internet (Layer names and number of layers in the literature). De totes formes, si en el futur s'afegeix la possibilitat de generar codi objecte en altres llenguatges de programació (mitjançant codi intermedi), sobretot C, això no hauria d'impedir treballar en capes inferiors.

Finalment, els *stakeholders* interessats en aquesta solució són, en primera instància, qualsevol desenvolupador/companyia de programari amb la necessitat constant d'integració amb altres sistemes i, en segon lloc, les companyies d'aquests altres sistemes per a facilitar la integració amb els primers.

Encara que el perfil de desenvolupador especialitzat en el desenvolupament de protocols és limitat, per altra part, companyies de *software* amb requisits d'integració són moltes, sinó la majoria.

Fer notar que, en el món dels protocols propietaris, moltes companyies podrien estar interessades a no utilitzar aquest tipus d'eines, per tal d'evitar la competència deslleial per part d'altres companyies competidores. En alguns sectors, es troben documentacions diferents de protocols d'altres empreses, però que, en realitat, acaben sent el mateix protocol. És a dir, si una empresa competidora és «*compatible*» amb el protocol d'una altra, té més oportunitats d'aconseguir una venda i prendre-li a la original.

Aquesta observació es pot deduir del següent text en la Viquipèdia (Protocol development):

«*For communication to take place, protocols have to be agreed upon. Recall that in digital computing systems, the rules can be expressed by algorithms and datastructures, raising the opportunity for hardware independence. Expressing the algorithms in a portable programming language, makes the protocol*

software operating system independent. ***The source code could be considered a protocol specification. This form of specification, however is not suitable for the parties involved.***

For one thing, this would enforce a source on all parties and for another, ***proprietary software producers would not accept this.*** *By describing the software interfaces of the modules on paper and agreeing on the interfaces, implementers are free to do it their way. This is referred to as source independence. By specifying the algorithms on paper and detailing hardware dependencies in an unambiguous way* [...]»

De fet, la majoria de documentacions porten un avís de confidencialitat i, a vegades, cal la signatura d'un acord *ex professo* de confidencialitat o, en anglès, NDA (*Non-Disclosure Agreement*).

I, per acabar, en el món dels protocols estàndards i oberts, cap també la possibilitat de l'adopció d'una eina com aquesta com, per exemple, amb l'HTTP. Amb tot això, aquesta possibilitat s'apunta com a més remota. Malgrat això, es troben exemples actuals com l'ASN.1 (*Abstract Syntax Notation One*), on el qual, mitjançant un compilador de fitxers escrits amb aquesta sintaxi, genera fonts en un llenguatge de programació concret. Encara que, en aquest cas, es treballa només sobre la capa de presentació OSI, la serialització i deserialització de les estructures de dades en memòria.

Pla de treball

Fites principals del projecte

Tal com es pot llegir en l'apartat anterior, per tenir en ment la viabilitat parcial o total del projecte, el propòsit és utilitzar el mètode MoSCoW per a delimitar els objectius i els seus requisits.

Per evitar ser una proposta massa o poc ambiciosa, i sabent com és la realitat dels projectes informàtics i les restriccions de temps, es tenen en compte les dues dimensions principals del programari: el temps i l'abast. La experiència demostra que una està en funció de l'altra i no té sentit intentar aconseguir quadrar totes dues.

Llavors, s'ha previst prioritzar els requisits de l'abast del projecte en nivells:

1. Ha de tenir: tot allò mínim, bàsic, imprescindible, obligatori.
2. Hauria de tenir: el desitjable, nivell mig, quasi imprescindible.
3. Podria tenir: el perfecte, ideal, però prescindible, totalment opcional.
4. No es farà ara: no hi cap, queda fora de l'abast.

És a dir, classificar-los en els requisits mínims imprescindibles, en els desitjables, en els prescindibles i en els que segur que no es realitzaran per al lliurament del TFG. Així, segons l'evolució del desenvolupament i el temps que vagi quedant disponible en acabat els obligatoris, el projecte és menys rígid i pressionant i més flexible i realista.

El plantejament és reduir els requisits a tals mínims (primer nivell) que, si tot va bé, s'haurien d'aconseguir en el primer lliurament parcial. Després un segon grup de requisits (segon nivell) en el segon lliurament parcial i, l'últim (tercer nivell) en el lliurament definitiu. Però, en el pitjor dels casos, es reserven els dos últims lliuraments per aconseguir el primer nivell dels requisits, i considerar encara així el projecte exitós.

Per tot això, s'estableixen dues fites principals, corresponents als dos primers nivells i als dos primers lliuraments parcials, respectivament:

1. Automatitzar amb (JFlex i CUP o Xtext) la majoria de característiques de protocols principalment textuals, sobretot la part de baix nivell. Això hauria d'incloure també els tests de la implementació resultant.
2. Crear un metallenguatge (*Domain Specific Language*) per a la definició de protocols, el qual generaria amb metaprogramació els fitxers fonts de JFlex i CUP o Xtext de la fita anterior. Evidentment, aquest llenguatge es compilaria amb JFlex i CUP o Xtext.

Per al tercer nivell i lliurament definitiu, s'estableix la següent fita opcional:

3. Crear una interfície gràfica d'usuari per a dissenyar el metallenguatge.

   Addicionalment, es podrien crear també pantalles auxiliars de gestió, altres temes d'arquitectura (veure pròxim apartat) i com aplicació independent.

Per al quart i últim nivell, es decideix no incloure:

4. La generació de codi intermedi, per a realitzar un «*cross-compiler*» i poder generar codi final en altres llenguatges de programació com C, per exemple.

Tasques i subtasques

A continuació, s'enumera la llista de requisits, funcionals i no funcionals, ordenats cronològicament, per importància i per dependències entre requisits.

1 Primera fita/lliurament
  1.1 Recopilació d'informació prèvia
    1.1.1 Cerca d'eines existents semblants
    1.1.2 Elecció de l'eina genèrica de compilació: JFlex/CUP o Xtext
    1.1.3 Caracterització principal dels protocols
    1.1.4 Disseny del protocol pels jocs de proves
    1.1.5 Trobar les diferències i matisos de la compilació de llenguatges de programació respecte els protocols de comunicació per a tenir en compte després en el desenvolupament
  1.2 Preparació prèvia de l'entorn de desenvolupament
  1.3 Implementació del protocol de proves
    1.3.1 Decisions prèvies d'anàlisi
    1.3.2 Separació en dues capes: alt nivell (*payload*) i baix nivell (control de flux i *framing*)
    1.3.3 *Logging*
    1.3.4 Bloquejos
  1.4 Eines de test
  1.5 Conclusions

2 Segona fita/lliurament
  2.1 Disseny del metallenguatge
  2.2 Requisits opcionals dins d'aquesta fita
    2.2.1 Disseny com a component
    2.2.2 Configuració
    2.2.3 Comptador seqüencial de missatges
    2.2.4 Relacionats amb el disseny com a aplicació
      2.2.4.1 Cua de sortida
      2.2.4.2 Procés en segon pla

3 Tercera fita/lliurament
  3.1 Interfície gràfica d'usuari per a la definició de protocols amb el metallenguatge

3.2 Internacionalització per les interfícies gràfiques d'usuari

3.3 Requisits opcionals dins d'aquesta fita i relacionats amb el disseny com a aplicació

3.3.1 Arquitectura

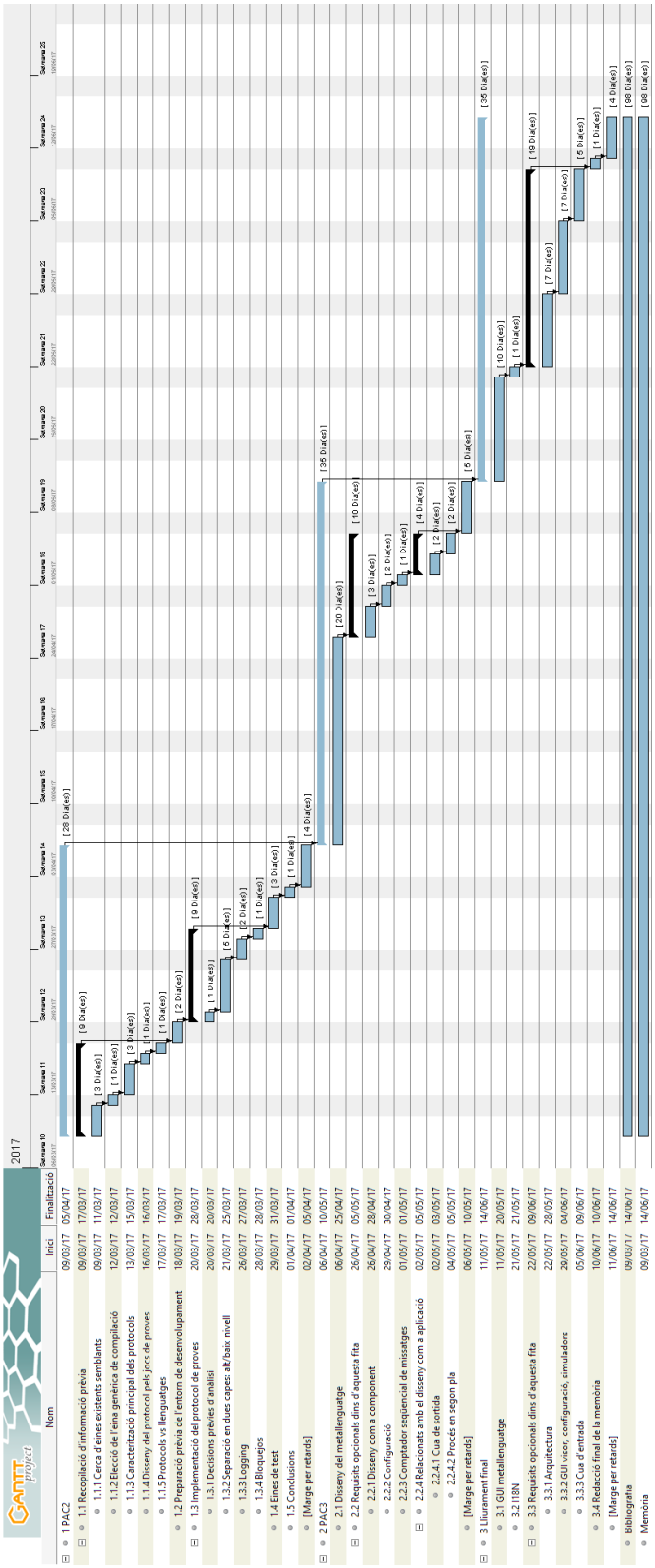
3.3.2 Interfícies gràfiques d'usuari per al visor, configuració, simuladors (en ordre de desenvolupament)

3.3.3 Cua d'entrada

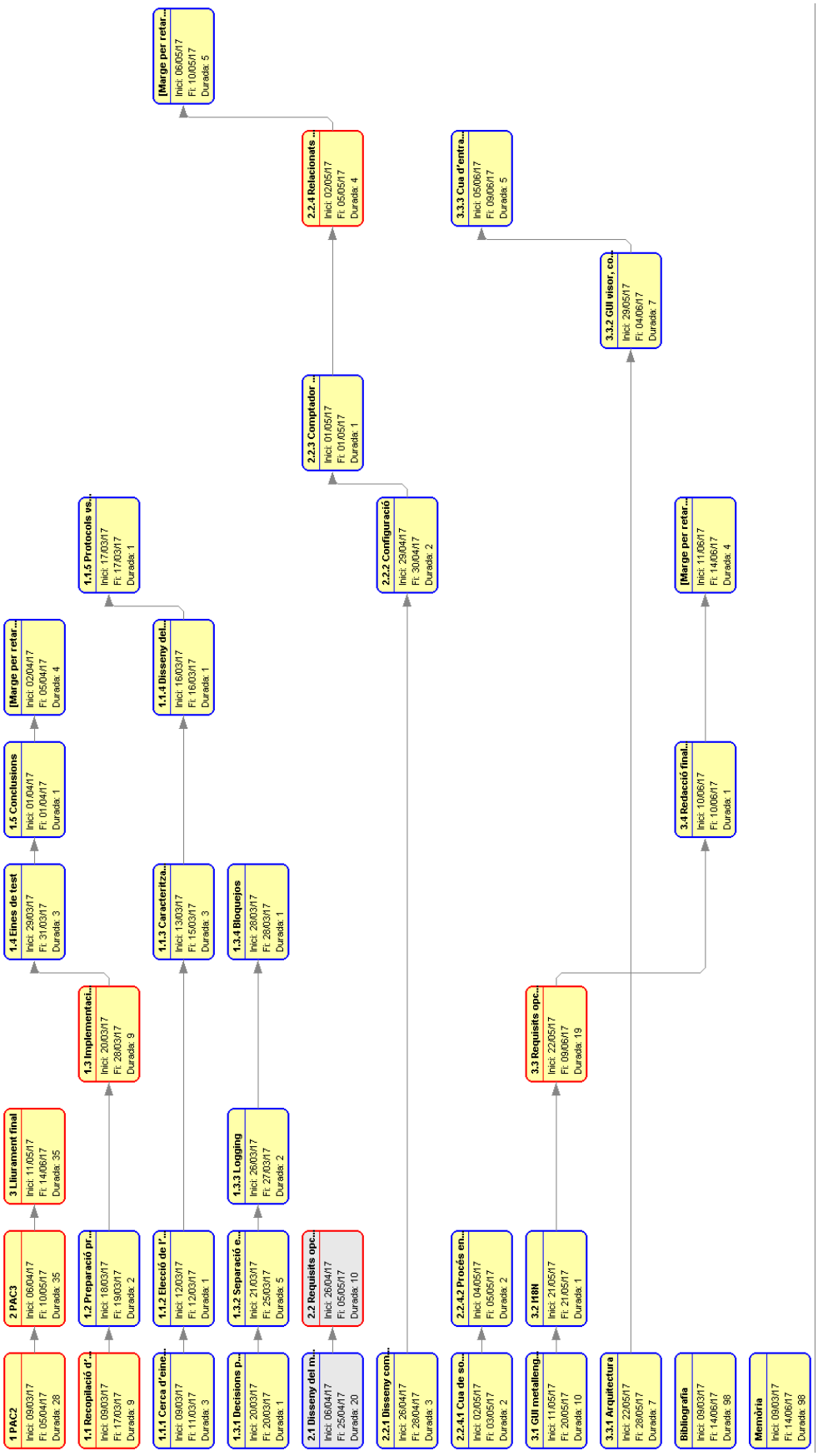
3.4 Redacció final de la memòria del projecte

Planificació

Aquest diagrama de Gantt mostra les dates i duracions orientatives en jornades de les tasques i subtasques:



I aquest altre diagrama PERT mostra les dependències entre les tasques i subtasques:



Metodologia de treball i model de desenvolupament

Aquest projecte s'ha de considerar més dins de les àrees de recerca, sobretot en les primeres etapes, que dins de les àrees d'un producte final, més a prop de les darreres etapes. Això dificulta el plantejament típic d'altres projectes més orientats cap a aplicacions d'usuari final.

En tot cas, per una part, ja s'ha remarcat l'ús del mètode MoSCoW com a tècnica de priorització de requisits. I d'alguna manera, aquesta forma de plantejament ja comporta una gestió de riscos implícita, al assumir diferents gradients possibles de l'abast segons evolucioni el projecte i el temps disponible.

I per una altra banda, la idea és utilitzar un model de desenvolupament iteratiu i incremental amb refinaments successius i establint versions parcials i estables, però sense entrar en cap model concret de cicle de vida d'enginyeria del programari. El propi pla de treball s'utilitzarà com a *taskboard* i guió de desenvolupament de tot el projecte.

Implementació del protocol de proves

Introducció

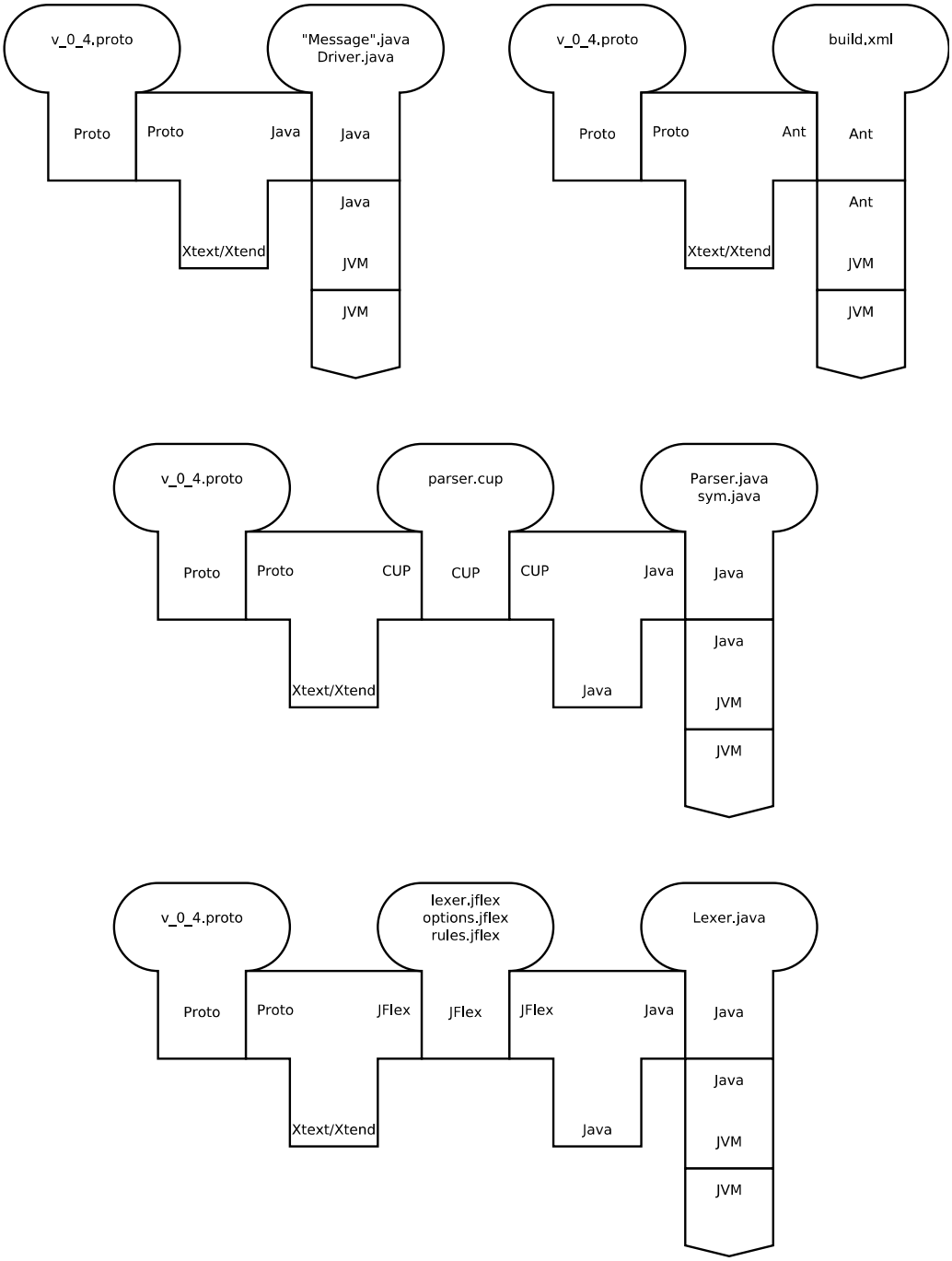
En aquest capítol, avançarem la descripció de la solució final amb la implementació del metallenguatge de la primera versió del protocol de proves. La idea és introduir al lector a l'objectiu de tot el projecte d'una manera més amena.

En els dos capítols següents, ja s'entrarà en el detall de tot el desenvolupament del projecte. Primerament, amb un capítol amb la descripció de la fase inicial de la recerca d'informació i, després, en un altre amb el relat de la implementació del codi.

A continuació, en el capítol actual, primer es relatarà el procediment seguit per a implementar el protocol de proves, junt amb el codi resultant dels fitxers generats amb la especificació del protocol. Més tard, es trobaran varis diagrames i figures que complementaran visualment la descripció del projecte resultant de la compilació del protocol.

Diagrames de Tombstone

Aquest és un esbós del funcionament de la compilació per a cada tipus de fitxer. S'han separat segons cada tipus per simplificar la figura. Cada diagrama es correspon als fitxers Java generats directament, les especificacions del analitzador lèxic i sintàctic i el programa compilador d'aquests en Ant.



Especificació informal del protocol de proves

Avancem ara les característiques de la primera versió del protocol:

- A nivell bàsic, es caracteritza per ser un protocol monoclient unidireccional per xarxa, actuant com a client TCP, pel port per defecte de la configuració (8.000) i amb connexió permanent. Cada missatge enviat pel servidor finalitza en un retorn de carro i salt de línia (CR+LF de Windows).
- A alt nivell, és la típica sortida d'impressora que envia els càrrecs d'un altre sistema, com podria ser un TPV, per exemple. Aquests són els camps definits per cada càrrec:
  - Codi d'article: de tipus enter de dos dígits
  - Descripció d'article: de tipus alfanumèric de 10 caràcters
  - Data i hora del càrrec: de tipus data i hora
  - IVA inclòs: de tipus booleà
  - Import: de tipus real de 6 dígits, inclosos el punt decimal i els dígits decimals

Creació del projecte

Es detallen, a continuació, els primers passos per a crear el projecte inicial per a poder implementar el protocol acabat d'especificar:

1. **Instal·lació** de l'entorn de desenvolupament, ja amb tot configurat i funcionant:
    1. Descarregar l'arxiu UOC_TFG_jromerae.zip (525 Mbs aprox.).
    2. Requisits d'instal·lació:
        1. Windows (64 bits)
        2. JDK de Java 8 (64 bits) instal·lat
        3. Descomprimir el contingut directament a l'arrel de "C:\".

