Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

DISPOSITIUS *HANDHELD* PER A APLICACIONS MÈDIQUES. DISSENY I IMPLEMENTACIÓ D'INTERFÍCIES DEL PROTOCOL START.

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Xavier Jurado Cristóbal i dirigit per Ramon Martí Escalé.

Bellaterra, juny de 2008

El firmant, Ramon Martí Escalé, professor del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha sigut realitzada sota la seva direcció per Xavier Jurado Cristóbal

Bellaterra, juny de 2008

Firmat: Ramon Martí Escalé

A tots els que m'heu recolzat durant aquests anys

Agraïments

Vull donar les gràcies a totes les persones amb qui he tingut el plaer de coincidir en aquests cinc anys de carrera. A tots els companys amb qui he compartit incomptables hores de feina i també de diversió. A l'Abraham, el Carlos, el Ferran i, especialment, al Gerard, per tots els moments viscuts.

Gràcies també als projectistes de SeNDA, al departament i en especial al meu director de projecte, Ramon Martí, per la seva paciència i els seus consells.

I gràcies a la meva família, que m'ha mostrat un suport incondicional durant tots aquests anys.

Índex

Capítol 1. Introducció **1**

1.1 Objectius **3**

1.2 Metodologia **3**

1.3 Estructura de la memòria **4**

Capítol 2. Estat de l'art **6**

2.1 Triatge **6**

2.1.1 Triatge simple 6

2.1.2 Etiquetes de triatge 7

2.1.3 START 7

2.2 GPS **8**

2.3 RFID **9**

2.3.1 Tipus d'etiquetes 9

2.4 Agents **10**

Capítol 3. Anàlisi **13**

3.1 Descripció de la proposta **13**

3.1.1 Requeriments funcionals 13

3.1.2 Requeriments no funcionals 14

3.2 Triatge **14**

3.2.1 L'algorisme START....14

3.2.2 Etiquetes de triatge.... 16

3.3 Components software **16**

3.3.1 Interfícies gràfiques per dispositius mòbils....16

3.3.2 Agents en aplicacions mèdiques.... 17

3.4 Components hardware **18**

3.4.1 Dispositiu mòbil: l'N810.... 18

3.4.2 Mòdul GPS....24

3.4.3 Mòdul RFID.... 25

3.5 Estudi de viabilitat **28**

3.5.1 Viabilitat tècnica....28

3.5.2 Viabilitat operativa....28

3.5.3 Viabilitat econòmica....28

3.5.4 Viabilitat legal....29

3.5.5 Planificació inicial de la proposta....29

Capítol 4. Disseny i implementació **31**

4.1 Disseny global de l'aplicació **31**

4.1.1 Components.... 31

4.1.2 Arquitectura....32

4.1.3 JADE.... 33

4.1.4 SWT....33

4.2 Agent interfície gràfica **34**

4.3 Agent d'emmagatzematge **35**

4.3.1 TriageDataAgent....35

4.3.2 TriageData.... 35

4.4 Controlador **38**

4.5 Model **39**

4.6 Vista general **40**

4.6.1 Layout....41

4.6.2 Menú....42

4.6.3 Controls específics.... 43

4.7 Vista de preferències **47**

4.7.1 Descripció....47

4.7.2 Disseny 47

4.8 Vista de triatge **48**

4.8.1 Descripció....48

4.8.2 Disseny 49

4.9 Vista START **49**

4.9.1 Descripció....49

4.9.2 Disseny 50

4.10 Vista d'edició manual **54**

4.10.1 Descripció....54

4.10.2 Disseny....55

4.11 Mòdul GPS **60**

4.11.1 Descripció....60

4.11.2 GPSController.java....61

4.11.3 GPSController.c.... 61

4.12 Mòdul RFID **62**

4.12.1 SerialBluetoothController....62

4.12.2 IDBlueController.... 63

Capítol 5. Proves i integració **66**

5.1 Proves **66**

5.1.1 Mòdul GPS 66

5.1.2 Mòdul IDBlue/RFID 67

5.1.3 Interfície gràfica 67

5.2 Test d'integració **68**

Capítol 6. Conclusions i línies d'ampliació **70**

Bibliografia **74**

Annex 1. Java **77**

Annex 2. Instal·lació i configuració de l'N810 **82**

Annex 3. Disseny d'icones per la interfície **86**

Capítol 1. Introducció

El 12 de maig del 2008, a la província de Sichuan de la República Popular de la Xina, un terratrèmol de 8.3 graus va acabar amb la vida de més de 60.000 persones i en va ferir 370.000 més.

Any rere any, casos com el de Sichuan s'han anat repetint; terratrèmols, tsunamis, inundacions o huracans han provocat situacions d'emergència on s'hi han vist involucrades un número desorbitant de persones.

És en aquests casos quan es despleguen mecanismes dissenyats específicament per situacions de pèrdues humanes massives. L'organització dels recursos disponibles durant els primers dies de l'incident és imprescindible per maximitzar el nombre de supervivents.

Mètodes de la medicina d'emergències com el triatge permeten a equips mèdics reduïts classificar les víctimes sobre el terreny segons la seva gravetat. D'aquesta manera, quan els primers equips d'assistència aconsegueixen arribar a la zona poden centrar-se en donar tractament a aquelles persones que més ho necessiten.

Al llarg dels anys han anat apareixent diferents mecanismes de triatge per una gran varietat de situacions. De tots ells, el protocol conegut amb el nom de START ha estat utilitzat amb èxit en situacions d'emergència reals. Aquest mètode es caracteritza principalment per ser capaç de diagnosticar l'estat d'una persona en un temps no superior als 60 segons.

Per tal de facilitar la feina als equips mèdics que arriben amb posterioritat a la zona, molts mètodes de triatge, inclòs START, lliga al voltant del coll o a les extremitats del pacient una etiqueta on, seguint un esquema de colors, s'informa de la seva necessitat d'atenció. A l'etiqueta, a més, s'hi emmagatzema informació breu sobre l'estat del pacient. D'aquesta manera es pot decidir, no sense implicacions morals, quins pacients seran atesos amb prioritat.

La tecnologia ha estat, generalment, absent en tots aquests escenaris. La inexistència d'infraestructures operatives no ha fet aconsellable la utilització de dispositius informàtics. Afortunadament, la industria aposta per desenvolupar sistemes cada vegada més petits i amb més autonomia; sistemes amb una potència i prestacions semblants a les d'un ordinador de sobretaula.

D'entre totes les famílies d'aparells, la dels dispositius mòbils (també coneguda amb el nom de *handheld devices*) s'ha centrat en oferir productes altament portables, d'una mida suficientment reduïda com per poder ser transportats en una butxaca. Se solen caracteritzar també per tenir una pantalla normalment tàctil de dimensions considerables (en relació amb la resta del cos).