           Això és necessari per a evitar problemes amb els *paths* absoluts.
2. Obrir l'entorn, executant l'arxiu "eclipse.bat", el qual es pot trobar dins de la carpeta recentment creada «UOC_TFG_jromerae».
3. **Obrir** el segon entorn per a implementar el protocol, amb el botó dret en el «Package Explorer / dsl / edu.uoc.interfaceman.dsl» i clicant en l'opció de menú «Run As / Eclipse Application».
4. **Creació** del projecte inicial:

   El projecte ja està tot creat, per si es vol evitar seguir els següents passos. En cas contrari, esborrar-lo abans.
    1. Clicar en l'opció de menú «File / New / Java Project», indicar el nom «cup.v_x_y» i finalitzar directament amb el botó «Finish».
    2. Clicar en l'opció de menú «File / New / Source Folder», indicar el nom «src-gen» i finalitzar directament amb el botó «Finish».
    3. Clicar en l'opció de menú «File / New / Folder», indicar el nom «lib» i finalitzar directament amb el botó «Finish».
    4. Clicar en l'opció de menú «File / New / Package», indicar el nom «edu.uoc.interfaceman.cup.v_0_4» i finalitzar directament amb el botó «Finish».
    5. Executar el mateix un altre cop, però ara sobre la carpeta «src-gen», en lloc de sobre la «src».
5. **Còpia** de recursos disponibles al primer entorn:
    1. Dependències:
        1. «Other Projects / interfaceman / interfaceman-runtime.jar» a carpeta «lib».

           Clic amb botó dret «Build Path / Add to Build Path».
        2. «Other Projects / interfaceman / lib / java-cup-11b-runtime.jar» a carpeta «lib».

           Clic amb botó dret «Build Path / Add to Build Path».
        3. «Other Projects / interfaceman / lib / java-cup-11b.jar» a carpeta «lib».
        4. «Other Projects / interfaceman / lib / jflex-1.6.1.jar» a carpeta «lib».

5. «Other Projects / interfaceman / lib / log4j-api-2.8.2.jar» a carpeta «lib».

   Clic amb botó dret «Build Path / Add to Build Path».

6. «Other Projects / interfaceman / lib / log4j-core-2.8.2.jar» a carpeta «lib».

   Clic amb botó dret «Build Path / Add to Build Path».

2. «Other Projects / interfaceman / resources / log4j2.xml» a carpeta «src».

Especificació formal del protocol de proves

Definirem ara el fitxer amb l'especificació del protocol en el metallenguatge.

Clicar amb el botó dret sobre «cup.v_x_y / src / edu.uoc.interfaceman.cup.v_0_4», triar opció de menú «New / File», indicar el nom «v_0_4.proto» i finalitzar directament amb el botó «Finish».

El contingut a copiar és el que es troba en la següent secció.

v_0_4.proto

```
package   edu.uoc.interfaceman.cup.v_0_4

channel   tcp_client

etx       CR LF

field     code        integer     2     "Codi d'article."
field     description string      10    "Descripció d'article."
field     dataHora    datetime          "Data i hora del càrrec."
field     VAT         bool              "IVA inclòs."
field     amount      real        6     "Import del càrrec."

message   Charge "Missatge que representa a un càrrec."
          code description dataHora VAT amount {
                logger.info(PATTERN, MESSAGE, "New charge received");
          }
```

Generació dels fitxers d'implementació

En quant gravem el contingut del fitxer anterior, automàticament ens crearà els següents fitxers a «src-gen / edu.uoc.interfaceman.cup.v_0_4»:

- **Charge.java**: classe per representar un càrrec rebut.
- **Driver.java**: classe que conté el programa principal que inicia el procés de comunicació.
- **parser.cup**: especificació sintàctica de la gramàtica per CUP.
- **lexer.jflex**: especifica lèxica de la gramàtica per JFlex.
- **options.jflex**: opcions comuns per a lexer.jflex que no depenen directament del protocol.
- **rules.jflex**: regles comuns per a lexer.jflex que no depenen directament del protocol.

També es genera un altre fitxer **build.xml** a la carpeta «src-gen» directament. Aquest és un script Ant que es pot executar directament, fent-hi clic a sobre amb el botó dret i clicant en l'opció de menú «Run As / Ant Build».

Llavors, a partir dels arxius de CUP i JFlex creats abans, aquest script generarà els fonts Java que resten amb la implementació:

- **Lexer.java**: analitzador lèxic JFlex pel protocol.
- **Parser.java**: analitzador sintàctic CUP pel protocol.
- **sym.java**: taula de símbols o testimonis usada per ambdós analitzadors.

Amb tot això, ja tenim tots els arxius fonts amb el protocol ja implementat a partir del fitxer amb la especificació.

A continuació, es transcriu el contingut dels fitxers generats més interessants: lexer.jflex, parser.cup, Charge.java i Driver.java. Després d'això, es descriurà el procés per a provar la implementació recentment creada.

lexer.jflex

```
package edu.uoc.interfaceman.cup.v_0_4;

import java_cup.runtime.SymbolFactory;
import java_cup.runtime.Symbol;
import java.io.IOException;
import java.time.*;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import static edu.uoc.interfaceman.runtime.Category.*;
import edu.uoc.interfaceman.runtime.channels.Channel;
import edu.uoc.interfaceman.runtime.*;

%%

// Opcions comuns per a tots els lexers
%include options.jflex

etx = \r\n

%states F2 F3 F4 F5

%%

<YYINITIAL>   {int}{2}  { yybegin(F2);  return newSymbol(INTEGER, new Integer(yytext().trim())); }
<F2>          {char}{10}{ yybegin(F3);  return newSymbol(STRING,  yytext()); }
<F3>          {datetime}{ yybegin(F4);  return newSymbol(DATETIME,LocalDateTime.parse(yytext(),
                                                         Message.FORMATTER_DATETIME)); }
<F4>          {boolean} { yybegin(F5);  return newSymbol(BOOLEAN, "1".equals(yytext())); }
```

```
<F5>          {real}{6} { yybegin(FE); return newSymbol(REAL,   new Float(yytext())); }
<FE>          {etx}     { yybegin(YYINITIAL); return newSymbol(ETX); }

.                   { yybegin(ERROR);
                          logger.error(PATTERN,   Category.ERROR,   String.format("Wrong
character (ignored): '%s'", yytext())); }
  // Regles comuns per a tots els lexers
  %include rules.jflex
```

parser.cup

```
  package edu.uoc.interfaceman.cup.v_0_4;

  import java_cup.runtime.*;
  import edu.uoc.interfaceman.cup.v_0_4.Lexer;
  import edu.uoc.interfaceman.cup.v_0_4.Charge;
  import edu.uoc.interfaceman.runtime.channels.Channel;
  import edu.uoc.interfaceman.runtime.Listener;
  import java.time.*;

  parser code {:
       Listener listener;

       public Parser(Channel channel, Listener listener) throws Exception {
              super();
              channel.run(() -> {
                     symbolFactory = new ComplexSymbolFactory();
                     setScanner(new Lexer(channel, symbolFactory));
                     this.listener = listener;
                     parse();
              });
       }
  :};

  terminal String         STRING;
  terminal Boolean        BOOLEAN;
  terminal Integer        INTEGER;
  terminal Float          REAL;
  terminal LocalDate      DATE;
  terminal LocalTime      TIME;
  terminal LocalDateTime  DATETIME;
  terminal                ETX;

  non terminal            LIST;
  non terminal Charge     CHARGE;

  LIST ::=      LIST    CHARGE:charge    {: listener.onMessage(charge); :}
                |       CHARGE:charge    {: listener.onMessage(charge); :};
  CHARGE ::=    INTEGER:code STRING:description DATETIME:dataHora BOOLEAN:VAT REAL:amount ETX
                       {: RESULT = new Charge(code, description, dataHora, VAT, amount); :};
```

Charge.java

```
  package edu.uoc.interfaceman.cup.v_0_4;

  import java.time.LocalDateTime;

  import edu.uoc.interfaceman.runtime.Message;

  /**
   * Missatge que representa a un càrrec.
   */
  public class Charge extends Message {

       /** Inici de missatge. */
       public static final String STX = "";
       /** Fi de missatge. */
       public static final String ETX = "\r\n";
```

```
        /** Codi d'article. */
        public Integer code;
        /** Descripció d'article. */
        public String description;
        /** Data i hora del càrrec. */
        public LocalDateTime dataHora;
        /** IVA inclòs. */
        public Boolean VAT;
        /** Import del càrrec. */
        public Float amount;

        /**
         * Nou càrrec.
         *
         * @param code codi d'article.
         * @param description descripció d'article.
         * @param dataHora data i hora del càrrec.
         * @param VAT IVA inclòs.
         * @param amount import del càrrec.
         */
        public Charge(Integer code, String description, LocalDateTime dataHora,
                              Boolean VAT, Float amount) {
              this.code           = code;
              this.description    = description;
              this.dataHora       = dataHora;
              this.VAT            = VAT;
              this.amount         = amount;
        }

        @Override
        public String stx() {
              return STX;
        }

        @Override
        public String etx() {
              return ETX;
        }

        @Override
        public String format() {
              return new StringBuilder  (format(code, 2)).append
                                        (format(description, 10)).append
                                        (format(dataHora)).append
                                        (format(VAT)).append
                                        (format(amount, 6)).toString();
        }

        @Override
        public String toString() {
              return String.format("Charge [code=%s, description=%s, dataHora=
%s, VAT=%s, amount=%s]",
                                   code, description, dataHora, VAT, amount);
        }
  }
```

Driver.java

```
  package edu.uoc.interfaceman.cup.v_0_4;

  import static edu.uoc.interfaceman.runtime.Category.*;

  import java.io.IOException;
  import java.net.UnknownHostException;

  import org.apache.logging.log4j.LogManager;
```

```
import org.apache.logging.log4j.Logger;

import edu.uoc.interfaceman.runtime.Listener;
import edu.uoc.interfaceman.runtime.Message;
import edu.uoc.interfaceman.runtime.TCPClientConfig;
import edu.uoc.interfaceman.runtime.channels.TCPClient;

/**
 * Programa principal.
 */
public class Driver implements Listener {

    /** Logger d'ús comú per a totes les capes. */
    public static final Logger logger =
                        LogManager.getLogger(Driver.class.getSimpleName());

    public static void main(String[] args) throws Exception {
          new Driver(new TCPClientConfig());
    }

    /**
     * Inicia el procés del protocol.
     *
     * @param config propietats de configuració.
     * @throws UnknownHostException si no es pot determinar l'adreça IP
     *           del servidor.
     * @throws IOException si es produeix un error de comunicació.
     * @throws Exception si es produeix qualsevol altre error.
     */
    public Driver(TCPClientConfig config) throws UnknownHostException,
                                                 IOException, Exception {
          logger.info(PATTERN, CONTROL, "Starting communication");
          new Parser(new TCPClient(config), this);
          logger.info(PATTERN, CONTROL, "Stopped communication");
    }

    @Override
    public void onMessage(Message message) {
          logger.debug(PATTERN, MESSAGE, message);
          logger.info(PATTERN, MESSAGE, "New charge received");
    }
}
```

Prova del protocol implementat

Per a realitzar aquest procés, podem copiar els següents fitxers del primer Eclipse en marxa:

- **DriverTest.java**:
  1. Copiar de «Other Projects / interfaceman / src / edu.uoc.interfaceman.test.tests.cup.v_0_4» a «cup.v_x_y / src / edu.uoc.interfaceman.cup.v_0_4».
  2. Canviar la declaració package al valor suggerit per l'editor.
- **input.txt**:
  1. Copiar de «Other Projects / interfaceman / resources / edu.uoc.interfaceman.test.poc.runtime» a «cup.v_x_y / src / edu.uoc.interfaceman.cup.v_0_4».

El programa posa en marxa alhora el servidor i el client, on el primer enviarà el contingut del fitxer input.txt al client i després un missatge generat en memòria. Cal observar que, en el fitxer input.txt, s'ha introduït un error de forma expressa en l'últim missatge, on el camp de codi d'article és alfanumèric en lloc de numèric. Això es pretén utilitzar, per una part, per verificar la detecció del error per part del analitzador lèxic i, per l'altra, la recuperació automàtica de l'error i que sap processar el missatge enviat des de memòria. Tot això, es pot observar en el log de sortida.

Podem executar el programa de test, fent-hi clic a sobre amb el botó dret i clicant en l'opció de menú «Run As / Java Application».

En les seccions següents, es pot trobar el contingut d'aquests dos arxius i del log de sortida.

DriverTest.java

```
package edu.uoc.interfaceman.cup.v_0_4;

import java.time.LocalDateTime;

import edu.uoc.interfaceman.cup.v_0_4.Charge;
import edu.uoc.interfaceman.cup.v_0_4.Driver;
import edu.uoc.interfaceman.runtime.TCPClientConfig;
import edu.uoc.interfaceman.runtime.channels.Channel;
import edu.uoc.interfaceman.runtime.channels.TCPServerMono;
import static edu.uoc.interfaceman.runtime.Category.*;

public class DriverTest {

	public static void main(String[] args) throws Exception {
		// configuració comú pel servidor i el client
		TCPClientConfig config = new TCPClientConfig();

		Runnable task = () -> {
			// engega servidor:  espera client
			try (TCPServerMono channel = new TCPServerMono(config)) {
				// client connectat: envia missatges
				channel.send("bin/edu/uoc/interfaceman/cup/v_0_4/input.txt");
				channel.send(new Charge(4, "Item 4", LocalDateTime.now(), true, 0.85f));
			} catch (Exception e) {
				Channel.logger.error(PATTERN, ERROR, e);
			}
		};

		// engega servidor en un thread perquè es bloqueja esperant al client
		new Thread(task).start();
		// engega el client
		new Driver(config);
	}
}
```

input.txt

```
   1Item 1    11/01/2017 09:10:011 10.50
   2Item 2    12/02/2017 10:20:020 -2.00
   3Item 3    13/03/2017 11:30:031  3.25
  a Item a    13/03/2017 11:30:031  3.25
```

Observi's l'error comentat anteriorment en l'última línia.

Log de sortida

```
   2017-06-15 06:17:32,384 [main] INFO  Driver       - CONTROL      Starting communication
   2017-06-15 06:17:32,389 [main] INFO  Channel      - CONTROL      Opening channel
   2017-06-15 06:17:32,403 [main] INFO  Channel      - CONTROL      Opened channel
   2017-06-15 06:17:32,403 [ad-0] INFO  Channel      - CONTROL      Opened channel
   2017-06-15 06:17:32,405 [main] DEBUG Channel      - CONTROL      Acquiring lock
   2017-06-15 06:17:32,408 [main] DEBUG Channel      - CONTROL      Acquired lock
   2017-06-15 06:17:32,427 [main] TRACE Lexer        - FIELD        ' 1'
   2017-06-15 06:17:32,428 [main] TRACE Lexer        - FIELD        'Item 1    '
   2017-06-15 06:17:32,431 [main] TRACE Lexer        - FIELD        '11/01/2017 09:10:01'
   2017-06-15 06:17:32,431 [main] TRACE Lexer        - FIELD        '1'
   2017-06-15 06:17:32,431 [main] TRACE Lexer        - FIELD        ' 10.50'
   2017-06-15 06:17:32,432 [main] TRACE Lexer        - DELIMITER    <CR><LF>
   2017-06-15 06:17:32,432 [main] TRACE Lexer        - FIELD        ' 2'
   2017-06-15 06:17:32,432 [ad-0] INFO  Channel      - CONTROL      Closed channel
   2017-06-15 06:17:32,432 [ad-0] INFO  Channel      - CONTROL      Closed channel
   2017-06-15 06:17:32,432 [main] DEBUG Driver       - MESSAGE      Charge         [code=1,
description=Item 1    , dataHora=2017-01-11T09:10:01, VAT=true, amount=10.5]
   2017-06-15 06:17:32,432 [main] INFO  Driver       - MESSAGE      New charge received
   2017-06-15 06:17:32,432 [main] TRACE Lexer        - FIELD        'Item 2    '
   2017-06-15 06:17:32,433 [main] TRACE Lexer        - FIELD        '12/02/2017 10:20:02'
   2017-06-15 06:17:32,433 [main] TRACE Lexer        - FIELD        '0'
   2017-06-15 06:17:32,433 [main] TRACE Lexer        - FIELD        ' -2.00'
   2017-06-15 06:17:32,433 [main] TRACE Lexer        - DELIMITER    <CR><LF>
   2017-06-15 06:17:32,433 [main] TRACE Lexer        - FIELD        ' 3'
   2017-06-15 06:17:32,433 [main] DEBUG Driver       - MESSAGE      Charge         [code=2,
description=Item 2    , dataHora=2017-02-12T10:20:02, VAT=false, amount=-2.0]
   2017-06-15 06:17:32,434 [main] INFO  Driver       - MESSAGE      New charge received
   2017-06-15 06:17:32,434 [main] TRACE Lexer        - FIELD        'Item 3    '
   2017-06-15 06:17:32,434 [main] TRACE Lexer        - FIELD        '13/03/2017 11:30:03'
   2017-06-15 06:17:32,434 [main] TRACE Lexer        - FIELD        '1'
   2017-06-15 06:17:32,434 [main] TRACE Lexer        - FIELD        '  3.25'
   2017-06-15 06:17:32,434 [main] TRACE Lexer        - DELIMITER    <CR><LF>
   2017-06-15 06:17:32,434 [main] ERROR Lexer        - ERROR        Wrong          character
(ignored): 'a'
   2017-06-15 06:17:32,435 [main] TRACE Lexer        - FIELD        ' 4'
   2017-06-15 06:17:32,435 [main] ERROR Lexer        - ERROR        Wrong           message
(ignored):
    Item a    13/03/2017 11:30:031  3.25

   2017-06-15 06:17:32,435 [main] DEBUG Driver       - MESSAGE      Charge         [code=3,
description=Item 3    , dataHora=2017-03-13T11:30:03, VAT=true, amount=3.25]
   2017-06-15 06:17:32,435 [main] INFO  Driver       - MESSAGE      New charge received
   2017-06-15 06:17:32,435 [main] TRACE Lexer        - FIELD        'Item 4    '
   2017-06-15 06:17:32,435 [main] TRACE Lexer        - FIELD        '15/06/2017 06:17:32'
   2017-06-15 06:17:32,435 [main] TRACE Lexer        - FIELD        '1'
   2017-06-15 06:17:32,435 [main] TRACE Lexer        - FIELD        '  0.85'
   2017-06-15 06:17:32,436 [main] TRACE Lexer        - DELIMITER    <CR><LF>
   2017-06-15 06:17:32,436 [main] INFO  Channel      - CONTROL      Closed channel
   2017-06-15 06:17:32,436 [main] TRACE Lexer        - DELIMITER
   2017-06-15 06:17:32,436 [main] DEBUG Driver       - MESSAGE      Charge         [code=4,
description=Item 4    , dataHora=2017-06-15T06:17:32, VAT=true, amount=0.85]
   2017-06-15 06:17:32,436 [main] INFO  Driver       - MESSAGE      New charge received
   2017-06-15 06:17:32,436 [main] TRACE Lexer        - DELIMITER
   2017-06-15 06:17:32,436 [main] INFO  Driver       - CONTROL      Stopped communication
```

Observi's l'error de color vermell en la consola que s'ha comentat prèviament.