Aquests aparells mòbils són especialment utilitzats en entorns empresarials, donada la capacitat d'integrar una gran quantitat d'informació en un espai tan reduït. Però creiem que també tenen qualitats adequades per ser utilitzats en les situacions d'emergència descrites.

El procés de triatge, per la seva simplicitat, pot ser realitzat per equips amb una mínima preparació mèdica, com poden ser bombers, policies o militars. El personal amb més coneixements generalment és destinat a l'atenció directa de les víctimes un cop són recollides de la zona del desastre.

Al mateix temps, però, el sistema no és suficientment intuïtiu com per ser utilitzat per persones que desconeguin què és el triatge. Si no es coneix l'algorisme a seguir per assignar un o altre estat a partir dels signes vitals del pacient, no es podrà ajudar activament en el procés.

Per altra banda, en tot procés de triatge és habitual trobar casos on no s'ha assignat l'estat més recomanable al pacient. Si es tria una prioritat major a la que el pacient necessita (*overtriage*), es corre el risc de saturar els pocs recursos mèdics disponibles amb persones que no requereixen atenció immediata. Per altra banda, assignar una prioritat menor a la real (*undertriage*), és a dir, infravalorar la necessitat d'atenció del pacient, pot portar a un empitjorament del seu estat i, ocasionalment, a la seva mort.

El sistema de triatge clàssic presenta també una limitació important alhora de donar a conèixer quines són les zones amb més gent afectada. Als equips d'assistència mèdica, capaços de recollir les víctimes i portar-les a hospitals, els hi pot interessar anar a zones ja analitzades on s'hi trobin els pacients més greus, però no coneixeran aquestes dades a priori perquè les etiquetes que contenen aquesta informació es troben lligades físicament a les víctimes.

Per totes aquestes raons, un sistema informatitzat que sigui capaç d'escollir l'estat del pacient de forma semiautomàtica i emmagatzemar-lo té la seva utilitat. La utilització de dispositius mòbils en aquest entorn no entorpeix la tasca habitual de triatge; la seva autonomia pot ser suficient per aguantar més d'un dia de funcionament, i poden oferir avantatges únics que justifiquin la seva implantació.

Aquest és el punt de partida del nostre projecte.

1.1 Objectius

- Disseny i implementació d'un assistent per al protocol de triatge START, que pugui ser utilitzat per personal sense coneixements de l'algorisme en sí, sense que això suposi un increment significatiu en el temps de triatge.
- Disseny i implementació d'una interfície interactiva que permeti la introducció de les dades de forma manual, per tot aquella gent que no desitgi utilitzar l'assistent.
- Creació d'una estructura capaç d'emmagatzemar totes les dades recollides de cada pacient, és a dir, una versió virtual de les etiquetes de triatge tradicionals.
- Juntament amb l'estat mèdic, les possibles dades personals (si es coneixen) o d'altres anotacions que es creguin convenients, l'aplicació haurà d'anotar la localització del pacient a través d'un dispositiu GPS.
- El sistema també haurà de ser capaç de llegir un RFID col·locat al pacient i d'emmagatzemar-lo juntament amb totes les dades recollides, per tal de facilitar les tasques d'identificació.
- Integració de tot el sistema desenvolupat en dispositius Nokia N810, tenint en compte les limitacions i necessitats específiques de plataformes mòbils, molt especialment les relacionades amb la usabilitat d'interfícies gràfiques.
- Integració del sistema en un entorn d'agents.

1.2 Metodologia

Per acomplir els objectius marcats, seguirem els següents passos:

1. Primer estudiarem les característiques del Nokia N810. De forma general, haurem de descobrir què suposa dissenyar i implementar aplicacions per a dispositius mòbils. Però també haurem d'analitzar les capacitats del nostre dispositiu, i trobar-ne les seves limitacions.
2. Tot seguit haurem d'estudiar el mètode START en detall, així com tota la terminologia bàsica relacionada amb la medicina d'emergències.

3. Un cop coneguem tota aquesta informació, estarem en disposició de realitzar un anàlisi dels requeriments funcionals i no funcionals del nostre sistema.
4. A continuació, haurem de dissenyar tots i cadascun dels elements en què hàgim decidit dividir el projecte.
5. Implementarem la totalitat del sistema utilitzant les eines que tinguem disponibles en el nostre dispositiu mòbil, intentant seguir patrons de disseny adequats a la nostra problemàtica.
6. Validarem la feina feta comprovant que tots els mòduls del sistema compleixen els requeriments exposats durant la fase d'anàlisi.
7. Finalment, valorarem la feina feta, determinarem les seves limitacions i establirem un conjunt de línies futures de desenvolupament.

1.3 Estructura de la memòria

Tenint en compte la metodologia utilitzada, hem estructurat la memòria de la següent forma:

- **Capítol 2: Estat de l'art**. En aquest capítol donarem coneixements necessaris per poder entendre la resta de la memòria. Parlarem del triatge i el mètode START, de la tecnologia GPS i RFID i del concepte d'agent i plataforma.
- **Capítol 3: Anàlisi**. Farem un anàlisi detallat dels requeriments funcionals i no funcionals del projecte. Mostrarem els principals mòduls en què es dividirà el projecte i realitzarem un estudi de viabilitat, oferint la planificació inicial de la proposta.
- **Capítol 4: Disseny i implementació**. En aquest capítol mostrarem els detalls del disseny i implementació de cadascun dels mòduls descrits en l'anterior capítol.
- **Capítol 5: Proves i integració**. Tal com el seu nom indica, en aquest capítol es realitzaran proves del sistema per comprovar que compleix amb els requisits exposats. Es provaran, per una banda, els diferents components de forma individual, per acabar realitzant un test d'integració de tot el sistema.
- **Capítol 6: Conclusions i línies d'ampliació**. Amb els resultats a la mà, veurem si hem acomplert els objectius del projecte plantejats en aquest capítol, i veurem línies d'investigació futures.
- **Annexes.** Hi mostrarem informació addicional relacionada amb el projecte, agrupada de forma temàtica.

Capítol 2. Estat de l'art

Al llarg d'aquest capítol presentarem alguns conceptes necessaris per entendre el projecte: què són i per què s'utilitzen els mètode de triatge, característiques del protocol START, la tecnologia GPS i RFID i el paradigma dels agents.

2.1 Triatge

El triatge és un mètode de la medicina d'emergències i desastres per la selecció i classificació dels pacients, basat en el seu estat així com en els recursos mèdics disponibles. Ideat pel baró Dominique-Jean Larrey (1766-1842), metge cirurgià francès sota les ordres de Napoleó, el triatge té com a objectiu primordial maximitzar el nombre de vides salvades.