Estructura del projecte

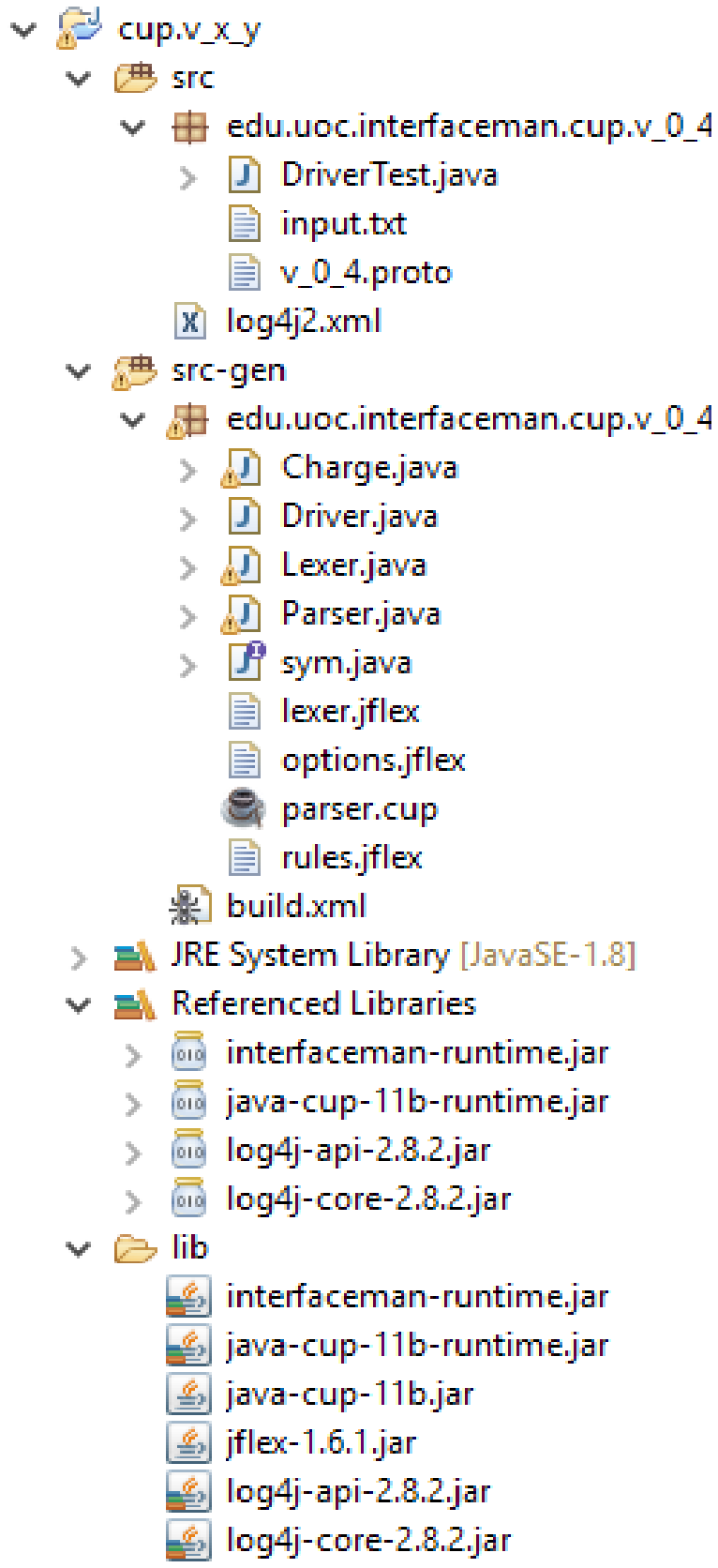


Diagrama UML de classes

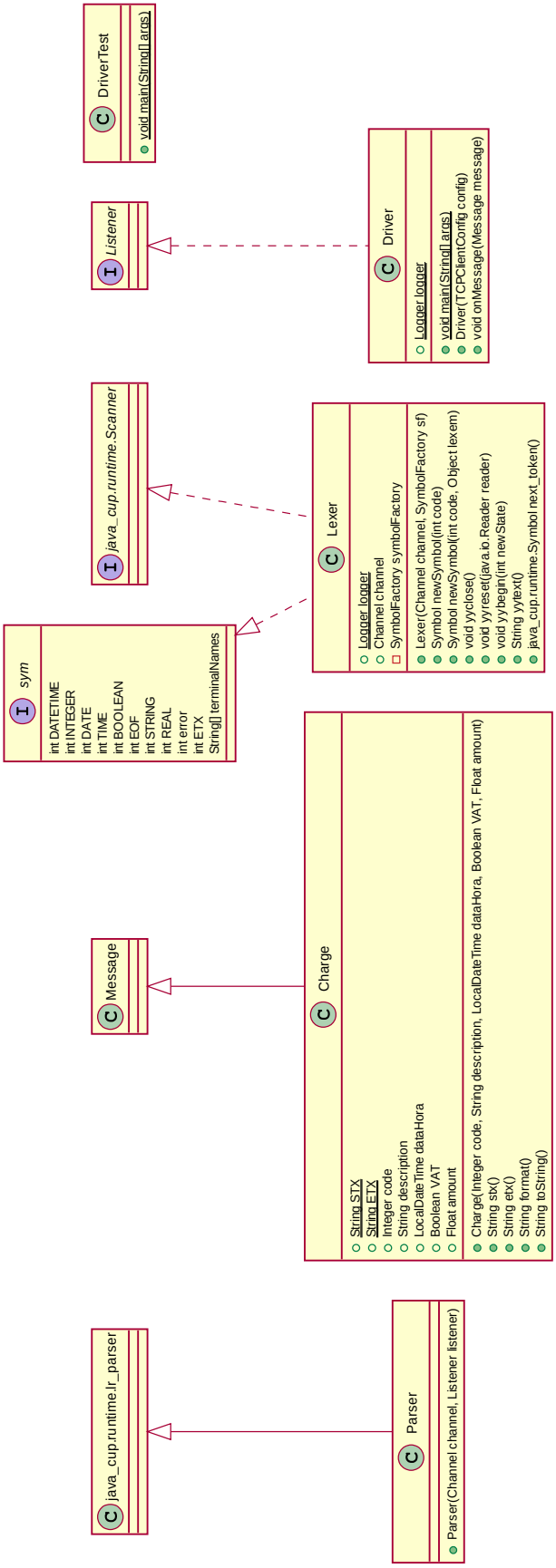
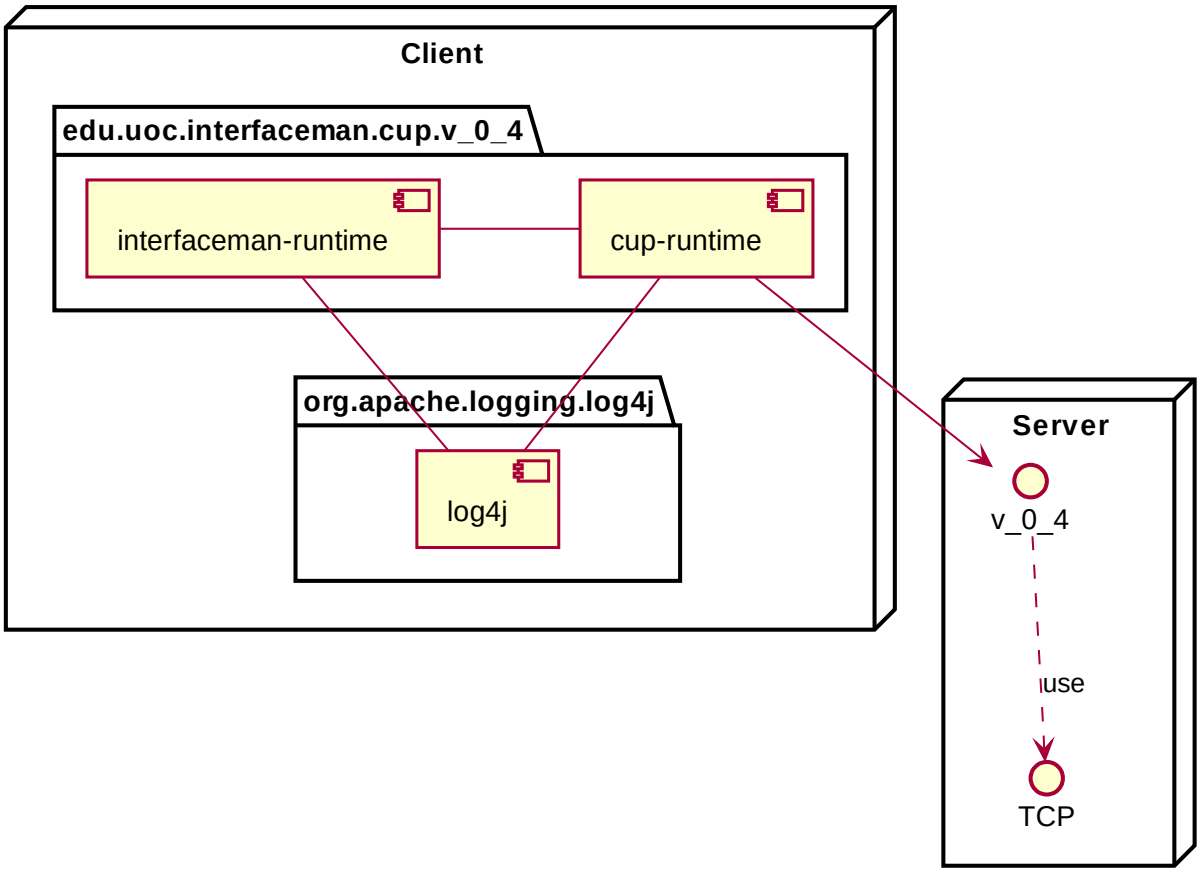


Diagrama UML de components de l'aplicació resultant



Recopilació d’informació prèvia

Recerca d'eines

Eines existents

Amb l'objectiu d'analitzar la possibilitat d'aprofitar desenvolupaments actuals o idees semblants, s'han cercat eines existents i propostes similars a la plantejada. Com era d'esperar, no hi han moltes aplicacions dins d'aquest camp, però sí més de les que s'esperava trobar.

De la idea bàsica d'usar un compilador genèric de fonts de programes per a utilitzar en protocols de comunicació, fins a l'idea final d'un editor visual i passant pel mig per la idea d'un metallenguatge, compilable també, per a formalitzar l'especificació d'un protocol, és d'aquesta última de la qual se n'han trobat més resultats. Pel camí també s'han descobert força documents i estudis que escriuen sobre el tema de recerca, els quals elaboren plantejaments anàlegs a la proposta.

Com a objectiu complementari, s'han buscat també altres eines genèriques de «compiladors de compiladors», com a alternatives a la parella formada per JFlex i CUP, implementacions de màquines d'estats, necessàries per a modelar el processament dels protocols, i per últim també implementacions sobre el port sèrie, ja que Java no en subministra una de nativa.

Compiladors de protocols amb DSL

Aleshores, entrant ja en matèria, sobre el programari existent per a compilar protocols de comunicació, la primera eina amb que ens trobem s'anomena significativament The Austin Protocol Compiler. Aquesta es caracteritza per a estar pensada per a protocols asíncrons sobre UDP, la seva sortida és en codi C, el seu metallenguatge s'anomena *Timed Abstract Protocol notation* i subministra un entorn d'execució o *runtime* pels sòcols Berkeley o *sockets*. La data de la primera edició del corresponent llibre és del 19 de novembre de 2004.

El següent treball s'anomena ALFred, *an ALF/ILP Protocol Compiler for Distributed Application Automated Design*. És un treball de recerca de l'INRIA –l'institut nacional francès d'investigació en informàtica i automàtica– i data del gener de 1996. El DSL que usa és l'Esterel, el qual es comentarà tot seguit. Es compon de dos compiladors: un de control anomenat ALF i un altre de manipulació de dades dit ILP, i el llenguatge dels fonts generats és en codi C.

Com dèiem fa un moment, l'Esterel és un llenguatge per a programar sistemes síncrons, el qual genera codi C o implementacions *hardware*. El seu desenvolupament va ser començat a principis de la dècada dels 80 per l'INRIA i l'Ecole des Mines de Paris. Es va intentar estandarditzar a l'IEEE vora els anys 2000 sense èxit. Existeix un entorn de desenvolupament comercial anomenat Esterel Studio.

Un altre IDL és l'Apache Thrift, una implementació publicada l'abril de 2007 per Facebook, però que ara és un projecte *open source* mantingut per l'*Apache Software Foundation*. És un *framework* que permet definir i crear serveis RPC multiplataforma en molts llenguatges de programació, entre ells el Java. Defineix una arquitectura en forma de *protocol stack* mitjançant l'ús d'una

biblioteca *runtime*. Encara que està pensat per protocols binaris, també disposa d'algun que altre protocol textual com JSON. Alguns dels altres avantatges que posseeix són treballar amb fitxers, sòcols, capçaleres de longitud, compressió i servidors mono i multi-client bloquejants i no bloquejants.

Prolac fa una aproximació al tema des de la perspectiva de la orientació a objectes. Aquest projecte és el resultat de la tesi de màster d'Eddie Kohler al MIT publicada el setembre de 1997. Encara que aquest enfoc és interessant, per desgràcia, el projecte ja no és mantingut i la implementació actual té problemes i no està finalitzada. Per altra part, només està pensat per la plataforma Linux.

BinPAC és un altre projecte força conegut, el qual sí es manté actualment. La primera versió va aparèixer l'abril de 2011. Genera codi C++ i es recolza en els coneguts Flex i Bison. És força complet, ja que té en compte els aspectes principals de la connexió, les dades i els seus tipus, els missatges, el flux o els estats.

Un altre projecte interessant és Ragel. Està enfocat cap a la màquina d'estats, la qual es pot representar amb el famós programari Graphviz. A la seva vegada, Ragel es recolza en el metallenguatge Colm. El qual té un funcionament similar a JFlex, on usa expressions regulars per reconèixer les seqüències de dades i permet inserir-hi codi *inline* per executar accions. Sap generar codi font pels llenguatges C, C++ i assemblador, però pot ser estès a d'altres llenguatges com Java o Ruby. Aquest cas també és el resultat de la tesi de doctorat del doctor Adrian D. Thurston publicada el desembre de 2008.

Hapy és un exemple de biblioteca que genera el *parser* al vol, en lloc de generar-lo en temps de compilació, és a dir, en temps d'execució. Està pensat especialment per a protocols de comunicació, ja que pot ser configurat amb un mode *on-line*. Aquest mode opera de forma iterativa per processar amb missatges limitats per separadors i prefixes on l'entrada completa no està disponible immediatament. Genera arbres amb objectes C++ i la primera versió va aparèixer el desembre de 2003.

La biblioteca libPDL també genera el compilador en temps d'execució com l'anterior. Utilitza un metallenguatge anomenat PDL (*Protocol Definition Language*) que permet codificar i descodificar missatges de protocols de comunicació. El llenguatge defineix un protocol mitjançant un diccionari de paquets, els quals es defineixen a la seva vegada amb una seqüència de variables, constants o altres paquets. Una de les particularitats que té és un mode que treballa a nivell de bits en lloc de bytes. És implementat amb Flex, Yacc i C++ i la primera versió de la qual es té constància és de 2004.

Protlr és un producte comercial que treballa amb protocols binaris. El codi font que genera està disponible en varis llenguatges: ANSI C, C++, Java, etc. A semblança amb l'eina anterior, aquesta també permet operar a nivell de bits. Disposa d'algunes particularitats interessants. La primera és que, al funcionar amb protocols binaris, també genera codi addicional per a ser usat amb el programari Wireshark i poder analitzar els missatges de xarxa. Una segona característica interessant és la generació de documentació HTML a partir de la definició DSL del protocol. I una última opció és que permet utilitzar-lo des del núvol.

I l'últim cas interessant trobat és el de GAPA, *a Generic Application-Level Protocol Analyzer and its Language*, un treball conjunt de diferents institucions

universitàries, l'IEEE i Microsoft. A diferència dels altres casos, aquest se centra en l'àmbit de la seguretat i l'anàlisi en temps real del tràfic de xarxa de qualsevol protocol de comunicació. Per aquesta raó, no hi entrarem en detall, però sí es recomana la lectura del text enllaçat.

Finalment afegir que aquesta llista d'aplicacions no és exhaustiva, però serveix com a mostra representativa d'alguns dels programes existents.

Documents sobre el tema de recerca

Durant la cerca d'aplicacions sobre la idea del projecte, s'han trobat força documents molt interessants que estudien i analitzen el mateix tema de recerca, tant dins de l'àmbit universitari com no.

Malgrat que seria convenient fer-ne una lectura, per agafar-ne idees i solucions als problemes comuns, la limitació de temps ho fa impossible. Tanmateix, almenys sí que en farem un breu esment en aquest parèntesi. Si més no, pot servir de cara a la continuïtat del projecte després del TFG.

Primer de tot, ja ens trobem amb altres persones fent-se la mateixa pregunta, Is it safe to use flex and bison as a protocol parser? o també Name me a Binary Parser.

En canvi, en aquest article, Lex and Yacc for Embedded Programmers, l'autor desenvolupa un exemple complet amb Lex i Yacc per a implementar un hipotètic protocol sèrie per a un sistema encastat que controla un motor.

I en aquesta presentació, Network Protocol Specification Languages & Compilers, es realitza una descripció més general del problema.

Ara, des d'un punt de vista més abstracte, referent als llenguatges de dominis específics, trobem varis treballs.

Abans, un altre cop més, algú es pregunta el mateix: Is there a domain-specific language for network communication protocols?

El treball Domain Specific Languages (DSLs) for Network Protocols planteja un enfoc molt semblant al proposat en el projecte. Analitza el desenvolupament de protocols des del disseny, passant per la implementació i fins els tests. Tracta el tema de la serialització de les dades i el comportament del protocol mitjançant màquines d'estat finites. També defineix un protocol d'exemple, tal com es fa en aquest projecte.

En la presentació de Defining a Network Protocol in a Domain Specific Language no només presenta també el problema general, sinó que, a més, també considera els aspectes del rendiment i la seguretat.

En el cas de CoNSoLe: A Domain Specific Language for Network Services es presenta un treball extens de recerca amb una aproximació similar al projecte. Descriu els mecanismes típics dels protocols que, més tard, ha de resoldre el metallenguatge. I també proposa d'exemple uns casos d'estudi que de forma progressiva aniran ampliant la semàntica del llenguatge. Després, va més enllà i, d'alguna manera similar, també proposa un entorn d'execució en forma de components.

I per últim, i relacionat indirectament amb aquest subapartat, indicar un parell de llenguatges estàndards usats en l'àrea dels protocols de comunicació, l'Specification and Description Language i l'Interface description language, i el llibre Communication Protocol Engineering, pensat per a sistemes encastats, el qual abasta tot el cicle de desenvolupament –anàlisi, disseny, implementació,

tests i verificació– usant els principals llenguatges de disseny: SDL, MSC, TTCN i UML. A més, ensenya com crear una biblioteca d'una màquina d'estats finita.

Compiladors de compiladors

Retornant a la qüestió de les eines i, en concret, als compiladors genèrics de compiladors, en aquest cas, trobem una llista molt més extensa que amb els compiladors de protocols.

En la Viquipèdia, es troba aquesta llarga relació, Comparison of parser generators, en diferents llenguatges, on compara algunes de les seves característiques. I en aquesta altra plana web, Open Source Parser Generators in Java, hi ha una recopilació exclusiva per Java.

Degut a l'abundància d'eines d'aquest tipus, no se n'analitzarà cap, excepte mencionar el generador més popular en Java, conegut amb el nom de JavaCC (Java Compiler Compiler), i l'existència d'un *plugin* Eclipse per CUP, CUP Eclipse Plugin.

Màquines d'estats finits en Java

Java no subministra cap implementació d'una màquina d'estats finits, o autòmat finit, de forma nativa. Per tant, cal cercar-ne alguna ja desenvolupada, si no és que la pròpia màquina d'estats que incorpori el compilador sigui suficient. Però si es requereix cobrir algun requisit especial, caldrà implementar-ne una de pròpia.

Dins del grup d'implementacions ja existents, StatefulJ figura com un bon candidat. Un altre producte interessant és Java Finite State Machine Framework, el qual permet crear la màquina per diferents vies: programàticament, amb un arxiu XML de configuració o gràficament amb un *plugin* per Eclipse.

Dintre de la possibilitat d'haver de crear-ne una de pròpia, de les diverses formes alternatives d'implementacions, dues de les que tenen més probabilitats de ser escollides són amb una taula de transicions d'estats i el patró de disseny estat.

Per una altra banda, existeix una adaptació de les màquines d'estats als processos de comunicació. És l'anomenada Communicating finite-state machine, en la qual es tenen en compte les operacions específiques de rebre i enviar sobre una col·lecció de canals de comunicació. Malgrat que disposar d'aquesta màquina seria una solució ideal, no se n'ha trobat cap implementació en Java.

I per acabar aquest subapartat sobre màquines d'estats, m'agradaria recomanar la lectura de l'article Why Developers Never Use State Machines, el qual és molt il·lustratiu i on coincideixo amb l'opinió expressada.

Comunicació sèrie en Java

Java no proporciona una implementació nativa per a totes les plataformes per a la comunicació per port sèrie, només per a sistemes *nix, tal com es desprèn d'aquesta pregunta How to send data to COM PORT using JAVA?

Tanmateix, subministra una API estàndard, Java Communications API (javadoc paquet javax.comm), per a que a terceres parts ofereixin la seva

implementació particular. Així doncs, cal buscar també una altra eina ja existent: How to get javax.comm API?

Una de les recomanades és RXTX (wiki) i està disponible com a *plugin* per a Eclipse, però sembla que fa força temps que no es desenvolupa. Una alternativa és utilitzar java-simple-serial-connector, la qual sembla estar més actualitzada, encara que tampoc massa més.

Elecció de les eines

Dins de la planificació proposada, aquesta subtasca 1.1.2 s'ha postergat fins després de totes les subtasques de la tasca 1.1 i abans de la tasca 1.2, perquè faltava informació que feia difícil prendre una decisió i el resultat de les tres subtasques següents faria més fàcil i millor la decisió de l'elecció de les eines.

Aleshores, amb la primera fita de la compilació de protocols, els requisits a complir per l'eina són els següents:

- Requisit no funcional: disponible per Java.
- Cal poder implementar el protocol dissenyat.

  Per exemple, les eines que només treballen en binari no es poden usar.
- També, per a poder implementar el protocol dissenyat, cal complir amb les particularitats dels protocols de comunicació respecte als llenguatges de programació que s'han descrit.
- Gratuït i, a poder ser, *open source* i llicència lliure.

La primera opció plantejada va ser la parella formada per JFlex i CUP. Els avantatges són que són coneguts i adaptables a les particularitats dels protocols. Una segona opció proposada és Xtext, però com a desavantatge, segurament no es pugui adaptar a les particularitats dels protocols. I com a terceres opcions hi han altres eines trobades durant la recerca d'informació. De les quals, les que compleixen les condicions dels requisits serien JavaCC, Apache Thrift i Ragel.

El primer seria una alternativa a JFlex/CUP, mentre que els altres dos incorporarien també el metallenguatge. Això últim, acceleraria el desenvolupament de tot el projecte, però no seria un treball sobre compiladors del tot, sinó més aviat sobre una aplicació genèrica sobre protocols, el que se sortiria del objectiu inicial del projecte. I, a més, perquè no donar una implementació pròpia i una visió de com hauria de ser una eina d'aquest tipus?

Per tant, la decisió queda entre JFlex/CUP i JavaCC. Però aquesta s'ha determinat deixar-ho per a després, durant les primeres proves, ja que JavaCC pot ser una bona opció a la primera. A més, també dependrà de la qüestió de la màquina d'estats finits.

Sobre aquest assumpte, s'ha de tenir en compte que en els protocols també tenim estats diferents, al igual que en els compiladors, sobre els qual progressen les transicions entre missatges, transaccions i sessions.

Com en el tema anterior, ara no és evident si la pròpia màquina d'estats que ofereix JFlex/CUP serà suficient, si JavaCC disposa d'una màquina d'estats, si en caldrà una implementació externa o inclús una de pròpia per a cobrir

funcionalitats especials. Per tant, també es demora aquesta decisió a fases més avançades del projecte.

Llavors, amb la segona fita del compilador del metallenguatge, podem usar JFlex/CUP com es va plantejar inicialment amb l'avantatge de que ja es coneix el seu funcionament. La segona opció és Xtext, que té el benefici de disposar ja de funcionalitats per a l'editor. A més, pot ser útil per a altres futurs projectes, com aprendre Xtend, encara que ara això no sigui directament vinculant per a aquest projecte. Un possible inconvenient trobat seria no poder usar el compilador fora de l'entorn d'eclipse, però s'ha pogut confirmar que sí és possible en les funcionalitats.

Com a terceres opcions, tenim els compiladors de protocols amb DSL ja incorporat, com Apache Thrift i Ragel. Aquests tenen el avantatge d'usar una solució estàndard i la comoditat de no haver de crear aquesta part, però poden tenir l'inconvenient de que el llenguatge no compleixi amb el protocol de proves dissenyat. Però, com en el cas de la primera fita, es prefereix evitar aquestes opcions i deixar aquesta part com a exercici de l'àrea de compiladors del TFG.

Però, a diferència d'abans, aquí sí que s'ha decidit triar ja l'eina i aprofitar els beneficis d'Xtext, sobretot també de cara a la tercera fita. És un punt intermedi amb tornar a fer-ho tot des del principi amb un compilador genèric i un compilador amb DSL ja incorporat.

En quant a la tercera fita del dissenyador del metallenguatge, podem optar per crear una interfície gràfica pròpia. L'avantatge és que no hi ha limitacions en quant al que es pot fer, però té l'inconvenient que s'ha de partir des de zero. Com a segona opció, tenim usar Xtext un altre cop. Els beneficis són que ja disposa d'un *rich editor* textual ja disponible, altres funcionalitats disponibles i, com abans es comentava, pot ser útil per a d'altres projectes. La pega podria ser estar limitats en quant a les funcionalitats que ofereix i es necessiten. Per exemple, editors de llistes (de camps, missatges, etc.) i gràfics com diagrames d'estats. Tanmateix, s'ha pogut verificar que aquestes funcionalitats són possibles en Xtext. Per tot això, en la tercera fita, també s'ha optat per decidir-se un altre cop pel Xtext.

Per una altra banda, afegir que s'ha trobat una eina per a crear diagrames UML, mitjançant l'ús d'un DSL i el *software* Graphviz, anomenada PlantUML. Apart d'utilitzar-lo per al seu ús obvi, també és interessant perquè converteix un metallenguatge (per a crear diagrames UML) en un altre (DOT: graph description language), el mateix que pretén arribar a fer aquest projecte. També suporta SDL (Specification and Description Language) i disposa d'un plugin per a Eclipse.

Protocols de comunicació

Caracterització principal dels protocols

A continuació, es definirà la llista de les principals característiques dels protocols de comunicació, principalment, des del punt de vista de la capa d'aplicació. Estan en ordre ascendent d'abstracció de conceptes i agrupades de forma esquemàtica i jeràrquica.

1. Medi físic:
    1. Canal de transmissió:
        1. Cable sèrie
        2. Xarxa
        3. Fitxers

           Hi han altres tipus com, USB, bluetooth, etc, que quedaran fora de l'anàlisi, ja que requereixen programació de drivers i no es programen aquest tipus de protocols.
    2. Paràmetres de configuració del canal de transmissió:
        1. Cable sèrie:
            1. Número de port: COM1, /dev/ttyS0, etc.
            2. Velocitat en bauds: 75, 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 i 115200 bit/s
            3. Control de paritat: none, even, odd, mark, space
            4. Data bits: 5-9 bits
            5. Stop bits: 1, 1.5, 2 bits
            6. Timeout recepció: 0-n mil·lisegons
            7. Control de flux per:
                1. Hardware: CTS/RTS, DTR/DSR
                2. Software: xOn/xOff
        2. Xarxa:
            1. Adreça IP del servidor
            2. Port TCP/UDP
            3. Timeout recepció: 0-n mil·lisegons
        3. Fitxers:
            1. Dos paths: entrada/sortida
            2. Noms estàtics: bloqueig d'accés exclusiu per evitar accés concurrent, escriptura amb append, lectura amb esborrat posterior
            3. Noms dinàmics: patró amb data/hora per evitar bloqueig mitjançant un delay de processament (marge per diferències de temps entre màquines)
            4. Reanomenat: escriptura amb nom temporal per evitar bloqueig
            5. Timeout recepció: 0-n mil·lisegons

3. Orientat a la connexió:
    1. No: sèrie, fitxers, xarxa (UDP)
    2. Sí: xarxa (TCP)
        1. Connexió permanent/temporal
        2. Reconnexió: en cas de caiguda o aturada per manteniment de servidor/client
2. Sentit de la comunicació:
    1. Simplex: unidireccional
    2. Duplex: bidireccional
        1. Half duplex
            1. Sincronització: tipus síncron, confirmació de recepció (ACK), ENQ(uiry) opcional per bloqueig del canal
        2. Full duplex:
            1. Sincronització: tipus asíncron, confirmació de recepció (ACK) opcional, respostes amb mateix ordre de recepció o desordenades amb identificador seqüencial de petició (per processament multithreading)
3. Format de la informació:
    1. Textual: ASCII
    2. Binari
    3. Híbrid: per framing i control de flux
4. Senyalització:
    1. Out-of-band: per exemple, protocol FTP
    2. In-band
5. Control de flux per:
    1. Hardware: per exemple, cable sèrie (CTS/RTS, DTR/DSR)
    2. Software: per exemple, xOn/xOff, ACK/NAK, ENQ
6. Framing dels missatges lògics:
    1. Header length: message i/o fields
    2. Separadors externs (start/end missatge/transacció) i interns (de camps): STX/ETX (per sincronització), TAB/CR/LF/FF, SOH/EOT, FS/GS/RS/US
7. Detecció d'errors:
    1. Checksums (per cable sèrie):
        1. LRC: per exemple, amb XOR
        2. CRC: CRC-16, CRC-32, etc.
    2. Error de transmissió (durant enviament/recepció): NAK/timeout, reintents retransmissió
    3. ACK: no error detectat
8. Sublayering (a nivell, dins, de la capa d'aplicació):
    1. Capa de baix nivell: medi físic (canal, paràmetres configuració), control de flux, framing, detecció d'errors, a/sincronia, connexió, etc.
    2. Capa d'alt nivell (payload):

1. Tipus de missatges lògics: command, request/reply
2. De/serialització d'estructures de dades: binari, textual, estàndards ASN.1, SOAP, ...
3. Diccionaris de (amb rol(s) d'origen):
    1. Camps: agrupacions de bytes, serialització de dades (de longitud fixa (amb aligning/padding) o variable).
    2. Missatges: agrupacions de camps (seqüència ordenada o no) (de longitud fixa o variable).

       Possibilitat d'alguns camps omesos o valors opcionals.
    3. Transaccions (si no present: és un únic command): agrupacions de missatges (requests/replies) o seqüències de commands (seqüència ordenada).

       Possibilitat d'alguns missatges opcionals (omesos) i subtransaccions (compostes o niades).
    4. Sessions (si present): agrupacions de transaccions, transaccions d'inicialització/finalització que encapsulen la comunicació.

       Per exemple, login-logout: si el login falla, avorta tota comunicació, però si el login és correcte, llavors sempre ha de fer logout al final.

9. Arquitectura de nodes:
    1. Sèrie: punt a punt (dos nodes)
    2. Xarxa:
        1. Centralitzada (client/servidor): servidor mono/multiclient
        2. Distribuïda:
            1. Punt a punt (dos nodes o més)
            2. Grup: broadcast/multicast
    3. Fitxers: generalment, punt a punt (dos nodes)
    4. Rols: un master i un (o més) slave, master i slave alhora

Disseny del protocol pels jocs de proves

En el disseny del protocol s'intentarà cobrir, és a dir, automatitzar, el màxim de característiques principals, sobretot les més usades pels protocols, però sense arribar a ser un requisit imprescindible.

Les característiques estan classificades i agrupades en els tres nivells de requisits MoSCoW. Els que no estan especificats són de nivell 4 i, per tant, no es realitzaran ara.

Al mateix temps, se segueix un disseny incremental per versions. On es van afegint característiques, extretes de la caracterització de l'apartat anterior, en ordre ascendent de complexitat. Això permet evolucionar el protocol, augmentant-ne la potència, expressivitat i flexibilitat, a canvi, és clar, d'augmentar-ne el cost de desenvolupament per la seva major dificultat.

Cal advertir que és probable que es produeixin algunes incoherències en el disseny, degut a la dificultat de crear un protocol que vagi incorporant totes les característiques seleccionades.

Nivell 1.
Versions 0.4 – 1.0

Versió 0.4

- Canals: xarxa (TCP)
- Connexió: sí (permanent)
- Sentit de la comunicació: simplex
- Format: híbrid
- Senyalització: in-band
- Framing missatges lògics: separadors externs (end missatge) CR/LF/RS
- Sublayering:
  - Capa de baix nivell: medi físic (canal), connexió
  - Capa d'alt nivell (payload): tipus de missatges lògics (command), de/serialització d'estructures de dades textual, diccionaris:
    - Camps: de longitud fixa (aligning/padding)
    - Missatges: seqüència de camps ordenada (de longitud fixa)
  - Transaccions: únic command
- Arquitectura nodes: punt a punt, centralitzada (client/servidor), servidor monoclient, rols (un master i un slave)

En aquesta primera versió, es necessita implementar un nombre mínim de característiques indispensables per a ser mínimament funcional.

Per altra banda, el lògic seria implementar primer la característica de no connexió amb el canal del port sèrie, però com que Java no subministra una implementació nativa, s'ajorna per a una versió posterior.

Els tipus dels camps seran: string, booleà, enter, real, data i/o hora.

Els camps (i unic missatge i transacció) seran: codi article, descripció article, data/hora, iva inclòs, import.

En resum, aquest primer protocol és la típica sortida d'impressora.

Versió 0.5

- Framing de missatges lògics: separadors externs (end transacció) FF/GS
- Sublayering:
  - Capa de baix nivell: framing
  - Capa d'alt nivell (payload): diccionaris:
    - Transaccions: commands

S’afegeixen els següents missatges:

1. Capçalera: codi article, descripció article, data/hora, iva inclòs, import
2. Total: «TOTAL.....», import

I la següent transacció:

1. capçalera
2. línies de detall
3. total

És a dir, ara el protocol incorpora la impressió de pàgina.

Versió 0.6

- Framing missatges de lògics: separadors externs (start/end missatge/transacció) STX/ETX (per sincronització), SOH/EOT
- Sublayering:
  - Capa d’alt nivell (payload): diccionaris:
    - Missatges: seqüència de camps ordenada (de longitud variable)

Ara s’introdueixen els separadors d’inici, el que ha de permetre sincronitzar i descartar qualsevol missatge/transacció inacabada i optimitzar la quantitat d’informació transmesa permetent missatges de longitud variable.

I es canvien els separadors per semblar-se més a un típic protocol d’integració que a una sortida d’impressora típica.

També s’elimina el missatge de capçalera, s’optimitza el de total i s’afegeix el concepte d’identificador d’ordre (command) com a camp prefix dels missatges:

1. «D»: línia de detall
2. «T»: línia de total

Versió 0.7

- Sentit de la comunicació: duplex (half duplex)
- Control de flux per software: ACK, NAK
- Sublayering:
  - Capa de baix nivell: control de flux, sincronia

Aquí s’afegeix la característica de la comunicació síncrona i bidireccional half duplex amb el següent control de flux:

- ACK=confirmació de recepció
- NAK=error de transmissió/recepció/no preparat/etc.

  En aquesta versió, només és a efectes informatius (per exemple, per comptadors estadístics de transmissions errònies i correctes).

Versió 0.8

- Detecció d'errors: checksums (LRC), ACK, NAK, timeout
- Sublayering:
  - Capa de baix nivell: detecció d'errors

En aquesta versió, s'introdueix la detecció d'errors automàtica, en la capa de baix nivell, amb un checksum simple (Longitudinal Redundancy Check), XOR dels bytes del payload inclòs l'ETX:

- Missatge: STX payload ETX checksum
- ACK=missatge correcte
- NAK=missatge corrupte
- timeout=no hi ha cap resposta positiva ni negativa en x mil·lisegons, possible tall de les comunicacions. A efectes pràctics, equival a un caràcter NAK.

Un altre cop, com en la primera versió, com que TCP és un canal de transport confiable, seria innecessària aquesta característica. Però, en moltes ocasions, un protocol permet la transmissió per més d'una canal, sèrie i TCP, i el protocol continua essent el mateix en ambdós canals, per no haver de desenvolupar diferents variants segons el canal.

Versió 0.9

- Sublayering:
  - Capa d'alt nivell (payload): tipus de missatges lògics (request/reply), diccionaris:
    - Transaccions: missatges (requests/replies)

En aquesta versió, s'incorporen els missatges de petició i la seva resposta a nivell lògic, mitjançant la seva definició com a transacció:

Missatge resultat al detall i total: «R», resultat(booleà), text explicatiu

Ex: «D» 12 ... → «R» false «Codi d'article desconegut: 12»

«T» 32.50 → «R» true «Import total quadrat»

Versió 1.0

- Arquitectura nodes: rols (master i slave alhora)

Finalment, per acabar, qualsevol dels dos costats de la comunicació pot iniciar una transacció:

- Nou missatge des de l'altre costat: «A», codi article, descripció article, iva inclòs, import
- Nova transacció: «A» → «R»

Nivell 2.
Versions 1.1 – 2.0

Versió 1.1

- Sublayering:
  - Capa d'alt nivell (payload): diccionaris:
    - Missatges: possibilitat d'alguns valors opcionals

«R», resultat(booleà), [text explicatiu]
«R» true « »

Versió 1.2

- Framing missatges lògics: header length message

Com a exemple, es transforma el missatge de resultat a tipus amb capçalera de longitud en lloc del tipus amb separadors externs (STX/ETX).

Versió 1.3

- Detecció d'errors: reintents

Comptador de intents de retransmissió en cas d'errors (NAK/timeout).

Versió 1.7

- Canals: fitxers (noms estàtics i/o dinàmics)
- Connexió: no

Es farà el cas que sigui més fàcil d'implementar.

Versió 2.0

- Configuració canals i generals protocol (timeout, reintents)
- Sublayering:
  - Capa de baix nivell: medi físic (paràmetres)

Via fitxer .properties: ex. serial.bauds = 9600

Nivell 3.
Versions 2.1 – 3.5

Versió 2.1

- Detecció d'errors: checksums (CRC-16)

Checksum LRC canviat per CRC-16.

Versió 2.2

- Control de flux per software: ENQ

Caràcter de control ENQ(uiry) a mode de keep-alive de baix nivell per a verificar si qualsevol dels dos costats és responsiu:

ENQ → ACK/NAK/timeout/error de comunicació

Versió 2.3

- Connexió: sí (temporal)

Afegida possibilitat d'establiment de connexió al iniciar una transacció i finalitzar-la al completar-la.

Versió 2.4

- Connexió: sí (reconnexió permanent)

Tolerància a fallades automàtica (és opcional i llavors s'atura), on –si es perd la connexió per tall, caiguda o aturada de qualsevol dels dos costats– entra en un estat especial de reconnexió continua, on ho intenta cada x segons (configurable).

Versió 2.5

- Framing missatges lògics: separadors interns (de camps) TAB/US/ |
- Sublayering:
  - Capa d'alt nivell (payload): diccionaris:
    - Camps: de longitud variable

Concepte natural dels compiladors on els tokens no constants són de longitud variable, separats mitjançant tokens separadors sí constants.

Versió 2.6

- Sublayering:
  - Capa d'alt nivell (payload): diccionaris:
    - Missatges: possibilitat d'alguns camps omesos

Gràcies a la modificació anterior dels camps de longitud variable i ús de separadors, ara és possible optimitzar la transmissió de dades ometent els valors dels camps (concepte null):

«R» | resultat(booleà) | [text explicatiu]

«R» | true |

Versió 2.7

- Sublayering:
  - Capa d'alt nivell (payload): diccionaris:
    - Missatges: seqüència de camps no ordenada

Nou subtoken identificador de camp i nou subseparador de valor (opcional), que permeten alterar l'ordre dels camps:

|CM=D|IM=32.50|ID=12

En aquest punt, tenim ja quelcom semblant a les funcionalitats que dóna un llenguatge de marques amb els avantatges que això implica. Per exemple, ser autodescriptiu amb metadades.

Això permet la possibilitat de ser forward compatible, ja que rebre nous camps futurs desconeguts, en la majoria de casos, no impedeix de continuar tractant els camps actuals coneguts.

Versió 2.8

- Sublayering:
  - Capa d'alt nivell (payload): diccionaris:
    - Transaccions: subtransaccions, compostes o niades i possibilitat d'alguns missatges opcionals (omesos).

Concepte afegit d'agrupacions de subtransaccions.

També és necessari incorporar el concepte (o restricció) de cardinalitat. Per exemple:

- 1 (obligatori)
- 0..1 o ? (opcional)
- 1..* o + (llista obligatòria de longitud variable)
- 0..* o * (llista opcional)
- m (nombre fix d'instàncies)
- m..n (rang variable d'instàncies, on m<=n)

Transacció nova en el protocol:

tiquet: «D-R» 1..*, «T-R» 0..1

Versió 3.0

- Sublayering:
  - Capa d'alt nivell (payload): diccionaris:

        - Sessions: transaccions d'inicialització/finalització

Missatges nous:

- «N», contrasenya
- «C», contrasenya
- «F»

Transaccions noves:

- Inici sessió normal: «N-R»
- Inici sessió amb creació addicional d'articles: «C-R»
- Fi sessió (qualsevol de les dues): «F-R»

Sessions (expressada, entre parèntesis, la cardinalitat):

- Normal: normal(1), tiquet(*), fi(1)
- Creació: creació(1), tiquet(*)/article(*), fi(1)

Versió 3.2

- Canals: cable sèrie

Sense control de flux per hardware.

Versió 3.3

- Sublayering:
- Capa d'alt nivell (payload): diccionaris amb rol(s) d'origen

Restriccions sobre qui origina (o inicia) què. Permet indicar un o més orígens, a nivell de tots els diccionaris de camps, missatges, transaccions i sessions.

Versió 3.5

- Sentit de la comunicació: duplex (full duplex (seqüència))
- Sublayering:
    - Capa de baix nivell: asincronia

Per permetre un protocol asíncron, cal identificar els missatges de les peticions amb un nou identificador de seqüència, per a poder associar després les seves corresponents respostes.

A més, cal implementar un mecanisme de garbage collection per a caducar missatges caducats mitjançant un timeout TTL (configurable).

Els protocols de comunicació vers els llenguatges de programació

Es troben unes poques diferències, tanmateix importants, i petits matisos de la compilació de llenguatges de programació respecte els protocols de comunicació que s'hauran de considerar en el desenvolupament.

La primera diferència que trobem és que, en molts protocols textuals, cada token té una longitud fixa, a diferència dels llenguatges de programació que són variables i utilitzen separadors com, per exemple, l'espai, el tabulador, el punt i coma o el salt de línia. Per tant, és necessari trobar una forma d'expressar els patrons de les expressions regulars amb uns quantificadors especificats amb un nombre concret, en lloc de nombres variables com ? (0 o 1), * (0 o més) i + (1 o més).

Una altra diferència es descobreix en el concepte EOF o fi de fitxer. Un compilador d'un font escrit en un llenguatge de programació deixa de processar l'entrada de caràcters en quant arriba al seu final. Doncs bé, en els protocols de comunicació es produeixen dos esdeveniments similars.

El més semblant s'esdevé amb un error de comunicació. Per exemple, quan des de l'altre costat tanquen la connexió de xarxa establerta, per tasques de manteniment del servei o per una caiguda accidental de l'altre procés. En aquest circumstància, hem de deixar de processar l'entrada, perquè ja no podem comunicar-hi més al quedar el canal de comunicació tancat.

Però, tanmateix, si volem crear un sistema tolerant a fallades, cal considerar aquest tall de comunicació de forma temporal i definir un procediment de restabliment de la connexió, en quant l'altre procés torni a estar disponible. És per això, que el nostre procés no s'aturarà del tot, però sí entrarà en un estat excepcional mentre duri aquest període. No cal dir, que en el moment del tall, qualsevol missatge, transacció o sessió a mig processar serà avortada. I, un cop restablerta la comunicació, el protocol tornarà al seu estat inicial i començarà des de zero com si fos la primera vegada.

El segon cas es dóna quan no hi han més dades a processar provinents des de l'entrada i no és deu a un error de comunicació. Aquest fet es produeix després d'esperar un temps màxim especificat en el protocol. En aquesta circumstància, però, a diferència del cas anterior, el canal encara roman obert. El qual significa que, en un moment posterior, poden arribar noves dades per processar.

En el cas de protocols master-slave, on actuem nosaltres com esclau, aquesta situació no és rellevant, ja que podem bloquejar-nos indefinidament en l'espera de noves dades.

Però, en el cas contrari i, sobretot amb protocols que especifiquen un temps màxim per a rebre una resposta a un missatge enviat, és imprescindible detectar aquesta casuística. Per exemple, per a repetir l'enviament del missatge anterior, assumint un error de transmissió en el primer enviament. És per això, que aquest EOF temporal es tractarà com un token més, especial i predefinit, per a poder actuar de la forma adequada segons dicti l'especificació del protocol.

L'última dissimilitud, però no menys important, es troba en que un compilador només processa una entrada de dades. Però, en els protocols bidireccionals, ens trobem amb dues entrades de dades. El canal d'entrada provinent del

sistema extern, el qual s'ha de parsejar i validar, i el propi canal de sortida cap al sistema extern.

Això permet no només validar l'entrada del sistema extern, menys susceptible a errors per ser el subministrador del protocol, sinó la nostra sortida, més susceptible a errors per ser una nova implementació i estar basada en la interpretació de la documentació i no en la creació del protocol.

No obstant això, és qüestionable si és necessari parsejar i validar la nostra sortida, ja que aquesta ha pogut ser provada i validada amb el nostre codi font. Per altra part, cap programari es pot assegurar lliure d'errors al 100%, pel que això el faria més robust. Però, pel contrari, el faria més ineficient, si ho considerem una acció innecessària.

També dependrà de si el sistema desenvolupat, disposa d'un component de serialització automàtic, feina oposada a la compilació, o es programa de forma ad hoc.

En aquest punt del projecte és difícil decidir sobre aquest conflicte, pel que, de moment, es decideix prendre la decisió més tard sobre la marxa durant el desenvolupament. Si és clar que, com a mínim, caldrà especificar el rol d'origen (canal d'entrada o sortida) en els diccionaris de camps, missatges, transaccions i sessions. Aquesta informació permetrà saber de quin costat de la comunicació volem generar el codi font, si el client o el servidor. És més, també si més tard volem crear automàticament eines de test per a poder simular el costat contrari i poder provar la nostra implementació del protocol.