Per aconseguir-ho, assigna a cada pacient un nivell de prioritat (mort, immediat, endarrerit o menor) d'acord amb les seva necessitat d'atenció. Així, un nivell que indiqui que el pacient pot ser demorat no vol dir que el seu diagnòstic final no sigui greu, sinó que a mesura que passin les hores el seu estat no variarà sensiblement.

Un cop classificats, els pacients seran atesos per les unitats mèdiques d'acord el nivell assignat durant el triatge.

2.1.1 Triatge simple

El triatge en situacions de pèrdues humanes massives on els pacients han de romandre a la zona afectada (l'anomenada **zona d'emergència**) durant llargs períodes de temps, com per exemple després d'un terratrèmol de magnitud elevada, difereix en gran mesura dels mètodes de triatge tradicional.

Les zones amb víctimes són molt àmplies, i les infraestructures existents no estan operatives. Els recursos mèdics disponibles són limitats, l'evacuació no és possible i els equips d'emergència de la zona no solen rebre ajuda exterior de manera immediata.

De tots els mètodes de triatge existents, el triatge simple és el dissenyat específicament per aquestes situacions. Es tracta, per tant, d'un procés que es realitza en la mateixa zona on es troben les víctimes. El seu objectiu és identificar aquells pacients que necessiten atenció prioritària i/o transport immediat cap a l'hospital de campanya més proper.

2.1.2 Etiquetes de triatge

Al finalitzar el procés de triatge, una etiqueta coneguda amb el nom *triage tag* és col·locada en el cos del pacient on s'indica l'estat assignat. Per tal de poder identificar a simple vista quin és l'estat d'una persona, les etiquetes acostumen a seguir un esquema de colors:

- Negre (mort): quan el pacient ha mort o no és possible fer res per salvar-li la vida
- Vermell (immediat): quan necessita atenció immediata
- Groc (endarrerit): quan, malgrat les ferides, la seva atenció pot ser retardada
- Verd (menor): quan té ferides lleus

A més de l'esquema de colors, a l'etiqueta de triatge se solen anotar dades com els signes vitals detectats, el tractament preventiu aplicat, les dades personals del pacient (si són conegudes) i un esquema del cos amb les ferides detectades.

No totes les dades són complimentades al mateix temps i pel mateix personal; s'ha de veure l'etiqueta com un petit informe mèdic d'urgència que es va ampliant a mesura que el pacient rep l'atenció necessària.

2.1.3 START

START (*Simple Triage And Rapid Treatment*) és una implementació del mètode de triatge simple orientat a personal amb poca experiència mèdica (voluntaris, personal d'emergència, bombers, policies, etc). Aquest és un fet determinant ja que permet l'aprofitament d'una bona part dels recursos humans disponibles.

Va ser desenvolupat l'any 1983 per membres de l'hospital *Hoah* en col·laboració amb el cos de bombers de *Newport Beach* (Califòrnia, EUA), i actualitzat l'any 1996 pel centre mèdic *Eisenhower* [1]. En l'actualitat és un dels pocs mètodes de triatge que ha demostrat la seva eficàcia en situacions reals d'emergència [2].

El protocol START recomana que, en una primera instància, el personal faci una crida per tal que totes les víctimes que es vegin amb cor de moure's es desplacin cap a una zona determinada, o bé ajudin en les tasques de triatge. La gent que respon favorablement a la crida és classificada amb etiquetes de color verd, és a dir, el menor grau de gravetat.

A continuació, s'avaluen les funcions respiratòries, circulatòries i neuronals de les víctimes restants. D'acord amb els resultats d'aquestes proves (que solen durar, en conjunt, menys de 60 segons), els pacients són classificats en una de les tres categories restants. Més endavant veurem en detall l'algorisme de decisió utilitzat pel protocol.

2.2 GPS

El Sistema de posicionament global, conegut amb el nom de GPS, és l'únic sistema de navegació per satèl·lit funcional a dia d'avui. Utilitzant una constel·lació de més de 24 satèl·lits que orbiten a 20.200 quilòmetres de la Terra, el sistema permet a qualsevol receptor GPS conèixer la seva situació, velocitat, direcció i hora. El sistema va ser desenvolupat i instal·lat, i actualment és operat, pel departament de Defensa dels Estats Units.

Per tal de calcular aquestes dades, el receptor GPS ha de localitzar les emissions de com a mínim amb 4 satèl·lits de la xarxa, dels quals rep uns senyals de navegació [3] indicant:

- L'hora i el dia del sistema (obtingut del rellotge atòmic que tenen abord), així com l'estat de funcionament del satèl·lit.
- Les efemèrides, informació orbital que permet al receptor calcular la posició a l'espai del satèl·lit; tenen una validesa de no més de quatre hores.
- Les dades conegudes amb el nom d'almanac, formades per l'estat, l'òrbita i codis d'identificació de tots els satèl·lits GPS; al contrari que les efemèrides, aquestes dades són vàlides durant uns 180 dies.

Sobre aquesta base, l'aparell sincronitza amb el seu rellotge intern i calcula el retard de les senyals, és a dir, la distància a cada satèl·lit. Conegudes les distàncies, es determina fàcilment la pròpia posició relativa respecte als satèl·lits. Coneixent a més les coordenades de cadascun d'ells, s'obtenen les coordenades reals del punt de mesurament. A la pràctica també s'utilitzen altres dades per tal d'obtenir resultats més precisos.

La fiabilitat depèn principalment del nombre de satèl·lits descoberts i de l'existència d'efectes atmosfèrics adversos que afectin a la velocitat de transmissió del senyal. Els GPS d'ús comú tenen un error de precisió de 15 metres, i si ens posem en contacte amb 9 satèl·lits enlloc dels 4 necessaris poden aconseguir precisions inferiors als 2,5 metres.

2.3 RFID

La identificació per radiofreqüència o RFID (de l'anglès *Radio Frequency Identification*) és un sistema automàtic d'identificació d'objectes basat en l'emmagatzematge i recuperació remot de dades utilitzant les anomenades etiquetes o *tags* RFID.

El mode de funcionament dels sistemes RFID és simple. L'etiqueta RFID, adherida a un objecte, transmet un identificador únic utilitzant un senyal de radiofreqüència. El senyal pot ser captat a distància per un lector RFID, el qual s'encarrega de llegir i transmetre aquesta informació a l'aplicació específica que utilitza RFID.

Les etiquetes estan formades per dos components. Per una banda, un circuit integrat per emmagatzemar i processar informació, així com per modular el senyal; conté un número d'identificació únic, i opcionalment, una memòria no volàtil (EEPROM). Per altra banda, l'etiqueta disposa d'una antena per rebre i transmetre els senyals.

2.3.1 Tipus d'etiquetes

Segons les necessitats d'alimentació elèctrica, podem distingir tres famílies d'etiquetes RFID: actives, semi-passives o passives.

Etiquetes passives

Els *tags* passius són aquells que no necessiten alimentació interna. Són capaços d'alimentar el circuit integrat a partir del corrent elèctric induït a l'antena per les senyals de ràdio freqüència que emet el lector RFID.

Es poden llegir a distàncies que varien des d'uns 10 mm fins a 6 m [4], depenent del disseny i la mida de l'antena, així com de la potència del lector. Les transmissions, però, duren el temps just per transmetre el número d'identificació.

Degut al seu disseny simple es poden crear a través d'un procés d'impressió que se sol integrar en un adhesiu.

Etiquetes actives

A diferència de les etiquetes passives, les actives tenen la seva pròpia font d'alimentació autònoma, que utilitzen per donar corrent als circuits integrats i propagar la seva senyal cap al lector. Conseqüentment, aquests *tags* són molt més fiables que els passius i poden ser llegits a majors distàncies (centenars de metres), però la seva mida el seu preu són també molt superiors.

Alguns d'ells integren sensors (de temperatura, humitat, llum, etc) per observar l'entorn al qual es troben adherit.

Etiquetes semi-passives

Aquesta família d'etiquetes incorporen també una font d'alimentació pròpia, però només la utilitzen per alimentar el xip i no per transmetre el senyal. De cara al lector, doncs, es comporta com una etiqueta passiva.

El xip intern, gràcies a la seva alimentació continuada, pot mantenir una màquina d'estats funcionant. Aquest fet sol aprofitar-se per oferir respostes basades en l'estat anterior de l'etiqueta i les dades transmeses pel lector.

Les etiquetes semi-passives ofereixen una fiabilitat semblant a les etiquetes actives i un temps de vida superior.

2.4 Agents

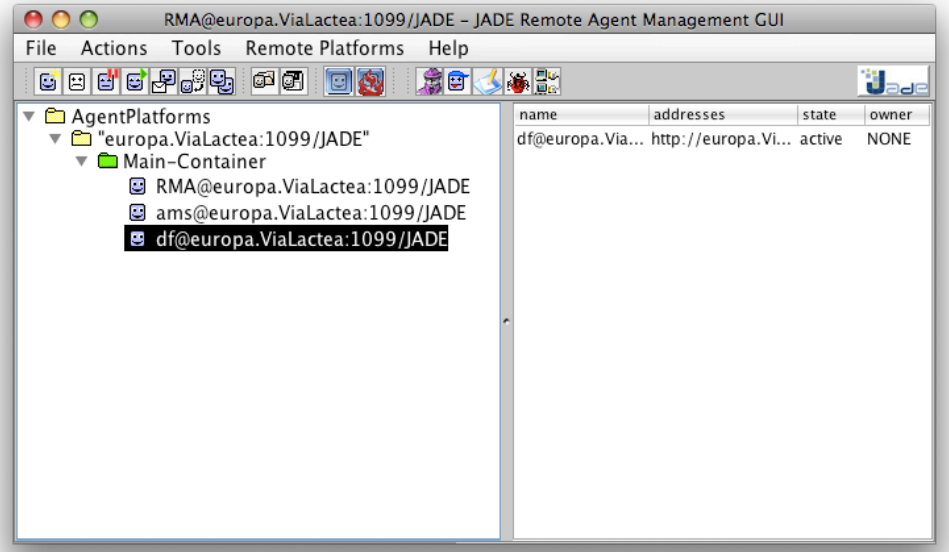
Un agent és una abstracció *software*, una idea, de la mateixa manera que "objecte" és un concepte abstracte de la programació orientada a objectes. El terme agent engloba una entitat complexa de *software* capaç d'actuar amb un cert grau d'autonomia i acomplir tasques en nom d'un usuari o un altre programa.

La majoria d'autors coincideixen en els següents aspectes alhora de definir el concepte d'agent [5]:

- Persistència: el codi no és executat sota demanda, sinó que ho fa de forma continuada i decideix per si mateix quan ha de realitzar les seves activitats.
- Autonomia: els agents tenen la capacitat de seleccionar, prioritzar i coordinar les seves tasques sense intervenció humana.
- Habilitat social: els agents són capaços d'iniciar altres components utilitzant mecanismes de comunicació, i poden ser capaços de cooperar en una acció determinada.
- Reactivitat: els agents són conscients de l'entorn on s'estan executant i tenen la capacitat d'adaptar-s'hi.

Així, a diferència dels objectes que són definits en termes de mètodes i atributs, un agent es defineix en termes del seu **comportament**. Per tal d'executar-se, els agents necessiten residir en un entorn anomenat **plataforma** o **agència**, que disposa d'eines bàsiques per tal de permetre'n la seva existència i operativitat.

Una de les plataformes desenvolupades sota el paradigma dels agents és JADE (*Java Agent DEvelopment Framework*), un *framework* totalment implementat en Java que proporciona eines per l'execució, migració i comunicació entre agents.



JADE. Interfície gràfica (agent RMA) de la plataforma.

Capítol 3. Anàlisi

Al llarg d'aquest capítol donarem una visió general del projecte i dels seus requeriments funcionals i no funcionals. Per una banda parlarem de l'estructura del projecte i, per l'altra, analitzarem de forma individual cadascun dels mòduls en què hem dividit el nostre treball i n'exposarem, si s'escau, els seus requeriments específics. Finalment, veurem també un estudi de la viabilitat del projecte.

3.1 Descripció de la proposta

Volem crear un sistema que implementi el protocol START en un dispositiu portàtil, que inclogui a més la localització GPS del pacient i un identificador únic en forma de RFID.

Les dades recollides durant el procés les emmagatzemarem en un agent, que romandrà en el dispositiu portàtil per poder ser processat a *posteriori* (ja que no és objectiu del nostre projecte entrar en aquesta fase).

3.1.1 Requeriments funcionals

Sense entrar en detalls, aquests són els requeriments funcionals que demanem al nostre projecte:

- S'haurà de crear una interfície gràfica que permeti als usuaris dels equips d'emergència completar un procés de triatge seguint el mètode START sense necessitat de conèixer el procediment en si.
- També haurà de permetre completar un procés de triatge sense l'ajuda del protocol START (seleccionant l'estat del pacient de forma manual).
- En qualsevol dels casos anteriors, s'hauran de poder complimentar les dades recollides amb d'altres que el personal mèdic consideri rellevants.
- El sistema ha de ser capaç de recuperar la posició GPS actual i incloure-la amb les dades del pacient.

- També ha de poder llegir el número d'identificació d'etiquetes RFID adherides als pacients.
- Per cada pacient classificat, s'haurà de crear un agent amb tota la informació recollida.
- S'haurà de poder cancel·lar el procés de triatge que s'està duent a terme.
- Serà necessari que el personal mèdic s'identifiqui abans de poder utilitzar l'aplicació, i que aquesta identificació s'inclogui en totes les dades recollides.
- Es podran personalitzar alguns aspectes del comportament de l'aplicació, com l'idioma de la interfície gràfica o el mètode de triatge a utilitzar.