Bibliografia

Bhatti, S.; Brady, E.; Hammond, K.; McKinna, J. (2009, 13 de gener). «Domain Specific Languages (DSLs) for Network Protocols». *University of St Andrews; Radboud University* [position paper]. [Data de consulta: 14 de març de 2017].

<https://eb.host.cs.st-andrews.ac.uk/drafts/ngna2009-dsl.pdf>

Borisov, N.; Brumely, D. J.; Wang, H. J.; Dunagan, J.; Joshi, P.; Guo, C. (2006, 7 de desembre). «A Generic Application-Level Protocol Analyzer and its Language». *UIUC; Carnegie Mellon University; Microsoft Research; UC Berkeley; ICE Nanjing* [informe de recerca]. [Data de consulta: 14 de març de 2017].

<https://www.isoc.org/isoc/conferences/ndss/07/papers/application_level_protocol_analyzer.pdf>

Braun, T.; Chrisment, I.; Diot, C.; Gagnon, F.; Gautier, L.; Hoschka, P. (1996, gener). «ALFred, an ALF/ILP Protocol Compiler for Distributed Application Automated Design». *Institut National de Recherche en Informatique et en Automatique* [informe de recerca]. [Data de consulta: 13 de març de 2017].

<https://pdfs.semanticscholar.org/3bdf/7291adaf83313747dbbd499989e881de2132.pdf>

ISSN 0249-6399

Candasamy, A. O. (2015, 17 de juliol). «Defining a Network Protocol in a Domain Specific Language». *Maynooth University* [presentació de tesi]. [Data de consulta: 14 de març de 2017].

<https://www.cs.nuim.ie/courses/desem/sites/default/files/Defining%20a%20Network%20Protocol%20in%20a%20Domain%20Specific%20Language.pdf>

Casey, J. (2011, 18 de febrer). «Network Protocol Specification Languages & Compilers». *Texas A&M University* [presentació de tesi]. [Data de consulta: 13 de març de 2017].

<http://groups.geni.net/geni/raw-attachment/wiki/LEARN/learn_tamu_jc_as.pdf>

Keane, J. (2006, 4 de febrer). «RXTX: The Prescription for Transmission». *Massachusetts Institute of Technology* [manual]. [Data de consulta: 23 de març de 2017].

<http://users.frii.com/jarvi/rxtx/intro.html>

Kohler, E. (2013, 2 d'abril). «An Object-Oriented Protocol Language». *Massachusetts Institute of Technology* [tesi de màster]. [Data de consulta: 13 de març de 2017].

<https://pdos.csail.mit.edu/archive/prolac/>

König, H. (2003). «Protocol Engineering» (1a. ed.). Cottbus: Springer.

McGuire, T. M. (205, 1 de juny). «The Austin Protocol Compiler» [llibre]. [Data de consulta: 13 de març de 2017].

<http://apcompiler.sourceforge.net/>

nmEdit software team (2006, 12 d'agost). «libPDL - Protocol Definition Language». *nmEdit software* [manual]. [Data de consulta: 13 de març de 2017].
<http://nmedit.sourceforge.net/subprojects/libpdl.html>

Oever, D. (2008, 14 de maig). «CoNSoLe: A Domain Specific Language for Network Services». *University of Twente* [tesi de màster]. [Data de consulta: 14 de març de 2017].
<http://essay.utwente.nl/58039/1/scriptie_D_van_%27t_Oever.pdf>

Popovic, M. (2006). «Communication Protocol Engineering» (1a. ed.). Boca Raton: CRC Press.

Power, L. (2003, 20 de febrer). «Lex and Yacc for Embedded Programmers». *embedded.com* [article]. [Data de consulta: 13 de març de 2017].
<http://www.embedded.com/design/prototyping-and-development/4024523/Lex-and-Yacc-for-Embedded-Programmers>

Skorkin, A. (2011, 1 de setembre). «Why Developers Never Use State Machines». *Skorks* [article]. [Data de consulta: 12 de març de 2017].
<http://www.skorks.com/2011/09/why-developers-never-use-state-machines/>

Sokolov, A. «java-simple-serial-connector». *Google Code* [manual]. [Data de consulta: 23 de març de 2017].
<https://code.google.com/archive/p/java-simple-serial-connector/>

The Bro Project (2017, 11 de febrer). «BinPAC — Bro 2.5 documentation». *The Bro Network Security Monitor* [documentació]. [Data de consulta: 13 de març de 2017].
<https://www.bro.org/sphinx/components/binpac/README.html>

Thurston, A. D. (2017, 12 de febrer). «Ragel State Machine Compiler». *Colm Networks* [article]. [Data de consulta: 13 de març de 2017].
<http://www.colm.net/open-source/ragel/>

Varis autors (2014, 15 de gener). «Communicating finite-state machine». *Viquipèdia* [article]. [Data de consulta: 12 de març de 2017].
<https://en.wikipedia.org/wiki/Communicating_finite-state_machine>

Varis autors (2016, 24 d'octubre). «State transition table». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/State_transition_table>

Varis autors (2017, 5 de març). «State pattern». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/State_pattern>

Varis autors (2016, 10 de novembre). «Esterel». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/Esterel>

Varis autors (2017, 10 de març). «Apache Thrift». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/Apache_Thrift>

Varis autors (2017, 2 de febrer). «Ragel». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/Ragel>

Varis autors (2017, 11 de març). «Comparison of parser generators». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/Comparison_of_parser_generators>

Varis autors (2017, 2 de març). «Compiler-compiler». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/Compiler-compiler>

Varis autors (2017, 27 de febrer). «JavaCC». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/JavaCC>

Varis autors (2016, 14 de desembre). «Protocol composition logic». *Viquipèdia* [article]. [Data de consulta: 13 de març de 2017].
<https://en.wikipedia.org/wiki/Protocol_composition_logic>

Varis autors (2016, 13 de novembre). «Specification and Description Language». *Viquipèdia* [article]. [Data de consulta: 15 de març de 2017].
<https://en.wikipedia.org/wiki/Specification_and_Description_Language>

Varis autors (2017, 4 de febrer). «Interface description language». *Viquipèdia* [article]. [Data de consulta: 15 de març de 2017].
<https://en.wikipedia.org/wiki/Interface_description_language>

Varis autors (2016, 12 de juliol). «Simplex communication». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Simplex_communication>

Varis autors (2017, 4 de març). «Duplex (telecommunications)». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Duplex_(telecommunications)>

Varis autors (2017, 4 de març). «In-band signaling». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/In-band_signaling>

Varis autors (2016, 6 d'octubre). «In-band and out-of-band signaling». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Signaling_(telecommunications)#In-band_and_out-of-band_signaling>

Varis autors (2016, 7 de setembre). «Software flow control». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Software_flow_control>

Varis autors (2017, 8 de març). «RTS, CTS, and RTR». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/RS-232#RTS.2C_CTS.2C_and_RTR>

Varis autors (2014, 10 de desembre). «Frame synchronization». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Frame_synchronization>

Varis autors (2015, 23 de novembre). «Longitudinal redundancy check». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Longitudinal_redundancy_check>

Varis autors (2017, 8 de març). «Cyclic redundancy check». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Cyclic_redundancy_check>

Varis autors (2017, 6 de març). «Serial port». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Serial_port>

Varis autors (2016, 8 de novembre). «Control character». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Control_character>

Varis autors (2016, 8 de novembre). «Transmission control». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Control_character#Transmission_control>

Varis autors (2016, 8 de novembre). «Data structuring». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Control_character#Data_structuring>

Varis autors (2017, 14 de març). «Abstract Syntax Notation One». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Abstract_Syntax_Notation_One>

Varis autors (2017, 5 de març). «Flow control (data)». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Flow_control_(data)>

Varis autors (2017, 5 de març). «Stop-and-wait». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Flow_control_(data)#Stop-and-wait>

Varis autors (2017, 5 de març). «Transmit flow control». *Viquipèdia* [article]. [Data de consulta: 17 de març de 2017].
<https://en.wikipedia.org/wiki/Flow_control_(data)#Transmit_flow_control>

Varis autors (2017, 28 de febrer). «Master/slave (technology)». *Viquipèdia* [article]. [Data de consulta: 20 de març de 2017].
<https://en.wikipedia.org/wiki/Master/slave_(technology)>

Varis autors. *Stackoverflow*. [Data de consulta: 23 de març de 2017].
<http://stackoverflow.com/questions/900950/how-to-send-data-to-com-port-using-java>

Varis autors (2017, 22 de març). «Markup language». *Viquipèdia* [article]. [Data de consulta: 25 de març de 2017].
<https://en.wikipedia.org/wiki/Markup_language>

Varis autors (2017, 20 de febrer). «Forward compatibility». *Viquipèdia* [article]. [Data de consulta: 25 de març de 2017].
<https://en.wikipedia.org/wiki/Forward_compatibility>

Varis autors (2017, 13 de març). «Formal database modeling technologies». *Viquipèdia* [article]. [Data de consulta: 25 de març de 2017].
<https://en.wikipedia.org/wiki/Cardinality_(data_modeling)#Formal_database_modeling_technologies>

Varis autors (2017, 10 de febrer). «Time to live». *Viquipèdia* [article]. [Data de consulta: 25 de març de 2017].

<https://en.wikipedia.org/wiki/Time_to_live>

Venkataram, P.; Manvi, S. S.; Babu, S. S. (2014). «Communication Protocol Engineering» (2a. ed.). Delhi: PHI Learning.

Implementació

Introducció

En aquest capítol es descriu el desenvolupament de la implementació d'exemple de la primera versió del protocol de proves.

Hi ha un primer apartat amb les decisions d'anàlisi preses, prèviament a la codificació.

En les tres seccions següents es descriu l'estructura principal del projecte d'Eclipse.

En el pròxim apartat, ja s'entren amb els detalls més concrets realitzats sobre la implementació, amb una petita introducció sobre una de les diferències entre llenguatges de programació i protocols de comunicació.

Després, es troba una secció amb una mostra dels fitxers d'exemple amb la implementació més específica del protocol, tot indicant quines parts concretes estan afectades per les característiques del protocol.

A continuació, hi ha un petit apartat annexe on s'analitza i es defineix un model general de test de protocols, amb un conjunt de requisits necessaris per a portar a terme aquesta tasca del desenvolupament.

Implementació del protocol de proves

Decisions prèvies d'anàlisi

A continuació, es descriuen les decisions preses sobre varis assumptes relatius a la implementació, abans de procedir al seu desenvolupament.

Disseny de la solució:

- *Estàtica amb metaprogramació i compilació: més ràpid i modificable*
- *Dinàmica: configurable i correcció en producció, però més lenta en execució*

L'opció estàtica és millor perquè és com funcionen les eines de compiladors de compiladors i aquest és l'objectiu inicial plantejat per aquest TFG.

Però, de totes maneres, és interessant la funcionalitat que atorga una solució dinàmica de poder corregir ràpidament en producció petits errors de programació.

Així mateix, també poder configurar en producció certs protocols molt simples que disposen d'un sol missatge, com sortides d'impressora o formats de fitxer senzills tipus CSV, els quals són altament configurables. Amb aquests protocols no té sentit crear implementacions que formin part com a component fix d'una aplicació.

I per a aconseguir-ho amb una solució estàtica, només caldria disposar de les eines de compilació en producció.

Capa d'alt nivell amb esdeveniments del protocol (via interfície Java):

- *Genèrics: onMessageReceived*
- *Específics: per exemple, onChargeReceived*

El què és segur, és l'esdeveniment genèric onMessageReceived, independentment de si s'implementen els esdeveniments específics de cada protocol.

De totes formes, afegir aquests esdeveniments resulta relativament fàcil. L'únic que cal realitzar és incorporar-los a la classe que implementi la interfície Java amb el mètode general onMessageReceived. I, més tard, cridar-los des de les accions de la gramàtica definida en CUP.

En la versió 0.4 del protocol de proves s'especifica que:

"Per altra banda, el lògic seria implementar primer la característica de no connexió amb el canal del port sèrie, però com que Java no subministra una implementació nativa, s'ajorna per a una versió posterior."

Com que en la tasca anterior de preparació prèvia de l'entorn de desenvolupament, ja s'ha avançat l'assumpte de la comunicació sèrie, ara ja es disposa de dues eines per a aquest menester. Raó per la qual ja es podria

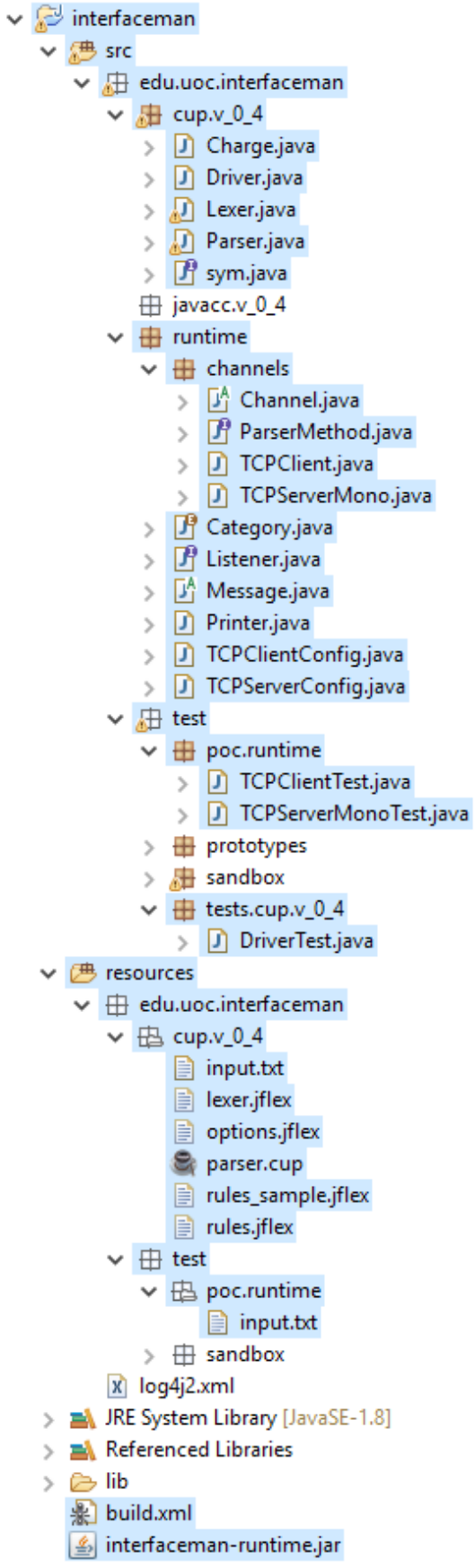
començar directament amb la característica de la no connexió, el que ho faria més fàcil.

De totes formes, avui en dia, la tendència és evolucionar cap a protocols de xarxa en lloc de protocols sèrie. Ja que, actualment, els cables de xarxa solen arribar a cada estació de treball, a diferència dels cables sèrie. Això és important quan la distància entre els dos sistemes és llarga i, no només això, sinó que el cable sèrie té limitacions de distàncies màximes que el cable de xarxa no sol tenir. Per tant, és més interessant avui treballar primer amb protocols de xarxa.

I una segona raó que es troba és que, si analitzem la programació que cal realitzar per a cada canal de transmissió –inclosos els que no estan orientats a la connexió, com el port sèrie, els fitxers o UDP per xarxa– en realitat, per a tots els casos, cal aprovisionar primer el recurs, previ a la comunicació real, i després també cal alliberar-lo. És a dir, sempre hi ha un pas inicial per a obrir el port sèrie, fitxer o inclús el sòcol UDP, i un pas final per a tancar-lo. El que ve a ser el mateix que l'establiment de connexió amb TCP per xarxa. I cal fixar-se també que en tots els altres canals, qualsevol operació de lectura i escriptura, potencialment pot generar un error, el que és equivalent a perdre la connexió amb TCP. Resumint, no hi ha cap diferència, en aquest aspecte, entre tots els canals analitzats amb la mecànica de les seves operatòries.

És per totes aquestes raons, que s'ha decidit continuar amb el pla original en l'evolució del protocol.

Estructura principal del projecte



Arxius i directoris del projecte

Breu descripció dels arxius i directoris més rellevants del projecte:

- \interfaceman\
  Arrel del projecte
  - *build.xml*
    Ant Build que genera les classes del compilador del paquet indicat a partir dels fitxers .jflex i .cup que conté
  - *interfaceman-runtime.jar*
    Arxiu JAR per als projectes que implementen els protocols i que conté les classes del paquet runtime de suport als protocols

  - \resources\edu.uoc.interfaceman.
    Directori de recursos que no són fonts Java
    - .cup.v_0_4.
      Paquet amb la versió 0.4 del protocol de prova
      - *input.txt*
        Fitxer amb entrades d'exemple (una errònia) pel compilador
      - *lexer.jflex*
        Fitxer JFlex principal amb la definició lèxica del protocol
      - *options.jflex*
        Opcions i declaracions comuns per a tots els lexers, per exemple, mètodes Java, macros tipus camps, estats comuns
      - *rules.jflex*
        Regles lèxiques comuns per a tots els lexers, per exemple, la gestió d'errors
      - *rules_sample.jflex*
        Exemples de regles lèxiques segons el tipus de camps
      - *parser.cup*
        Fitxer CUP amb la definició sintàctica i semàntica del protocol, per exemple, mètodes Java, símbols tipus camps i separadors, gramàtica, accions
    - .test.poc.runtime.
      - *input.txt*
        Fitxer amb l'entrada «Hello World!» usats pels tests del paquet «poc.runtime»

  - \src\edu.uoc.interfaceman.
    Directori amb els fonts Java
    - .cup.v_0_4.
      Paquet amb la versió 0.4 del protocol de prova

- *Charge.java*

  Missatge que representa a un càrrec rebut o a enviar (serialitzar)
- *Driver.java*

  Programa principal que inicia el procés del protocol i conté la capa d'alt nivell que processa el *payload* dels missatges rebuts
- *Lexer.java*

  Classe amb el lexer generat per JFlex
- *Parser.java*

  Classe amb el parser generat per CUP
- *sym.java*

  Taula de símbols generada per CUP

- .runtime.

  Paquet amb el *runtime* de suport als protocols
  - *Category.java*

    Categories disponibles pel logging
  - *Listener.java*

    Interfície per tractar els missatges rebuts de forma general amb esdeveniments
  - *Message.java*

    Classe abstracta que encapsula un missatge rebut o per enviar (serialitzar) que formata els diferents tipus de camps suportats
  - *Printer.java*

    Utilitat per escriure texts per la sortida estàndard i d'error
  - *TCPClientConfig.java*

    Propietats de configuració per a un client TCP (host i port TCP)
  - *TCPServerConfig.java*

    Propietats de configuració per a un servidor TCP (port TCP)

  - .channels.

    Paquet amb els diferents canals de transmissió suportats
    - *Channel.java*

      Classe abstracta a implementar per qualsevol canal de transmissió amb dos mètodes per enviar un missatge o un fitxer
    - *ParserMethod.java*

      Interfície funcional per a implementar pel parser
    - *TCPClient.java*

      Canal de transmissió per a un client TCP
    - *TCPServerMono.java*

Canal de transmissió per a un servidor TCP monoclient

- .test.

  Paquet amb codi de test

  - .poc.runtime.

    Paquet amb proves de concepte del paquet *runtime*

    - *TCPClientTest.java*

      Test pel canal TCPClient que envia un fitxer i espera rebre un byte

    - *TCPServerMonoTest.java*

      Test pel canal TCPServerMono que realitza el mateix test que el client

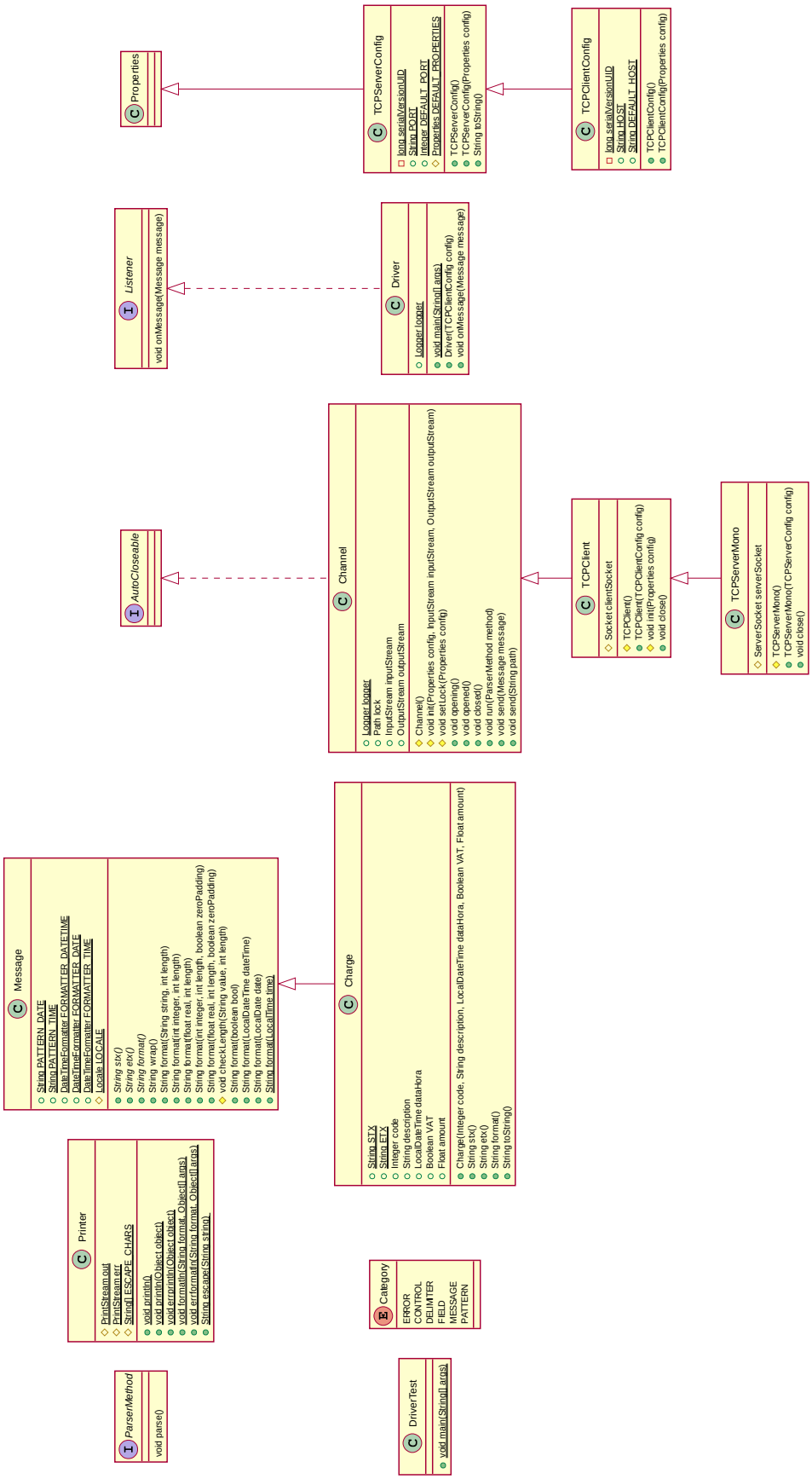
  - .tests.cup.v_0_4.