3.1.2 Requeriments no funcionals

- El sistema serà utilitzat pel personal d'emergències: se suposen uns coneixements mínims sobre el triatge, així que el temps d'aprenentatge de l'aplicació haurà de ser de pocs minuts.
- S'hauran de prendre decisions de forma automàtica (sense la intervenció de l'usuari) sempre que sigui possible. La tasca de l'usuari haurà de centrar-se en el procés de triatge.
- L'aplicació encarregada de realitzar el procés START haurà de ser accessible de forma tàctil amb els dits. La resta de la interfície serà dissenyada també amb aquest objectiu, però no serà obligatori que el compleixi.
- El triatge d'un pacient seguint el mètode START s'ha de poder realitzar en menys de 60 segons.
- S'hauran de minimitzar els recursos utilitzats pel sistema, donat que haurem de treballar en aparells portàtils amb molta menys potència que un ordinador de sobretaula.
- El sistema s'ha de poder utilitzar fins i tot quan no existeixin dades del GPS ni de l'etiqueta RFID.
- Haurem de treballar amb dispositius mòbils Nokia N810, però la interfície gràfica haurà de ser portable a altres dispositius.
- El sistema s'haurà de poder integrar en una plataforma d'agents JADE.
- El lector d'RFID utilitzat serà l'IDBlue de la companyia Cathexis/Baracoda.

3.2 Triatge

3.2.1 L'algorisme START

Com hem comentat en l'anterior tema, el protocol START defineix que totes aquelles persones que puguin caminar i desplaçar-se cap a una zona determinada seran etiquetades amb el color verd, indicant que no necessiten atenció immediata.

L'estat de les persones que no han respòs a la crida dels membres de l'equip d'emergència és avaluat segons la resposta a una sèrie de proves. En una primera instància, es comproven les funcions respiratòries:

1. Si la víctima no respira, el personal neteja les vies respiratòries i aplica les maniobres necessàries per aconseguir que la víctima torni a respirar.
   1.1. Si els intents fracassen, es pressuposa que el pacient és **mort** i, per tant, és classificat amb l'etiqueta de color negre.
   1.2. Si s'aconsegueix reanimar la funció respiratòria del pacient, la víctima és etiquetada de color vermell indicant la seva necessitat d'**atenció immediata**.
2. Si la víctima respira, aleshores es mesura amb quina freqüència ho fa.
   2.1. Si aquest valor és superior a les 30 respiracions per minut, la víctima és marcada amb l'etiqueta d'**atenció immediata** de color vermell ja que aquest és un dels principals senyals de *shock*.
   2.2. Si la freqüència és inferior a 30 inspiracions (o expiracions) per minut, aleshores es passa a mesurar el pols i les funcions circulatòries.
      2.2.1. Si no es troba el pols, es mesure el seu *capillary refill*, és a dir, el temps que tarda la sang a emplenar vasos capil·lars buits. Aquesta valor es mesura calculant els segons que tarda la punta del dit a recuperar el color després de ser pressionat pel personal de triatge.
         2.2.1.1. Si el temps és superior als dos segons, la víctima necessita **atenció immediata** i és marcada amb l'etiqueta vermella.
         2.2.1.2. En cas contrari, es procedeix a avaluar l'estat mental de la víctima tal com expliquem en el següent punt.
      2.2.2. Si es troba el pols, aleshores es mesura l'estat mental del pacient.
         2.2.2.1. Si és capaç de seguir instruccions senzilles, el pacient és marcat amb l'etiqueta de color groc indicant que la seva **atenció pot ser aplaçada** unes hores.
         2.2.2.2. Si no és capaç de seguir-les, aleshores el vermell és el color de l'etiqueta escollida per indicar la necessitat d'**atenció immediata**.

La versió inicial del protocol només tenia en compte l'estat mental, la freqüència de respiració i el test capil·lar del pacient. Una de les modificacions plantejades l'any 1996 [6], inclosa en aquesta versió, va ser mesurar també el pols. És un canvi eficaç en situacions de baixes temperatures, on el resultat del test capil·lar no és concloent.

La nostra versió del protocol START haurà d'implementar l'algorisme de triatge exactament com l'hem descrit en aquest apartat. Assignarà l'estat del pacient de forma automàtica segons les dades entrades per l'usuari. No hi haurà lloc per les situacions d'*undertriage* o *overtriage* descrites en la introducció.

3.2.2 Etiquetes de triatge

Ja que no existeix cap normativa al respecte, els camps i la forma de les etiquetes de triatge clàssiques varien de fabricant en fabricant. Així i tot, la majoria inclouen les següents seccions:

- Els quatre estats en els quals es pot classificar el pacient
- Informació sobre els signes vitals detectats
  - Pols, pressió sanguínia, estat mental, número d'inspiracions per minut, etc)
- Informació personal del pacient
  - Sexe, nom, adreça, etc
- Situació de les ferides del pacient (generalment dibuixades sobre un esquema)

A més a més, certs camps de les etiquetes estan preparades ser omplerts vàries vegades, indicant així l'evolució del pacient al llarg del temps.

Totes aquestes característiques les volem mantenir i millorar amb el nostre sistema. No només oferirem els camps més habituals descrits, sinó que a més els relacionarem amb la persona que els ha introduït al sistema. Serà possible entrar una mateixa dada tantes vegades com es vulgui, i tots els valors proporcionats seran guardats juntament amb l'hora en què van ser introduïts.

3.3 Components software

3.3.1 Interfícies gràfiques per dispositius mòbils

Usabilitat

Crear una interfície és un repte per qualsevol enginyer. Es tracta d'un dels molts components que formen un projecte, però és la principal característica que serà avaluada pels usuaris. La nostra no és una aplicació comercial, però la usabilitat serà igualment un fet determinant, ja què es traduirà directament en la velocitat del procés de triatge i, per tant, en el número de persones que podran ser ateses.

En primer lloc hem de tenir present que interactuar amb una interfície utilitzant els dits és molt diferent de fer-ho a través d'un *mouse*, o fins i tot a través d'un llapis. La interfície ha de continuar essent accessible, el que suposa que els controls hauran de tenir una certa mida per facilitar-ne la seva selecció. Per norma general, aquesta mida mínima és d'1cm^2 [7].

En segon lloc, hem de recordar que estem treballant amb una pantalla de 800 per 480 píxels, menor a la d'un ordinador personal. És un error comú creure que reduir la mida de la interfície és la millor solució. Tot al contrari, hem de **sintetitzar** el que volem mostrar a l'usuari per pantalla i, com hem comentat, augmentar la seva mida perquè sigui accessible.

Finalment, hem de tenir present que l'N810 no disposa de cap sistema de reconeixement d'escriptura o de veu. L'única forma d'escriure és utilitzant el teclat *QWERTY* de l'aparell, però és una feina feixuga degut a la mida de les tecles. Per tant, hem d'intentar reduir l'entrada de text.

Portabilitat

Ja que el nostre sistema s'ha d'integrar en una plataforma d'agents JADE, desenvolupada completament en **Java**, creiem que és una bona idea realitzar la interfície gràfica en aquest mateix llenguatge.

Això ens ofereix els següents avantatges:

- Poder accedir de forma nadiua a JADE, sense la necessitat de cap *middleware*
- Poder desenvolupar i executar aquesta part del projecte en qualsevol sistema que suporti Java, no exclusivament en el dispositiu mòbil
- Produir un *software* portable de forma implícita
- Facilitar el procés de desenvolupament gràcies a les característiques del llenguatge (gestió de memòria automàtica, gran quantitat d'APIs a la nostra disposició, etc)

3.3.2 Agents en aplicacions mèdiques

Un dels requeriments no funcionals enumerats al començament d'aquest capítol ha estat la necessitat d'integrar el nostre sistema en una plataforma JADE. El nostre projecte podria ser dissenyat sense utilitzar agents, però hem volgut preparar-lo per poder funcionar en aquest entorn.

Revistes de referència en l'àrea com *IEEE: Intelligent Systems* han dedicat un número complet [8] a parlar de les aplicacions mèdiques que s'estan desenvolupant utilitzant la tecnologia d'agents. Els avantatges són clars: poder dotar els agents amb un cert grau d'intel·ligència permet prendre decisions en temps real que ajudin al diagnòstic o al tractament de pacients de forma automatitzada.

Quin paper tindran els agents en el nostre sistema? Per una banda, seran els encarregats d'emmagatzemar els resultats de tot el procés, és a dir, les dades resultants de diagnosticar cada pacient. Un cop creats, els agents romandran en la plataforma a l'espera de rebre instruccions. No implementarem cap comportament: aquest és un punt que deixem obert a línies futures d'investigació.

Per altra banda, la nostra interfície també estarà continguda dins d'un sol agent; d'aquesta manera, podrem accedir fàcilment a JADE per anar llençant els agents d'emmagatzematge necessaris.

3.4 Components hardware

3.4.1 Dispositiu mòbil: l'N810

Consideracions prèvies

Tal com detallaven a la introducció del capítol, un dels requeriments del nostre projecte és el d'utilitzar aparells Nokia N810 com a dispositius mòbils. Aquesta decisió ha estat presa pel grup SeNDA, a les ordres del qual hem desenvolupat aquest treball. De totes maneres, creiem convenient fer una descripció de les necessitats que hauria de complir qualsevol aparell per ser considerat una opció viable.

El nostre projecte planteja la possibilitat d'utilitzar dispositius de mà en aplicacions mèdiques, i busca demostrar la viabilitat d'aquesta idea implementant una interfície del protocol START pel triatge de pacients en situacions d'emergències mèdiques.

Volem, a més, millorar el procés de triatge incorporant una mesura de la posició GPS de cada pacient, i per fer-ho necessitem un receptor GPS. També necessitem un lector d'etiquetes RFID per poder identificar de forma única cada pacient, de cara a la nostra aplicació.

Els requisits pel nostre dispositiu mòbil es perfilen d'aquesta manera: ha de complir les qualitats necessàries per poder ser transportat i utilitzat fàcilment per una sola persona, incorporar un receptor de GPS i alguna interfície amb què interactuar amb un lector RFID. La pantalla ha de ser suficientment gran com per poder treballar-hi amb les mans, sense la necessitat d'utilitzar llapis, perquè aquest és un element que es podria perdre fàcilment en una situació real d'emergència.

A més a més, de cara a facilitar la nostra feina de desenvolupament, seria desitjable que utilitzés alguna distribució de Linux com a sistema operatiu, tant per l'aspecte monetari com per la gran quantitat de programari lliure que existeix per aquesta plataforma i que podríem utilitzar en el nostre benefici.

Descripció general de l'aparell

El **Nokia N810** ha estat el model escollit amb el qual haurem de realitzar la totalitat del nostre projecte. Compleix sobre el paper, com veurem a continuació, els requisits plantejats en l'anterior punt.

N810. Visió general del dispositiu amb el teclat desplegat.

L'N810 és un dispositiu de mà de Nokia per xarxes sense fils. Malgrat la relació entre la companyia i els terminals de telefonia mòbils, aquests aparell no és un telèfon. El dispositiu està dissenyat principalment per ser utilitzat com un terminal de connexió a Internet, i ho pot fer utilitzant la tarja *Wi-Fi* integrada o mitjançant una connexió Bluetooth amb un altre dispositiu.

El sistema operatiu de l'aparell s'anomena **Internet Tablet OS 2008** (OS2008), i està basat en la plataforma de desenvolupament **maemo 4.0** (que, al seu torn, està basada en Debian). Internament, és molt semblant a aquesta distribució de Linux.

Característiques tècniques

- Mida
  - 128 x 72 x 14 mm
  - 226 g
- Pantalla
  - 800 x 480 píxels a 65.536 colors (16 bits)
  - Tàctil
- Processador
  - Texas Instruments OMAP 2420 a 400Mhz
- Memòria
  - 128 MB RAM DDR
  - 256 MB sistema arxius
  - 2 GB de memòria *flash* interna (`/media/mmc2`)
  - Ranura d'expansió miniSD, compatible també amb microSD
- Energia
  - 4 hores ús continuat (navegació)
  - 14 dies en repòs
- *Hardware* integrat
  - GPS *chipset* Texas Instruments 5300
  - Micròfon i altaveus
  - Càmera VGA
  - Sensor de llum ambiental
  - WLAN (IEEE 802.11b/g) STLC4550
  - Bluetooth 2.0 EDR
  - Connector Micro USB 2.0 High Speed
  - Teclat *QWERTY*

maemo

maemo és una plataforma de desenvolupament construïda amb components de *software* lliure i basada en Debian, una de les distribucions de Linux més estables i conegudes. La plataforma va ser presentada l'any 2005 per Nokia, juntament amb el primer dispositiu que l'utilitzava, l'Internet Tablet 770. El nom comercial del sistema operatiu d'aquestes unitats es va anomenar Nokia Internet Tablet OS 2006.

Nokia, amb l'ajuda de la comunitat *Open Source*, ha continuat desenvolupant noves versions de maemo, i les ha anat incorporant en els dispositius N800 i N810, amb el nom d'Internet Tablet OS 2007 i 2008 respectivament.