    Paquet que prova la versió 0.4 del protocol

    - *DriverTest.java*

      Prova alhora el protocol del client i el servidor

Diagrama UML de les principals classes



Detalls de la implementació

Abans d'entrar en detall, explicarem com s'ha resolt la primera diferència de la compilació de llenguatges de programació respecte els protocols de comunicació que s'ha de considerar en el desenvolupament. Recordem quina és:

> *La primera diferència que trobem és que, en molts protocols textuals, cada token té una longitud fixa, a diferència dels llenguatges de programació que són variables i utilitzen separadors com, per exemple, l'espai, el tabulador, el punt i coma o el salt de línia. Per tant, és necessari trobar una forma d'expressar els patrons de les expressions regulars amb uns quantificadors especificats amb un nombre concret, en lloc de nombres variables com ? (0 o 1), * (0 o més) i + (1 o més).*

Per sort, no ha calgut modificar el codi font de l'analitzador lèxic, perquè aquest ja disposa d'un sintaxi per a especificar aquestes longituds fixes de la forma `{nombre_caràcters}`. Per exemple, per a especificar una expressió regular amb 5 dígits, es realitzaria de la següent forma: `[0-9]{5}` .

En la primera versió 0.4 del protocol dissenyat per a les proves, ja hem hagut d'usar aquesta característica. Passem ara a recordar les característiques d'aquesta primera versió i, a continuació, anirem relatant una per una com s'han anat resolent cadascuna d'elles.

- Canals: xarxa (TCP)
- Connexió: sí (permanent)
- Sentit de la comunicació: simplex
- Format: híbrid
- Senyalització: in-band
- Framing missatges lògics: separadors externs (end missatge) CR/LF/RS
- Sublayering:
  - Capa de baix nivell: medi físic (canal), connexió
  - Capa d'alt nivell (payload): tipus de missatges lògics (command), de/serialització d'estructures de dades textual, diccionaris:
    - Camps: de longitud fixa (aligning/padding)
    - Missatges: seqüència de camps ordenada (de longitud fixa)
  - Transaccions: únic command
- Arquitectura nodes: punt a punt, centralitzada (client/servidor), servidor monoclient, rols (un master i un slave)

En aquesta primera versió, es necessita implementar un nombre mínim de característiques indispensables per a ser mínimament funcional.

Per altra banda, el lògic seria implementar primer la característica de no connexió amb el canal del port sèrie, però com que Java no subministra una implementació nativa, s'ajorna per a una versió posterior.

Els tipus dels camps seran: string, booleà, enter, real, data i/o hora.

Els camps (i unic missatge i transacció) seran: codi article, descripció article, data/hora, iva inclòs, import.

En resum, aquest primer protocol és la típica sortida d'impressora.

Els canals de xarxa per TCP han quedat reflectits en les classes TCPClient i TCPServerMono del subpaquet runtime.channels.

El concepte de canal orientat a la connexió desapareix com a tal, per les raons que s'han argumentat en l'apartat d'anàlisi prèvia, tots els canals requereixen ser oberts abans. Però la característica de connexió permanent queda reflectida en els fets de que el canal és obert només instanciar-lo i no s'ha previst, per a aquesta versió, cap mecanisme específic de tancament ni reobertura. La comunicació d'un costat es finalitza automàticament, en el cas dels canals TCP, al ser tancada la connexió per l'altre costat de la comunicació, tant si aquest és el servidor com el client.

El sentit unidireccional de la comunicació simplex es troba en el servidor només envia missatges mentre que el client només els rep. Per tant, només ha calgut generar el compilador del costat del client, que és qui ha de parsejar i validar la sortida del servidor. El servidor no valida la seva pròpia sortida, ja que es considera validada en temps de desenvolupament, mentre que l'entrada del client sí cal validar-la, perquè és el que ens ve de l'altre costat i sobre el que no en tenim control en un entorn de producció.

El format híbrid dels missatges, camps de dades textuals i caràcters de control per la separació de cada missatge individual, es pot veure en els fitxers lexer.jflex i options.jflex. Per exemple:

```
digit = [0-9]
real  = [0-9 \-.]
date  = {digit}{2}\/{digit}{2}\/{digit}{4}
etx   = \r\n
//etx = \x1E    // RS
```

La senyalització *in-band* es demostra en els separadors pel *framing* i per l'únic canal de transmissió utilitzat.

I el *framing* dels missatges lògics que finalitzen cada missatge, caràcters CR/LF/RS, ja s'ha vist en la macro i regla lèxica etx dels fitxers jflex, per una part, i en el fitxer parser.cup, per l'altra part, amb els símbols ETX i CHARGE i la regla sintàctica CHARGE:

```
terminal             ETX;
non terminal Charge  CHARGE;
CHARGE ::= INTEGER:code STRING:description DATETIME:dataHora
BOOLEAN:VAT REAL:amount ETX
```

La separació en dues capes d'alt nivell (*payload*) i baix nivell (control de flux i *framing*), la trobarem en varis fitxers.

Per a la capa de baix nivell, el medi físic (canal) i la seva connexió, tenim els arxius jflex, parser.cup, subpaquet runtime.channels i classes Java TCPClientConfig, TCPServerConfig, Lexer, Parser i sym.

Per a la capa d'alt nivell (payload), la trobarem reflectida en els fitxers jflex, parser.cup i fonts Java Listener, Message, Driver, Charge, Lexer, Parser i sym.

El ara únic tipus de missatge lògic (command) es descobreix en la classe Charge i el fet que encara no tenim cap resposta en aquesta versió.

La deserialització d'estructures de dades textuals es visualitza mitjançant l'analitzador lèxic en els fitxers lexer.jflex, options.jlfex, rules_sample.jflex i Lexer.java. Per exemple:

```
  char      = .
  digit     = [0-9]
  boolean   = [01]
  int       = [0-9 \-]
  real      = [0-9 \-.]
  date      = {digit}{2}\/{digit}{2}\/{digit}{4}
  time      = {digit}{2}:{digit}{2}:{digit}{2}
  datetime  = {date}" "{time}
  // Exemples de regles segons el tipus de camps
  {char}{10}{ return newSymbol(STRING,    yytext()); }
  {int}{2}  { return newSymbol(INTEGER,   new
Integer(yytext().trim())); }
  {real}{6} { return newSymbol(REAL,      new Float(yytext())); }
  {boolean} { return newSymbol(BOOLEAN,   "1".equals(yytext())); }
  {datetime}{ return newSymbol(DATETIME,LocalDateTime.parse(yytext(),
Message.FORMATTER_DATETIME)); }
  {date}          {return newSymbol(DATE, LocalDate.parse(yytext(),
Message.FORMATTER_DATE)); }
  {time}          {return newSymbol(TIME, LocalTime.parse(yytext())); }
```

De forma més indirecta, també es troba en els arxius parser.cup, Parser.java, sym.java i Message.java.

La serialització és més simple i es localitza només en les classes Java Message i Charge.

El diccionari de camps es defineix dins de lexer.jflex, parser.cup i Charge.java. La longitud fixa ja hem indicat abans com s'aconsegueix en les regles lèxiques.

En quant al tema de l'*aligning* i el caràcter de *padding*, per la part lèxica, és complicat validar algunes combinacions d'alineació i farciment. Per aquesta raó, s'ha simplificat l'anàlisi lèxica, recolzant-se principalment en la conversió des d'string de Java. I per la part de la serialització d'aquests atributs, també s'ha simplificat reduint el nombre de casos més comuns dins de la classe Message. Aquests es troben explicats, més endavant, en la taula de tipus de camps.

El diccionari de missatges es declara amb les subclasses de Message i en el fitxer parser.cup, amb els símbols no terminals i les seves regles sintàctiques equivalents. Dins de cada missatge, la seqüència ordenada de camps és definida dins del mateix fitxer parser.cup i també dins del lexer.jflex. S'ha de destacar especialment que, en aquest últim arxiu, cal definir una sèrie d'estats per cada camp per a poder seguir l'ordre correcte. D'altra manera, la regla implícita *longest match input* de l'analitzador lèxic impediria escanejar correctament l'entrada. A continuació, s'especifica un exemple concret:

```
  %states F2 F3 F4 F5

  %%

  <YYINITIAL> {int}{2}      { yybegin(F2);        return newSymbol(INTEGER,     ...); }
  <F2>        {char}{10}    { yybegin(F3);        return newSymbol(STRING,      ...); }
  <F3>        {datetime}    { yybegin(F4);        return newSymbol(DATETIME,    ...); }
  <F4>        {boolean}     { yybegin(F5);        return newSymbol(BOOLEAN,     ...); }
  <F5>        {real}{6}     { yybegin(FE);        return newSymbol(REAL,...); }
  <FE>        {etx}         { yybegin(YYINITIAL); return newSymbol(ETX); }
```

Així s'aconsegueix restringir cada regla lèxica només a un estat concret. És a dir, fer correspondre cada camp només a una posició concreta dins del missatge. En el cas de l'exemple, al inici de cada missatge només es pot rebre un camp enter de dos dígits. Si és així, després només pot arribar un string de 10 caràcters. I així, fins a l'últim camp i després del separador de fi de missatge, on torna a començar la seqüència de camps.

Per últim, dins dels missatges, la seva longitud fixa ve definida implícitament per les longituds fixes dels camps que conté.

Respecte al tema de transaccions, al ser un protocol unidireccional i amb una única ordre, aquest assumpte s'ha deixat per a pròximes versions del protocol. A més, també dependrà de la decisió pendent sobre si cal validar també la sortida de cada costat de la comunicació o es planteja d'una altra forma. Dit d'altra manera, l'enfoc de les transaccions sempre és vist des del punt de vista d'un observador extern que contempla la seqüència de l'intercanvi dels diferents missatges entre els dos costats de la comunicació. No és clar ara com reflectir això en cada costat.

Per a l'arquitectura dels nodes, aquesta és punt a punt i centralitzada perquè el protocol implementat segueix el paradigma clàssic client/servidor. També s'ha desenvolupat el servidor només com a monoclient. I pels rols, el disseny del protocol defineix el servidor com a *master* i el client com a *slave*. Pel fet de ser unidireccional, no ha calgut crear el compilador del servidor, perquè aquest només envia i no rep cap missatge que validar. És a dir, la instància del servidor l'únic que executa és crear el canal TCPServerMono i enviar missatges.

Com s'ha pogut veure, en aquesta primera versió, s'ha necessitat desenvolupar un nombre mínim de característiques indispensables per a ser mínimament funcional. Les quals estan reflectides principalment en el subpaquet runtime i en els arxius comuns options.jflex i rules.jflex, gràcies a la capacitat d'incloure fitxers en JFlex.

La qüestió de la no connexió amb el canal del port sèrie, ja ha sigut comentada anteriorment.

En quant als tipus dels camps definits i els seus atributs són els següents:

Tipus	Longitud	Alineació	*Padding*
string	variable	esquerra	espai
integer	variable	dreta	espai / zero
real	variable	dreta	espai / zero
boolean	fixa		
datetime	fixa		
date	fixa		
time	fixa		

En aquesta versió, la longitud variable vol dir que no és fixa per a tots els camps d'aquest tipus, però si és fixa per a cada camp definit. És a dir, el camp descripció sempre té una longitud fixa de deu caràcters, però un altre camp string podria tenir una longitud de cinc, per exemple. En canvi, qualsevol camp de tipus boolean sempre ocuparà un caràcter. En una versió posterior, la

longitud variable serà realment variable per a cada camp, al utilitzar separadors de camps que ho permetran.

L'atribut d'alineació s'ha predefinit amb els casos usuals dels texts alineats a l'esquerra i els nombres, enters i reals, a la dreta. I pel caràcter de *padding*, en els casos de valors més curts de la longitud màxima definida, l'espai està predefinit pel tipus string i pels nombres, permet escollir entre espais o zeros a l'esquerra.

Els fitxers del projecte afectats pels tipus de camps són lexer.jflex, options.jflex, rules_sample.jflex, parser.cup, sym.java, Message.java i Charge.java.

Finalment, la descripció informal dels camps de l'únic missatge d'aquest primer protocol típic de sortida d'impressora és:

- codi d'article (tipus int de longitud dos)
- descripció d'article (tipus string de longitud deu)
- data i hora (tipus datetime)
- IVA inclòs (tipus boolean)
- import (tipus real de longitud sis)

De forma breu, ara es detallarà la implementació realitzada dels aspectes del *logging* i els bloquejos.

En quant al *logging*, el nou fitxer Category.java permet categoritzar els missatges del *logging* en varis tipus: error, control, delimitadors, camps i missatges de protocol.

El Lexer, Channel i Driver ara usen el *logger* enlloc del Printer. En el Parser, de moment, com abans, no ha calgut utilitzar el *logger*, ja que ho fa indirectament a través del Driver. Els missatges de *logging* també indiquen qui d'aquests és el responsable d'emetre cada missatge per localitzar on pot residir qualsevol problema detectat.

La classe Printer incorpora un nou mètode de suport al *logging* per a traduir els caràcters de control a les seves abreviatures, ex. <CR><LF>.

El fitxer log4j2.xml ha sigut modificat per a mostrar els errors de protocol en vermell per la consola, usant un *stream* de sortida diferent pels errors, System.err en lloc de System.out. Al usar dos *streams* diferents té la conseqüència que pot provocar que els missatges d'error apareguin desordenats respecte als normals. També, el fet de que el *lexer* utilitzi *buffers* de lectura, *pushback* i *lookahead* provoca que l'ordre d'alguns missatges normals apareguin desordenats entre ells respecte de com s'esperaria. És un inconvenient que caldrà investigar de cara al futur.

Un altre aspecte a tenir en compte en un *log* comú compartit entre els dos costats de la comunicació, per exemple en DriverTest.java, són els *buffers* propis del sistema operatiu, que poden provocar que un costat tanqui el seu canal, mentre l'altre costat encara no ha llegit totes les dades, ja que aquestes romanen pendents en un *buffer* de recepció. Això s'accentua, sobretot amb protocols asíncrons o unidireccionals.

I respecte, al assumpte dels bloquejos, s'ha creat el nou fitxer ParserMethod.java, que és una interfície funcional (per *lambda expression* en Java 8) per a implementar pel *parser* i que permet facilitar el bloqueig automàtic per a tots els canals, per part de la classe Channel.

Destacar que s'ha utilitzat un bloqueig via fitxer en lloc de memòria, perquè no és suficient bloquejar (llegeixis impedir) la comunicació del mateix protocol entre els diferents *threads* d'un mateix procés, sinó que cal fer-ho també entre processos diferents. Per a aconseguir-ho, es crea un fitxer temporal per a cada canal configurat. És a dir, per exemple, un mateix protocol de tipus monoclient executat dos cops, però per dos ports TCP diferents, no ha d'impedir l'execució del segon procés. Per l'altre costat, dos protocols diferents, però pel mateix port TCP, sí s'ha d'impedir la segona execució. Apart de que això indicaria una mala configuració del mateix port per a servidors diferents.

Comentar també, que cada sistema operatiu incorpora els seus propis mecanismes de bloqueig segons el canal de comunicació. Per exemple, el *socket* servidor o el port sèrie en Windows. Altres com el *socket* client, els fitxers o el port sèrie en Linux no s'impedeix de forma automàtica la seva obertura compartida. En altres casos, existeixen modes d'obertura especials, com per exemple l'accés exclusiu per fitxers en Windows.

Finalment, l'arxiu interfaceman-runtime.jar inclou la part del *runtime* de suport per als projectes que incorporin els protocols generats.

Fitxers d'exemple d'implementació del protocol

Aquesta és una mostra dels fitxers afectats per a implementar un protocol. S'indiquen amb un altre color les parts que depenen de cada protocol.

lexer.jflex

```
  package edu.uoc.interfaceman.cup.v_0_4;

  import java_cup.runtime.SymbolFactory;
  import java_cup.runtime.Symbol;
  import java.io.IOException;
  import java.time.*;
  import org.apache.logging.log4j.LogManager;
  import org.apache.logging.log4j.Logger;
  import static edu.uoc.interfaceman.runtime.Category.*;
  import edu.uoc.interfaceman.runtime.channels.Channel;
  import edu.uoc.interfaceman.runtime.*;

  %%

  // Opcions comuns per a tots els lexers
  %include options.jflex

  etx = \r\n

  %states F2 F3 F4 F5

  %%

  <YYINITIAL>   {int}{2}  { yybegin(F2);  return newSymbol(INTEGER, new Integer(yytext().trim())); }
  <F2>          {char}{10}{ yybegin(F3);  return newSymbol(STRING,  yytext()); }
  <F3>          {datetime}{ yybegin(F4);  return newSymbol(DATETIME,LocalDateTime.parse(yytext(),
                                                    Message.FORMATTER_DATETIME)); }
  <F4>          {boolean} { yybegin(F5);  return newSymbol(BOOLEAN, "1".equals(yytext())); }
  <F5>          {real}{6} { yybegin(FE);  return newSymbol(REAL,    new Float(yytext())); }
  <FE>          {etx}     { yybegin(YYINITIAL); return newSymbol(ETX); }

  .                   { yybegin(ERROR);
                        logger.error(PATTERN,   Category.ERROR,   String.format("Wrong
character (ignored): '%s'", yytext())); }
  // Regles comuns per a tots els lexers
  %include rules.jflex
```

parser.cup

```
  package edu.uoc.interfaceman.cup.v_0_4;

  import java_cup.runtime.*;
  import edu.uoc.interfaceman.cup.v_0_4.Lexer;
  import edu.uoc.interfaceman.cup.v_0_4.Charge;
  import edu.uoc.interfaceman.runtime.channels.Channel;
  import edu.uoc.interfaceman.runtime.Listener;
  import java.time.*;

  parser code {:
       Listener listener;

       public Parser(Channel channel, Listener listener) throws Exception {
              super();
              channel.run(() -> {
                     symbolFactory = new ComplexSymbolFactory();
                     setScanner(new Lexer(channel, symbolFactory));
                     this.listener = listener;
                     parse();
              });
       }
  :};
```

```
terminal String         STRING;
terminal Boolean        BOOLEAN;
terminal Integer        INTEGER;
terminal Float          REAL;
terminal LocalDate      DATE;
terminal LocalTime      TIME;
terminal LocalDateTime  DATETIME;
terminal                ETX;

non terminal            LIST;
non terminal Charge     CHARGE;

LIST ::=      LIST    CHARGE:charge    {: listener.onMessage(charge); :}
              |       CHARGE:charge    {: listener.onMessage(charge); :};
CHARGE ::=    INTEGER:code STRING:description DATETIME:dataHora BOOLEAN:VAT REAL:amount ETX
                      {: RESULT = new Charge(code, description, dataHora, VAT, amount); :};
```

Charge.java

```
package edu.uoc.interfaceman.cup.v_0_4;

import java.time.LocalDateTime;

import edu.uoc.interfaceman.runtime.Message;

/**
 * Missatge que representa a un càrrec.
 */
public class Charge extends Message {

    /** Inici de missatge. */
    public static final String STX = "";
    /** Fi de missatge. */
    public static final String ETX = "\r\n";

    /** Codi d'article. */
    public Integer code;
    /** Descripció d'article. */
    public String description;
    /** Data i hora del càrrec. */
    public LocalDateTime dataHora;
    /** IVA inclòs. */
    public Boolean VAT;
    /** Import del càrrec. */
    public Float amount;

    /**
     * Nou càrrec.
     *
     * @param code codi d'article.
     * @param description descripció d'article.
     * @param dataHora data i hora del càrrec.
     * @param VAT IVA inclòs.
     * @param amount import del càrrec.
     */
    public Charge(Integer code, String description, LocalDateTime dataHora,
                  Boolean VAT, Float amount) {
        this.code           = code;
        this.description    = description;
        this.dataHora       = dataHora;
        this.VAT            = VAT;
        this.amount         = amount;
    }

    @Override
    public String stx() {
        return STX;
```

```
        }

        @Override
        public String etx() {
                return ETX;
        }

        @Override
        public String format() {
                return new StringBuilder  (format(code, 2)).append
                                          (format(description, 10)).append
                                          (format(dataHora)).append
                                          (format(VAT)).append
                                          (format(amount, 6)).toString();
        }

        @Override
        public String toString() {
                return  String.format("Charge  [code=%s,  description=%s,  dataHora=
%s, VAT=%s, amount=%s]",
                                      code, description, dataHora, VAT, amount);
        }
    }
```

Driver.java

```
    package edu.uoc.interfaceman.cup.v_0_4;

    import static edu.uoc.interfaceman.runtime.Category.*;

    import java.io.IOException;
    import java.net.UnknownHostException;

    import org.apache.logging.log4j.LogManager;
    import org.apache.logging.log4j.Logger;

    import edu.uoc.interfaceman.runtime.Listener;
    import edu.uoc.interfaceman.runtime.Message;
    import edu.uoc.interfaceman.runtime.TCPClientConfig;
    import edu.uoc.interfaceman.runtime.channels.TCPClient;

    /**
     * Programa principal.
     */
    public class Driver implements Listener {

        /** Logger d'ús comú per a totes les capes. */
        public static final Logger logger =
                            LogManager.getLogger(Driver.class.getSimpleName());

        public static void main(String[] args) throws Exception {
                new Driver(new TCPClientConfig());
        }

        /**
         * Inicia el procés del protocol.
         *
         * @param config propietats de configuració.
         * @throws UnknownHostException si no es pot determinar l'adreça IP
         *             del servidor.
         * @throws IOException si es produeix un error de comunicació.
         * @throws Exception si es produeix qualsevol altre error.
         */
        public Driver(TCPClientConfig config) throws UnknownHostException,
                                                     IOException, Exception {
                logger.info(PATTERN, CONTROL, "Starting communication");
                new Parser(new TCPClient(config), this);
```

```
            logger.info(PATTERN, CONTROL, "Stopped communication");
        }

        @Override
        public void onMessage(Message message) {
            logger.debug(PATTERN, MESSAGE, message);
            // TODO processar el missatge rebut (ex. guardar a la BD)
        }
    }
```

Disseny del metallenguatge

En aquest apartat es descriu la solució final amb la implementació del metallenguatge de la primera versió del protocol de proves.