Versió	*Codename*	Nokia OS	Dispositius compatibles
maemo 2.2	Gregale	OS2006	Nokia N770
maemo 3.X	Bora	OS2007	Nokia N800
maemo 4.0	Chinook	OS2008	Nokia N810 i N800 (actualitzable)
maemo 4.1	Diablo	N/A	N/A

Versions de la plataforma maemo. La versió 4.1 encara es troba en desenvolupament.

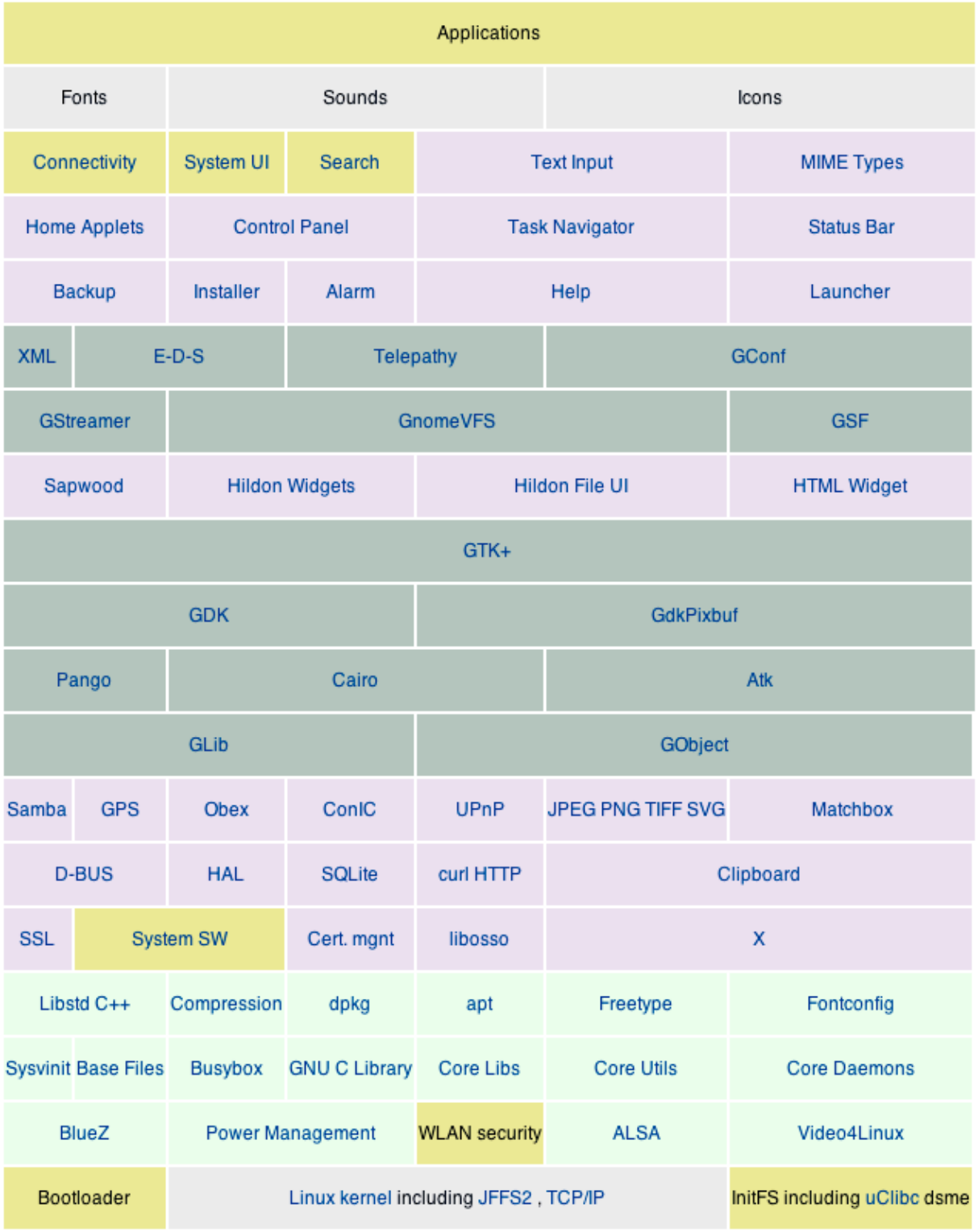
Arquitectura

El *kernel* de maemo és una versió de la sèrie 2.6 del *kernel* de Linux. Tots els dispositius sobre els quals corre maemo disposen d'un microprocessador de la sèrie OMAP de Texas Instrument, format per un processador ARM de propòsit general i un DSP (Processador Digital del Senyal).

El *software* a l'espai d'usuari *linka* amb la llibreria GNU C estàndard (i no amb la llibreria uClib d'altres dispositius mòbils). D'aquesta manera, maemo intenta disminuir el cost de portar aplicacions d'altres sistemes Linux.

El sistema de gestió de paquets, la jerarquia del sistema de fitxers i d'altres polítiques de disseny són molt semblants a les d'una distribució Debian. D'aquesta manera, per exemple, podem gestionar la instal·lació de *software* directament des del terminal amb la comanda `apt-get`.

L'entorn gràfic de maemo s'anomena Hildon, i es pot considerar una versió per dispositius portàtils de GNOME. Aquest, al seu torn, està basat en dues llibreries: GTK+, per totes aquelles funcions purament gràfiques, i la llibreria de propòsit general GLib (desenvolupada pel mateix equip). A baix nivell, GTK+ fa ús de l'entorn de finestres XOrg i Xephyr.



Arquitectura de maemo. Llistat exhaustiu dels components que formen el sistema [9]

Desenvolupant per maemo 4.0 'Chinook'

Com hem vist, a molts nivells maemo és una plataforma idèntica a qualsevol distribució Debian. Així i tot, existeixen una sèrie de components que han estat desenvolupats per Nokia i que no se n'ha fet públic el codi font. Alguns exemple són el sistema de control del GPS, totes les funcions de connexió WiFi o la interfície gràfica Hildon.

Per tal de poder aprofitar aquestes funcionalitats, maemo ofereix una sèrie d'APIs més o menys documentades [10]. Les APIs oficials de estan escrites en llenguatge C estàndard, i disposen de *bindings* per C++ i Python.

Malgrat és possible instal·lar un compilador (per exemple, GCC) en el nostre N810, les limitacions de potència i memòria no fan aconsellable aquesta tècnica.

Així, per desenvolupar aplicacions per maemo fa falta un sistema de compilació creuada que ens permeti enllaçar codi amb les llibreries de la plataforma i compilar-lo a llenguatge assemblador ARM, independentment de l'arquitectura nadiua del sistema on realitzem el desenvolupament (x86, PowerPC...). Amb aquest fi, maemo utilitza el joc d'eines que ofereix **Scratchbox**, un projecte o*pen source* dissenyat per resoldre aquest problema.