A partir dels fitxers d'implementació manual, se n'han extret tots els textos i s'indiquen amb un altre color les parts úniques a generar per a tots els fitxers. La resta és codi repetit, de forma directa o indirecta.

```
package edu.uoc.interfaceman.cup.v_0_4;
etx = \r\n
%states F2 F3 F4 F5
<YYINITIAL>    {int}{2}  { yybegin(F2);  return newSymbol(INTEGER,new Integer(yytext().trim())); }
<F2>           {char}{10}{ yybegin(F3);  return newSymbol(STRING, yytext()); }
<F3>           {datetime}{ yybegin(F4);  return newSymbol(DATETIME,LocalDateTime.parse(yytext(),

      Message.FORMATTER_DATETIME)); }
<F4>           {boolean} { yybegin(F5);  return newSymbol(BOOLEAN,"1".equals(yytext())); }
<F5>           {real}{6} { yybegin(FE);  return newSymbol(REAL,    new Float(yytext())); }
<FE>           {etx}     { yybegin(YYINITIAL); return newSymbol(ETX); }
import edu.uoc.interfaceman.cup.v_0_4.Lexer;
import edu.uoc.interfaceman.cup.v_0_4.Charge;
Missatge que representa a un càrrec.
Charge
/** Codi d'article. */
public Integer code;
/** Descripció d'article. */
public String description;
/** Data i hora del càrrec. */
public LocalDateTime dataHora;
/** IVA inclòs. */
public Boolean VAT;
/** Import del càrrec. */
public Float amount;
/**
 * Nou missatge que representa a un càrrec.
 *
 * @param code codi d'article.
 * @param description descripció d'article.
 * @param dataHora data i hora del càrrec.
 * @param VAT IVA inclòs.
 * @param amount import del càrrec.
 */
public Charge(Integer code, String description, LocalDateTime dataHora, Boolean VAT,Float amount){
      this.code        = code;
      this.description = description;
      this.dataHora    = dataHora;
      this.VAT         = VAT;
      this.amount      = amount;
}
(format(code, 2)).append
(format(description, 10)).append
(format(dataHora)).append
(format(VAT)).append
(format(amount, 6))
"Charge [code=%s, description=%s, dataHora=%s, VAT=%s, amount=%s]",
              code, description, dataHora, VAT, amount
import java.net.UnknownHostException;
import edu.uoc.interfaceman.runtime.TCPClientConfig;
import edu.uoc.interfaceman.runtime.channels.TCPClient;
TCPClientConfig
TCPClientConfig
* @throws UnknownHostException si no es pot determinar l'adreça IP del servidor.
UnknownHostException,
TCPClient
// TODO processar el missatge rebut (ex. guardar a la BD)
```

A partir d'aquests textos trobats, es dissenya la següent especificació informal del llenguatge:

```
package       edu.uoc.interfaceman.cup.v_0_4
channel       mode
etx           control+
field+        name type [length] [0] [«desc»]
message       name [«desc»] field+ [{codi_java}]
```

```
{mode         tcp_client|tcp_server}
{control      CR|LF|RS}
{type         string|integer|real|boolean|datetime|date|time}
```

Els textos subratllats indiquen valors a indicar pel desenvolupador, segons les característiques de cada protocol. El mode del canal ara només permet indicar si és un client o servidor TCP. La declaració etx permet indicar un o més caràcters de control dels suportats. La llista de camps obliga a definir almenys un camp. Dins d'aquesta regla, els paràmetres length i desc són opcionals segons el tipus del camp. El missatge obliga a definir almenys un camp del missatge. El codi Java és opcional i permet declarar el codi que s'executarà quan arribi un missatge.

DSL.xtext

A continuació, l'especificació formal de la gramàtica del metallenguatge ideat:

```
grammar edu.uoc.interfaceman.dsl.DSL with org.eclipse.xtext.common.Terminals

generate dSL "http://www.uoc.edu/interfaceman/dsl/DSL"

Protocol:
     package=Package
     channel=Channel
     etx=ETX
     fields=Fields
     message=Message;


Package:
     'package' name=QualifiedName;

QualifiedName:
     ID ('.' ID)*;


Channel:
     'channel' name=Mode;

Mode:
     'tcp_client' | 'tcp_server';


ETX:
     'etx' name=Control;

Control:
     ('CR' | 'LF' | 'RS')+;


Fields:
     name+=Field+;

Field:
     'field' name=ID type=Type desc=STRING?;


Type:
     _String | Integer | Real | Boolean | DateTime | Date | Time;

_String:     name='string'  length=INT;
Integer:     name='integer' length=INT zero?='0'?;
Real:        name='real'           length=INT zero?='0'?;
Boolean:     name='bool';
DateTime:    name='datetime';
Date:        name='date';
Time:        name='time';
```

```
Message:
     'message' name=ID desc=STRING? fields=FieldsList body=CODE_BLOCK?;

FieldsList:
     name+=[Field]+;

terminal CODE_BLOCK : '{' -> '}';
```

Eines de test

Per a definir un model general de test del protocol, els requisits necessaris haurien de ser:

- Disposar dels programes simuladors dels dos extrems de la comunicació
  - Sistema extern contra el que volem comunicar
  - Sistema intern propi de la nostra aplicació
- Accions
  - Manuals
    - Configuració del canal (o canals disponibles en el protocol)
    - Obertura i tancament del canal (o canals disponibles en el protocol)
    - Enviament de qualsevol missatge del protocol
    - Modificació dels camps dels missatges
    - Recepció de qualsevol missatge del protocol
    - Enviament de resposta positiva («ACK») o negativa («NAK»)
    - Enviament de caràcters de control com «ENQ»
    - Proves negatives per provar la robustesa de la implementació
      - Enviaments fora de sincronia
      - Enviament de checksums erronis
  - Automàtiques
    - Configuració per defecte del canal (o canals disponibles en el protocol)
    - Obertura i tancament del canal (o canals disponibles en el protocol) al entrar i sortir del programa
    - Valors per defecte dels camps dels missatges
    - Emmagatzemament del últim missatge enviat per reenviament manual
    - Recepció de qualsevol missatge amb enviament de resposta predefinida («ACK») o («NAK») de forma contínua o un cop cada vegada
    - Transaccions i/o sessions completes amb recepció i respostes predefinides

Tots aquests són els requisits mínims o, almenys, és desitjable disposar de la majoria d'ells per a executar les proves amb confiança. És important, sobretot, adonar-se de la necessitat de poder realitzar tests manuals i poder provocar errors en els enviaments simulats del sistema extern. La nostra implementació del protocol no només ha de fer el que s'espera, si tot va bé, sinó que també no ha de fer el que no s'espera que faci, si rebem res malament per part de l'altre extrem.

Aquests programes es poden (meta)programar ràpida i fàcilment, ja que no requereixen una implementació autònoma ni completa perquè no formen part del producte final.

Bibliografia

Benz, S.; Engelmann, B. (2014, 25 de novembre). «20 Facts about Xtend». *BMW Car IT GmbH* [documentació]. [Data de consulta: 10 de juny de 2017].

<http://jnario.org/org/jnario/jnario/documentation/20FactsAboutXtendSpec.html>

Bettini, L. (2016). «Implementing Domain-Specifc Languages with Xtext and Xtend» (2a. ed.). Birmingham: Packt Publishing.

Green, M. D. (2014, 30 de juny). «Using Regular Expressions to Check String Length». *Sitepoint.com* [article]. [Data de consulta: 29 d'abril de 2017].

<https://www.sitepoint.com/web-foundations/using-regular-expressions-to-check-string-length/>

Hedgecock, W. (2016, 18 de febrer). «jSerialComm». *Fazecast* [manual]. [Data de consulta: 22 de maig de 2017].

<http://fazecast.github.io/jSerialComm/>

Klein, G.; Rowe, S.; Décamps, R. (2015, 20 d'abril). «JFlex User's Manual». *JFlex.de* [manual]. [Data de consulta: 29 d'abril de 2017].

<http://jflex.de/manual.html>

Petter, M. «CUP». *Technischen Universität München* [documentació]. [Data de consulta: 12 de juny de 2017].

<http://www2.cs.tum.edu/projects/cup/>

Simon Tuffs, P. (2007, 26 de febrer). «Deliver Your Java Application in One-JAR!». *Sourceforge* [article]. [Data de consulta: 12 de juny de 2017].

<http://one-jar.sourceforge.net/version-0.95/>

Sokolov, A. «java-simple-serial-connector». *Google Code* [manual]. [Data de consulta: 22 de maig de 2017].

<https://code.google.com/archive/p/java-simple-serial-connector/>

Trætteberg, H.; arnaudroques «plantuml - generate UML diagrams from files and view them in Eclipse». *Github*. [Data de consulta: 29 d'abril de 2017].

<https://github.com/hallvard/plantuml>

Varis autors (2015). «The Java™ Tutorials». *Oracle* [documentació]. [Data de consulta: 29 d'abril de 2017].

<https://docs.oracle.com/javase/tutorial/>

Varis autors. «Regex to read fixed width numeric fields». *Stackoverflow*. [Data de consulta: 29 d'abril de 2017].

<http://stackoverflow.com/questions/5116422/regex-to-read-fixed-width-numeric-fields>

Varis autors. «Java + Eclipse: Synchronize stdout and stderr». *Stackoverflow*. [Data de consulta: 29 d'abril de 2017].

<http://stackoverflow.com/questions/2896908/java-eclipse-synchronize-stdout-and-stderr#4200604>

Varis autors (2017, 24 de maig). «15 Minutes Tutorial». *Xtext* [documentació]. [Data de consulta: 13 de maig de 2017].

<http://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html>

Varis autors (2017, 24 de maig). «15 Minutes Tutorial - Extended». *Xtext* [documentació]. [Data de consulta: 13 de maig de 2017].

<http://www.eclipse.org/Xtext/documentation/103_domainmodelnextsteps.html>

Varis autors (2017, 24 de maig). «Five simple steps to your JVM language». *Xtext* [documentació]. [Data de consulta: 13 de maig de 2017].

<http://www.eclipse.org/Xtext/documentation/104_jvmdomainmodel.html>

Varis autors. «Sharing a serial port between two processes». *Superuser*. [Data de consulta: 22 de maig de 2017].

<https://superuser.com/questions/488908/sharing-a-serial-port-between-two-processes#489554>

Varis autors (2001, 5 de gener). «Accessing Serial Devices». *Linux Network Administrators Guide* [manual]. [Data de consulta: 22 de maig de 2017].

<http://www.tldp.org/LDP/nag2/x-087-2-serial.devices.html>

Varis autors. «tty_ioctl». *Linux man* [manual]. [Data de consulta: 22 de maig de 2017].

<https://linux.die.net/man/4/tty_ioctl>

Varis autors. «Using java file locks within single JVM and across multiple JVMs». *Stackoverflow*. [Data de consulta: 26 de maig de 2017].

<https://stackoverflow.com/questions/5715346/using-java-file-locks-within-single-jvm-and-across-multiple-jvms>

Varis autors. «File lock between threads and processes». *Stackoverflow*. [Data de consulta: 26 de maig de 2017].

<https://stackoverflow.com/questions/28113009/file-lock-between-threads-and-processes>

Varis autors (2016, 12 de gener). «Package java.nio». *Oracle* [manual]. [Data de consulta: 26 de maig de 2017].

<https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html>

Varis autors (2016, 12 de gener). «Package java.nio.channels». *Oracle* [manual]. [Data de consulta: 26 de maig de 2017].

<https://docs.oracle.com/javase/8/docs/api/java/nio/channels/package-summary.html>

Varis autors (2016, 12 de gener). «Package java.nio.file». *Oracle* [manual]. [Data de consulta: 26 de maig de 2017].

<https://docs.oracle.com/javase/8/docs/api/java/nio/file/package-summary.html>

Varis autors (2016, 12 de gener). «Package java.io». *Oracle* [manual]. [Data de consulta: 26 de maig de 2017].

<https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html>

Varis autors (2017, 3 de març). «Xtend - Documentation». *Xtend* [documentació]. [Data de consulta: 10 de juny de 2017].

<https://eclipse.org/xtend/documentation/index.html>

Varis autors (2011, 29 de març). «JAR : MANIFEST.MF Class-Path referencing a directory». *Blogspot* [article]. [Data de consulta: 12 de juny de 2017].

<http://todayguesswhat.blogspot.com.es/2011/03/jar-manifestmf-class-path-referencing.html>

Varis autors. «Use of the MANIFEST.MF file in Java». *Stackoverflow*. [Data de consulta: 12 de juny de 2017].

<https://stackoverflow.com/questions/12767886/use-of-the-manifest-mf-file-in-java#12767929>

Varis autors (2016, 19 de juliol). «Adding Classes to the JAR File's Classpath». *Oracle* [documentació]. [Data de consulta: 12 de juny de 2017].

<https://docs.oracle.com/javase/tutorial/deployment/jar/downman.html>

Conclusions

Malgrat que ara només s'ha implementat la versió bàsica del protocol i que no seria molt difícil realitzar-ne una implementació *ad hoc*, observant el fitxer v_0_4.proto, és fàcil adonar-se que realment no cal programar codi en Java, excepte el tractament del missatge rebut, el qual també resideix en el fitxer d'especificació.

És molt més fàcil escriure un fitxer de configuració com el del metallenguatge que escriure codi de programació. Això té els següents avantatges:

- La tasca de parsejar i validar els missatges rebuts sempre és complicada de programar, feixuga de provar i propensa als errors, sobretot, si es fa de forma *ad hoc*.
- Quant menys codi es programi, menys probabilitat hi haurà de cometre errors.
- El codi generat pels analitzadors lèxic i sintàctic és menys propens a errors, perquè ja ha sigut provat per altres desenvolupadors. Per tant, menys temps dedicats a les proves d'aquest codi.
- Les regles lèxiques i sintàctiques es poden provar de forma immediata, ja que qualsevol canvi és fàcil d'afegir.

Per altra part, per a resoldre els casos que no pugui contemplar aquesta eina, es podria modificar el codi generat, però jo no em plantejaria modificar el codi generat pel compilador del metallenguatge. Primer, perquè no és codi generat per un programador, sinó per un programa. Malgrat que el codi Java generat propi és *human readable*, el de JFlex i CUP no ho és tant. Segon que crec que, en principi, la majoria de casos dels diferents protocols són assolibles via les corresponents regles dels compiladors i, per tant, hauria de ser possible cobrir qualsevol cas. Això, assumint que disposem d'un *runtime*, principalment, complet amb classes de suport. I, per últim, qualsevol canvi introduït al codi generat automàticament es perdria, si cal canviar el fitxers de JFlex i CUP i regenerar el codi. I mantenir aquests tipus de fonts automàtics i modificats manualment, sol ser generalment complicat d'aconseguir.

Objectius aconseguits

Amb tot això, s'ha aconseguit finalitzar el desenvolupament de la primera fita (nivell *must have* MoSCow), exceptuant el primer nivell del protocol de proves, del qual només se n'ha implementat la primera versió.

Era més interessant no implementar més versions del protocol, de moment, i passar al següent nivell de les tasques obligatòries de la segona fita (segon nivell MoSCow). El qual consistia en desenvolupar l'apartat del llenguatge DSL amb Xtext i generar automàticament el contingut dels fitxers jflex, cup i java.

Així, d'aquesta manera, s'ha tancat el cicle, ja que fins s'havia treballat la part *front-end* d'anàlisi dels compiladors i amb això altre també s'ha treballat amb la part del *back-end* de generació de codi objecte o codi font. I això també està més relacionat amb l'àrea de compiladors del TFG i menys relacionat amb la matèria dels protocols de comunicació.

Finalment, això també ha simplificat al màxim la tercera fita, que tractava de desenvolupar la interfície gràfica.

Continuïtat del projecte

Òbviament era difícil, si no impossible, haver abastat tots els requisits del projecte. De fet, quan es va fer la gestió dels requisits, ja n'era conscient d'haver sobredimensionat excessivament l'abast. D'aquí, la decisió d'haver triat com a metodologia de treball, des de l'inici, el mètode MoSCoW.

Però, la raó d'haver definit tots aquests requisits va ser aprofitar tota l'experiència acumulada de tots els anys professionals havent implementat més d'un centenar de protocols. Era una oportunitat per reescriure les mateixes idees de zero, però millorant tots aquells aspectes que ara faria d'una altra manera millor.

És per això que, aprofitant el que penso que és un bon pla de treball restant, crec que amb tota probabilitat continuaré el projecte i podré finalitzar-lo totalment en sis mesos o un any. I aprofitant el fet de que el metallenguatge és l'especificació més bàsica prevista, es mirarà d'usar un llenguatge d'especificació estàndard com, per exemple, SDL.

Llista de figures

ANNEX

Entorn de desenvolupament: Eclipse

Instal·lació i configuració d'Eclipse

1. Windows (64 bits)
2. JDK de Java 8 (64 bits)
    1. Descarregar Java SE Development Kit 8 Downloads
    2. Instal·lar
3. Eclipse IDE for Java and DSL Developers (64 bits)

    The essential tools for Java and DSL developers, including a Java & Xtend IDE, a DSL Framework (Xtext), a Git client, XML Editor, and Maven integration.
    1. Instal·lació
        1. Crear directori base (proposta «C:\UOC_TFG_jromerae» per a evitar problemes amb paths absoluts)
        2. Descarregar Eclipse IDE for Java and DSL Developers
        3. Descomprimir arxiu «eclipse-dsl-neon-3-win32-x86_64.zip»
        4. Crear arxiu «eclipse.bat»

            ```
            @echo off
            start eclipse\eclipse
            ```
        5. Crear carpeta «workspace»
        6. Executar arxiu «eclipse.bat»
        7. Assignar «workspace» per defecte
    2. Plugins
        1. Compilador JFlex/CUP (plugin eclipse)
            1. Opció de menú «Help/Install New Software...»
            2. Afegir l'adreça URL http://www2.in.tum.de/projects/cup/eclipse prement enter
            3. Seleccionar tot
            4. Clic en botons «Next» i «Finish»
            5. Seguir les instruccions
        2. Metallenguatge
            1. Xtext/Xtend: veure punt Eclipse més amunt
        3. Subversion (amb plugin subversive)
            1. Plugin
                1. Opció de menú «Help/Install New Software...»
                2. Seleccionar l'adreça URL «Neon - http://download.eclipse.org/releases/neon»
                3. Escriure «subversive» en el camp de filtre
                4. Seleccionar tot

5. Clic en botons «Next» i «Finish»
6. Seguir les instruccions

2. Connectors
    1. Obrir la perspectiva «SVN Repository Exploring»
    2. Seleccionar les últimes versions de «SVN kit» i «Native JavaHL»
    3. Clic en botons «Next» i «Finish»
    4. Seguir les instruccions
3. Configuració
    1. Opció de menú «Window/Preferences»
    2. «Team/Ignored Resources»
    3. «Add Pattern...»: «.settings»

4. PlantUML (plugin eclipse)
    1. Instal·lació
        1. Opció de menú «Help/Install New Software...»
        2. Afegir l'adreça URL http://files.idi.ntnu.no/publish/plantuml/repository/ prement enter
        3. Seleccionar tot
        4. Clic en botons «Next» i «Finish»
        5. Seguir les instruccions
        6. Descomprimir dir. «release/bin» d'arxiu «graphviz-2.38.zip» dins d'un dir. «graphviz»
    2. Configuració
        1. Opció de menú «Window/Preferences»
        2. Opció «PlantUML»
        3. Indicar el path al executable «dot.exe»: «...\graphviz\bin\dot.exe»

Creació i configuració del projecte

1. Subversion
   Crear repositori
    1. Perspective «SVN Repository Exploring»
    2. «SVN Repositories»
    3. «New Repository»: «UOC_TFG_jromerae/subversion»
    4. Botó dret sobre repositori «New/Project Structure...»
    5. «Monolithic layout»
    6. Entrar comentari «Estructura repositori»

2. Projecte Eclipse simple
    1. «New/Other/Cup/Cup Java Project»

1. «Project name»: «interfaceman»
2. Subversion
   Primers commits
   1. Botó dret sobre projecte «Team/Share Project...»
   2. «SVN»
   3. Seleccionar repositori creat abans
   4. «Simple Mode»
   5. «Browse»: «trunk»
   6. Entrar comentari «Commit inicial»
   7. Entrar comentari «Projecte buit»
3. «New/Package» en «src»
   Base: «edu.uoc.interfaceman.»
   .«(runtime o ~~common~~ o tools)»
   .«cup.v_0_4»
   .«javacc.v_0_4»
   .«test.(sandbox, prototypes?, poc?, tests (unitaris))»
   Subversion «Team/Commit...»: «Estructura paquets»
4. Subversion Tag (*)
   1. Perspective «SVN Repository Exploring»
   2. «SVN Repositories»
   3. Botó dret sobre repositori «Show History»
   4. «History»: «Refresh»
   5. Botó dret sobre última revisió «Tag from x...»
      «Tag»: «v.0.1»
      «Comment»: «V.0.1»

3. Sandbox de compiladors
   Moure prova CUP en «test.sandbox...» de «src» i «resources»
   1. Reconfigurar CUP
      1. «Propietats projecte/Builders/Remove .cupAntBuilder»
      2. Dependència «java-cup-11b-runtime.jar»
         1. Reanomenar dir. «tools» per «lib»
         2. Modificar directament fitxer «.classpath» (des de Navigator)
            1. «<classpathentry kind="lib" path="~~tools~~/java-cup-11b-runtime.jar"/>»
            2. «<classpathentry kind="lib" path="lib/java-cup-11b-runtime.jar"/>»
      3. Moure dir. «cup» dins de «sandbox.» (des de Navigator)
         1. «Quick Fix» del package en «Driver.java»
      4. «New/Source Folder»: «resources»
      5. «New/Package» en «resources»
         «edu.uoc.interfaceman.test.sandbox.cup.example»

1. Moure fitxers dins de package: «input.txt», «lexer.jflex», «parser.cup»
6. Modificar fitxer «build.xml»
7. Modificar fitxers
    1. «lexer.jflex»
        1. «package edu.uoc.interfaceman.test.sandbox...»
    2. «parser.cup»
        1. «package/import edu.uoc.interfaceman.test.sandbox...»
        2. «bin/edu/uoc/interfaceman/test/sandbox/cup/example/input.txt»
8. «Run As» sobre «build.xml»
   «edu/uoc/interfaceman/test/sandbox/cup/example»
9. Botó «External Tools/External Tools Configurations...»
    1. Pestanya «JRE»: «Run in the same JRE as the workspace»
    2. «Apply»
10. «Run» sobre «Driver.java»

Subversion «Team/Commit...»: «Reconfigurar CUP»

2. Subversion Tag (*)
   «Tag»: «v.0.2»
   «Comment»: «V.0.2»

4. Dependències
   Afegides manualment
    1. Java logging framework standard: Log4j
        1. Instal·lació
            1. Copiar de «apache-log4j-2.8.2-bin.zip» els fitxers «log4j-api-2.8.2.jar» i «log4j-core-2.8.2.jar» a «lib»
            2. Seleccionar ambdós i botó dret «Build Path/Add to Build Path»
               Subversion «Team/Commit...»: «Log4j instal·lació»
        2. Configuració
            1. «New/Other/XML/XML File» en «resources»
               «File name»: «log4j2.xml»
            2. Contingut per defecte de
               https://logging.apache.org/log4j/2.x/manual/configuration.html
               Subversion «Team/Commit...»: «Log4j configuració»
        3. Sandbox
            1. «New/Class» en subpaquet «edu.uoc.interfaceman.test.sandbox»
               «Logging.java»
            2. «Run»
               Subversion «Team/Commit...»: «Log4j sandbox»

2. Màquina d'estats amb StatefulJ
    1. Instal·lació
        1. Copiar els fitxers «statefulj-fsm-3.0.jar» i «statefulj-common-3.0.jar» a «lib»
        2. Copiar de «apache-log4j-2.8.2-bin.zip» el fitxer «log4j-slf4j-impl-2.8.2.jar» a «lib»
        3. Copiar de «slf4j-1.7.25.zip» el fitxer «slf4j-api-1.7.25.jar» a «lib»
        4. Seleccionar tots i botó dret «Build Path/Add to Build Path»

           Subversion «Team/Commit...»: «StatefulJ instal·lació»
    2. Sandbox
        1. «New/Class» en subpaquet «edu.uoc.interfaceman.test.sandbox»

           «StateMachine.java»
        2. «Run»

           Subversion «Team/Commit...»: «StatefulJ sandbox»

3. Comunicació sèrie (en ordre de preferència, tots amb events)
    1. jSerialComm (API més simple: no té senyals de hardware ni excepcions)
        1. Instal·lació
            1. Copiar el fitxer «jSerialComm-1.3.11.jar» a «lib»
            2. Botó dret «Build Path/Add to Build Path»

               Subversion «Team/Commit...»: «jSerialComm instal·lació»
        2. Configuració
            1. Botó dret en «Referenced Libraries» sobre «jSerialComm-1.3.11.jar» «Build Path/Configure Build Path...»
            2. «Edit» «Javadoc location»

               «Javadoc URL»: http://fazecast.github.io/jSerialComm/javadoc
            3. «Validate»

               Subversion «Team/Commit...»: «jSerialComm configuració»
        3. Sandbox
            1. «New/Package» en «src»

               «edu.uoc.interfaceman.test.sandbox.serial»
            2. «New/Class» en subpaquet

               «Serial.java»
            3. «Run»

               Subversion «Team/Commit...»: «jSerialComm sandbox»

2. java-simple-serial-connector (API simple: sí té senyals de hardware i excepcions, antiquat)
    1. Instal·lació
        1. Copiar de «jSSC-2.8.0-Release.zip» els fitxers «jssc.jar», «jssc-2.8.0-javadoc.jar» i «jssc-2.8.0-src.jar» a «lib»
        2. Seleccionar «jssc.jar» i botó dret «Build Path/Add to Build Path»

           Subversion «Team/Commit...»: «jssc instal·lació»
    2. Configuració
        1. Botó dret en «Referenced Libraries» sobre «jssc.jar» «Build Path/Configure Build Path...»
        2. «Edit» «Source attachment»

           «Workspace location»: «jssc-2.8.0-src.jar»
        3. «Edit» «Javadoc location»

           «Javadoc in archive/Workspace file»: «jssc-2.8.0-javadoc.jar»

           «Path within archive»: «javadoc»
        4. «Validate»

           Subversion «Team/Commit...»: «jssc configuració»
    3. Sandbox
        1. «New/Class» en subpaquet

           «edu.uoc.interfaceman.test.sandbox.serial»

           «Serial2.java»
        2. «Run»
        3. «New/Class» en subpaquet

           «edu.uoc.interfaceman.test.sandbox.serial»

           «Serial3.java»
        4. «Run»

           Subversion «Team/Commit...»: «jssc sandbox»

4. Subversion Tag (*)

   «Tag»: «v.0.3»

   «Comment»: «V.0.3»

Empaquetament de l’entorn

Per a evitar problemes amb els paths absoluts, descomprimir a «C:\».

Paths absoluts

Afectats al moure el directori base:

1. Workspace
    1. Desplegar «Recent Workspaces»
    2. Botó dret sobre el workspace «Remove from launcher selection»
    3. «Browse» i seleccionar el nou directori del workspace
2. Subversion
    1. «Window/Perspective/Open Perspective/SVN Repository Exploring»
    2. «SVN Repositories»
    3. Botó dret sobre repositori «Location Properties...»
    4. «Browse» i seleccionar el nou directori del repositori
    5. «Finish» i acceptar avís
3. PlantUML
    1. «Window/Preferences/PlantUML»
    2. Indicar el path al executable «dot.exe»: «...\graphviz\bin\dot.exe»

Tests per verificar el correcte funcionament:

1. Workspace
    1. Desplegar l'arbre del projecte
2. Compiladors
    1. CUP
        1. «Run As» sobre «build.xml» «edu/uoc/interfaceman/test/sandbox/cup/example»
        2. «Run» sobre «Driver.java»
3. Subversion
    1. «Window/Perspective/Open Perspective/SVN Repository Exploring»
    2. «SVN Repositories»
    3. Botó dret sobre repositori «Show History»
4. PlantUML
    1. Obrir «StateMachine.java»
    2. «Window/Show View/Other.../PlantUML/PlantUML»
5. Sandbox («Run»)
    1. «Logging.java»
    2. «StateMachine.java»
    3. «Serial.java»
    4. «Serial2.java»

Enllaços de descàrregues

Pàgines web de descàrregues i enllaços directes a la descàrrega:

- Java
    - http://www.oracle.com/technetwork/java/javase/downloads/index.html

    - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
    - http://download.oracle.com/otn-pub/java/jdk/8u121-b13/e9e7ea248e2c4826b92b3f075a80e441/jdk-8u121-windows-x64.exe
- Eclipse
    - http://www.eclipse.org/downloads/eclipse-packages/
    - http://www.eclipse.org/downloads/packages/eclipse-ide-java-and-dsl-developers/neon3
    - http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/3/eclipse-dsl-neon-3-win32-x86_64.zip&mirror_id=1190
- Compilador
    - JFlex/CUP
        - http://www2.cs.tum.edu/projects/cup/eclipse.php
        - https://sourceforge.net/projects/cup-lex-eclipse/files/
- Màquina d'estats
    - http://www.statefulj.org/fsm/
        - http://mvnrepository.com/artifact/org.statefulj/statefulj-fsm/3.0
        - http://central.maven.org/maven2/org/statefulj/statefulj-fsm/3.0/statefulj-fsm-3.0.jar
        - http://mvnrepository.com/artifact/org.statefulj/statefulj-common/3.0
        - http://central.maven.org/maven2/org/statefulj/statefulj-common/3.0/statefulj-common-3.0.jar
        - https://logging.apache.org/log4j/2.x/log4j-slf4j-impl/index.html
        - https://www.slf4j.org/download.html
        - https://www.slf4j.org/dist/slf4j-1.7.25.zip
    - Java Finite State Machine Framework
        - http://unimod.sourceforge.net/tools-packages.html
        - http://unimod.sourceforge.net/quickstart.html
        - http://unimod.sourceforge.net/eclipse-plugin.html
        - http://unimod.sourceforge.net/download.html
        - https://sourceforge.net/projects/unimod/files/
        - https://sourceforge.net/projects/unimod/files/UniMod/1.3.39.1/com.evelopers.unimod-1.3.39.1.zip/download#
        - https://sourceforge.net/projects/unimod/files/UniMod/01.02.015/UniMod-Bundle-01.02.015.zip/download#
        - http://stackoverflow.com/questions/10772443/which-java-state-machine-framework-has-a-functional-visual-eclipse-3-7-editor
- Comunicació sèrie
    - http://stackoverflow.com/questions/12317576/stable-alternative-to-rxtx
    - jSerialComm
        - http://fazecast.github.io/jSerialComm/

        - http://fazecast.github.io/jSerialComm/binaries/jSerialComm-1.3.11.jar
    - java-simple-serial-connector
        - https://code.google.com/archive/p/java-simple-serial-connector/
        - https://code.google.com/archive/p/java-simple-serial-connector/wikis/jSSC_Start_Working.wiki
        - https://code.google.com/archive/p/java-simple-serial-connector/wikis/jSSC_examples.wiki
        - https://github.com/scream3r/java-simple-serial-connector/releases
        - https://github.com/scream3r/java-simple-serial-connector/releases/download/v2.8.0/jSSC-2.8.0-Release.zip
- Subversion
    - http://www.eclipse.org/subversive/downloads.php
    - https://github.com/subclipse/subclipse/wiki
- PlantUML
    - http://plantuml.com/download
    - http://plantuml.com/eclipse
    - http://plantuml.com/graphviz-dot
    - http://graphviz.org/Download_windows.php
    - http://graphviz.org/pub/graphviz/stable/windows/graphviz-2.38.zip
- Log4j
    - https://logging.apache.org/log4j/2.x/download.html
    - http://apache.rediris.es/logging/log4j/2.8.2/apache-log4j-2.8.2-bin.zip

Apèndixs de codi

Logging.java

```
  package edu.uoc.interfaceman.test.sandbox;

  import org.apache.logging.log4j.LogManager;
  import org.apache.logging.log4j.Logger;

  // http://javabeat.net/log4j-2-example/
  // https://logging.apache.org/log4j/2.x/manual/configuration.html
  public class Logging {

      static          final          Logger          logger          =
LogManager.getLogger(Logging.class.getSimpleName());

      public static void main(String[] args) {
            logger.info("info log");
            logger.trace("trace log");
            logger.error(new Exception("Excepció"));
      }
  }
```

StateMachine.java

```
  package edu.uoc.interfaceman.test.sandbox;

  import java.util.LinkedList;
  import java.util.List;

  import org.statefulj.fsm.FSM;
  import org.statefulj.fsm.RetryException;
  import org.statefulj.fsm.TooBusyException;
  import org.statefulj.fsm.model.Action;
  import org.statefulj.fsm.model.State;
  import org.statefulj.fsm.model.impl.StateImpl;
  import org.statefulj.persistence.memory.MemoryPersisterImpl;

  // https://logging.apache.org/log4j/2.0/log4j-slf4j-impl/index.html
  //      http://whoopdicity.blogspot.com.es/2014/04/configuring-slf4j-
with-log4j2-using.html
  public class StateMachine {

      // Events
      //
      static String eventA = "Event A";
      static String eventB = "Event B";

      // States
      //
      static State<Foo> stateA = new StateImpl<Foo>("State A");
      static State<Foo> stateB = new StateImpl<Foo>("State B");
      static  State<Foo>  stateC  =  new  StateImpl<Foo>("State  C",
true); // End State

      // Actions
      //
      static Action<Foo> actionA = new HelloAction<Foo>("World");
      static Action<Foo> actionB = new HelloAction<Foo>("Folks");

      public static void main(String[] args) throws TooBusyException {
            /* Deterministic Transitions */

            // stateA(eventA) -> stateB/actionA
            //
            stateA.addTransition(eventA, stateB, actionA);

            // stateB(eventB) -> stateC/actionB
            //
            stateB.addTransition(eventB, stateC, actionB);

            // In-Memory Persister
            //
            List<State<Foo>> states = new LinkedList<State<Foo>>();
            states.add(stateA);
            states.add(stateB);
            states.add(stateC);

            MemoryPersisterImpl<Foo> persister =
                  new MemoryPersisterImpl<Foo>(
                        states,   // Set of States
                        stateA);  // Start State

            // FSM
```

```
            //
            FSM<Foo> fsm = new FSM<Foo>("Foo FSM", persister);

            // Instantiate the Stateful Entity
            //
            Foo foo = new Foo();

            // Drive the FSM with a series of events: eventA, eventA,
eventA
            //
            fsm.onEvent(foo,    eventA);       //    stateA(EventA)    ->
stateB/actionA

            foo.setBar(true);

            fsm.onEvent(foo, eventA);  // stateB(EventA) -> stateB/NOOP

            foo.setBar(false);

            fsm.onEvent(foo, eventA);  // stateB(EventA) -> stateC/NOOP
      }
  }


  //Stateful Entity
  //
  class Foo {

      @org.statefulj.persistence.annotations.State
      String state;   // Memory Persister requires a String

      boolean bar;

      public String getState() {
            return state;
      }

      // Note: there is no setter for the state field
      //       as the value is set by StatefulJ

      public void setBar(boolean bar) {
            this.bar = bar;
      }

      public boolean isBar() {
            return bar;
      }

  }


  //Hello <what> Action
  //
  class HelloAction<T> implements Action<T> {

      String what;

      public HelloAction(String what) {
            this.what = what;
      }
```

```
      public void execute(T stateful,
                          String event,
                          Object ... args) throws RetryException {
            System.out.println("Hello " + what);
      }
  }
```

Serial.java

```
  package edu.uoc.interfaceman.test.sandbox.serial;

  import com.fazecast.jSerialComm.SerialPort;

  public class Serial {

      private static final int TIMEOUT = 5_000;

      public static void main(String[] args) {
            SerialPort[] l = SerialPort.getCommPorts();
            for (SerialPort sp : l)
                  System.out.println(sp.getDescriptivePortName());
            p();

            SerialPort sp3 = SerialPort.getCommPort("COM3");
            p(sp3.openPort());
            boolean open = sp3.isOpen();
            p(open);
            if (open) {
                  SerialPort sp4 = SerialPort.getCommPort("COM4");
                  p(sp4.openPort());
                  open = sp4.isOpen();
                  p(open);
                  if (open) {
                        String buffer = "A";
                        p(buffer);
                        p(SerialPortDecorator.writeString(sp3,
buffer));
                        p();


      sp4.setComPortTimeouts(SerialPort.TIMEOUT_READ_SEMI_BLOCKING,
TIMEOUT, 0);
                        buffer   =   SerialPortDecorator.readString(sp4,
1);
                        if (buffer != null)
                              p(buffer);
                        p(sp4.closePort());
                  }
                  p(sp3.closePort());
            }
      }

      private static void p(Object o) {
            System.out.println(o);
      }

      private static void p() {
            System.out.println();
      }
  }
```

```
  class SerialPortDecorator {
      public static String readString(SerialPort sp, int bytesToRead)
{
            byte[] buffer = new byte[bytesToRead];
            int len = sp.readBytes(buffer, bytesToRead);
            String string = (len<0) ? null : new String(buffer, 0,
len);
            return string;
      }

      public static int writeString(SerialPort sp, String string) {
            byte[] buffer = string.getBytes();
            int len = sp.writeBytes(buffer, buffer.length);
            return len;
      }
  }
```

Serial2.java

```
  package edu.uoc.interfaceman.test.sandbox.serial;

  import jssc.SerialPort;
  import jssc.SerialPortException;
  import jssc.SerialPortList;
  import jssc.SerialPortTimeoutException;

  public class Serial2 {

      private static final int TIMEOUT = 5_000;

      public static void main(String[] args) {
            String[] portNames = SerialPortList.getPortNames();
            for(int i = 0; i < portNames.length; i++)
                  System.out.println(portNames[i]);
            p();

            boolean open3 = false,
                        open4 = false;
            SerialPort  sp3 = new SerialPort("COM3"),
                              sp4 = new SerialPort("COM4");;
            try {
                  p(sp3.openPort());
                  open3 = sp3.isOpened();
                  p("Port opened: " + open3);
                  if (open3) {
                        p(sp4.openPort());
                        open4 = sp4.isOpened();
                        p("Port opened: " + open4);
                        if (open4) {
                              p("Params setted: " + sp3.setParams(9600,
8, 1, SerialPort.PARITY_NONE));
                              p("Params setted: " + sp4.setParams(9600,
8, 1, SerialPort.PARITY_NONE));
                              p();

                              String buffer = "A";
                              p(buffer);
                              p("Successfully writen to port: " +
sp3.writeString(buffer));
                              p();
```

```
                        buffer = sp4.readString(1, TIMEOUT);
                        if (buffer != null)
                              p(buffer);
                  }
            }
      } catch (SerialPortTimeoutException e) {
            p(e);
      } catch (SerialPortException e) {
            p(e);
      } finally {
            if (open3)
                  try {
                        p("Port closed: " + sp3.closePort());
                  } catch (SerialPortException e) {
                        p(e);
                  }
            if (open4)
                  try {
                        p("Port closed: " + sp4.closePort());
                  } catch (SerialPortException e) {
                        p(e);
                  }
      }
  }

  private static void p(Object o) {
        System.out.println(o);
  }

  private static void p() {
        System.out.println();
  }
}
```

Serial3.java

```
package edu.uoc.interfaceman.test.sandbox.serial;

import java.util.Arrays;

import jssc.SerialPort;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;

public class Serial3 {

    static SerialPort serialPort;

    public static void main(String[] args) {
          serialPort = new SerialPort("COM3");
          try {
                serialPort.openPort();// Open port
                serialPort.setParams(9600, 8, 1, 0);// Set params
                int     mask     =     SerialPort.MASK_RXCHAR     +
SerialPort.MASK_CTS + SerialPort.MASK_DSR;// Prepare mask
                serialPort.setEventsMask(mask);// Set mask
                serialPort.addEventListener(new
SerialPortReader());// Add SerialPortEventListener
          } catch (SerialPortException ex) {
```

```
                System.out.println(ex);
            }
        }

        /*
         * In this class must implement the method serialEvent, through
it we learn about
         * events that happened to our port. But we will not report on
all events but only
         * those that we put in the mask. In this case the arrival of
the data and change the
         * status lines CTS and DSR
         */
        static class SerialPortReader implements SerialPortEventListener
{

            public void serialEvent(SerialPortEvent event) {
                if (event.isRXCHAR()) {// If data is available
                    if  (event.getEventValue()  >=  10)  {//  Check
bytes count in the input buffer
                        // Read data, if 10 bytes available
                        try {
                            byte           buffer[]           =
serialPort.readBytes(10);
                            System.out.println(buffer.length);

        System.out.println(Arrays.toString(buffer));
                        } catch (SerialPortException ex) {
                            System.out.println(ex);
                        }
                    }
                } else if (event.isCTS()) {// If CTS line has changed
state
                    if (event.getEventValue() == 1) {// If line is
ON
                        System.out.println("CTS - ON");
                    } else {
                        System.out.println("CTS - OFF");
                    }
                }  else  if  (event.isDSR())  {///  If  DSR  line  has
changed state
                    if (event.getEventValue() == 1) {// If line is
ON
                        System.out.println("DSR - ON");
                    } else {
                        System.out.println("DSR - OFF");
                    }
                }
            }
        }
    }
```

build.xml

```
<!-- https://ant.apache.org/manual/ -->

<project default="compile">

    <property name="lib"    location="lib" />
```

```
      <taskdef name="cup"
      classname="java_cup.anttask.CUPTask"     classpath="${lib}/java-
cup-11b.jar" />
      <taskdef name="jflex"   classname="JFlex.anttask.JFlexTask"
      classpath="${lib}/JFlex.jar" />

      <path id="libraries">
            <files includes="${lib}/java-cup-11b-runtime.jar" />
      </path>

      <target name="generate">
            <input    addproperty="_package"    message="Package    (amb
&amp;quot;/&amp;quot;):"/>
            <property name="source" location="src" />
            <property name="resources" location="resources/${_package}"
/>
            <cup  srcfile="${resources}/parser.cup"  destdir="${source}"
parser="Parser"            interface="true"            locations="true"
debugsymbols="true" />
            <jflex      file="${resources}/lexer.jflex"      destdir="$
{source}" />
      </target>

      <target name="compile" depends="generate">
            <property name="target" location="bin" />
            <javac     includeantruntime="false"     srcdir="${source}"
destdir="${target}">
                  <classpath refid="libraries" />
            </javac>
      </target>
  </project>
```