Scratchbox inclou un compilador creuat basat en GCC, un conjunt d'*scripts* per crear un sistema de fitxers virtual i instal·lar-hi el sistema base de maemo (necessari tant pels *headers* com per les llibreries amb les quals s'han d'enllaçar els binaris) i un emulador (QEMU) per poder executar les nostres aplicacions sense haver de disposar de cap aparell extern. Scratchbox només funciona sobre Linux i sobre plataformes x86.

Java

Per tal de poder executar *software* Java en un sistema, fa falta una JVM (veure Annex 1). Malauradament, ni Sun ni Nokia ofereixen una màquina virtual per l'N810 (ni per cap altra sistema que utilitzi maemo). És un fet excepcional, donada la gran acceptació d'aquesta tecnologia en dispositius mòbils.

Per sort, existeixen alternatives (no oficials) de codi lliure que haurem d'analitzar per trobar la que millor s'adapti a les nostres necessitats:

- Llibreries de classe estàndard (necessari per poder executar Jade amb garanties)
- Alguna API gràfica (per la nostra interfície)
- Suport per JNI (com veurem més endavant serà necessari per accedir al GPS)
- Velocitat d'execució acceptable

Seguint aquests criteris, hem avaluat les solucions disponibles i hem obtingut els següents resultats:

JVM	JNI	GUI	Velocitat
cacaovm	Sí	Sí (SWT)	Moderada
PhoneME	No	No	Ràpida
JamVM	Parcial	No	Lenta
Kaffe	Sí	Sí (AWT)	Lenta

Les dues úniques implementacions que suporten JNI i ofereixen una API gràfica són Kaffe i cacaovm. La primera, però, l'hem de descartar perquè un cop instal·lada ocupa més de 100 MB, i ens deixa gairebé sense espai a la unitat arrel (que recordem, és de 256 MB) .

Per tant, **cacaovm** és la nostra única opció. Com a característiques generals, es tracta d'una implementació de JVM que ofereix compilació *Just-In-Time* (JIT) per accelerar l'execució dels programes, i utilitza la llibreria de classes de GNU coneguda amb el nom de GNU classpath.

Podríem instal·lar cacaovm a partir de les fonts, però existeix una versió compilada que inclou totes les dependències gràcies a Jalimo. El projecte Jalimo té com a objectiu proveir a dispositius mòbils basats en Linux un entorn Java estable i complet; les instruccions d'instal·lació d'aquest i d'altres paquets poden ser consultades a l'Annex 2.

3.4.2 Mòdul GPS

L'N810 disposa d'un receptor intern de GPS que utilitzarem per aconseguir les coordenades del pacient en el moment del diagnòstic (latitud i longitud). Aquestes dades poden ser de gran ajuda pels equips d'emergència, ja que els permetria conèixer a priori no només l'estat de les víctimes, sinó la distribució de les mateixes sobre el terreny.

Funcionament

maemo utilitza gpsd, un *daemon* que llegeix les dades provinents del receptor GPS (en el nostres cas, a través del port sèrie `/dev/pgps`), els hi dóna format i les distribueix a totes aquelles aplicacions que les necessiten utilitzant *sockets* TCP/IP.

En un sistema linux tradicional, generalment es controla el funcionament del GPS utilitzant únicament gpsd. En maemo, però, això no és possible perquè no podem iniciar directament aquesta aplicació.

El motiu, és que el port sèrie a través del qual ha de llegir les dades no està sempre disponible. Aquest port és creat per un *middleware*, propietari de Nokia, anomenat gpsdriver. És l'encarregat d'iniciar el xip intern GPS, carregar-hi l'almanac més recent (des de l'arxiu `/var/lib/gps/nvd_data`) per agilitzar el procés de sincronització, crear el port sèrie i, finalment, arrencar el propi gpsd. També inicia un petit *applet* al menú del sistema (`/usr/libexec/navicore-gpsd-helper`) perquè l'usuari tingui coneixement que el GPS està encès.

A més a més, gpsdriver manté un llistat de les aplicacions que han demanat accés al GPS, i quan no n'hi ha cap executant-se, atura gpsd, l'*applet* i el propi receptor per tal d'estalviar energia.

Amb tot, el temps de sincronització amb el sistema GPS varia de 3 a 5 minuts en condicions òptimes. És un temps molt elevat comparat amb altres receptors GPS del mercat, que solen sincronitzar en menys d'un minut. La causa sembla ser un *bug* que se solucionarà amb la propera actualització de maemo [11].

maemo, per tant, es veu obligat a oferir la seva pròpia *API* per accedir al navegador GPS: **Liblocation**. Es tracta d'una llibreria orientada a objectes, escrita en C utilitzant *GLib*, que posa a disposició dels programadors mètodes per iniciar, aturar i recuperar dades del navegador GPS.

Accés al GPS

Ja que l'única forma viable de controlar el GPS en l'N810 és utilitzant l'API de localització escrita en C, hem d'estudiar com accedir-hi des de la nostra aplicació Java.

Existeixen dues solucions al nostre problema: programar una *daemon* en C que emmagatzemi les dades a disc i llegir-les des de Java, o bé utilitzar JNI (veure Annex 1). Hem escollit l'opció de JNI perquè ens permetrà tenir un control real sobre el receptor GPS i consumirà menys recursos que la primera opció (no haurà d'accedir a disc contínuament).

3.4.3 Mòdul RFID

Consideracions prèvies

Tal com detallàvem en el llistat de requeriments no funcionals, haurem de treballar amb un lector (i gravador) RFID anomenat IDBlue. La decisió d'utilitzar aquest model ha estat presa novament pel grup SeNDA, però en aquest cas no existia cap altre dispositiu en el mercat que complís les necessitats del nostre projecte: l'IDBlue és l'únic lector d'RFID de mà amb una interfície Bluetooth.

Necessitàvem un aparell que es pogués comunicar amb l'N810 sense necessitat de cable (no disposa de cap interfície on hi podem connectar físicament perifèrics) i que a més tingués unes dimensions i pes adequats per poder ser portat a sobre, idealment, penjat al voltant del coll o en una bandolera (accessible en tot moment, perquè s'utilitzarà amb cada pacient).

IDBlue

IDBlue és un dispositiu lector i gravador d'etiquetes RFID amb interfície Bluetooth. Presenta un disseny portable en forma de llapis que permet ser utilitzat com a punter en pantalles tàctils (força útil en el nostre cas). Presenta un sol botó, la utilitat del qual pot ser configurada a través del seu *driver*.

IDBlue™. Fotografia del lector i gravador RFID

IDBlue funciona sobre una plataforma RISC de 8 bits amb una freqüència de rellotge de 8Mhz [12]. Disposa d'una memòria *flash* interna de 128 KB que permet emmagatzemar fins a 1000 *tag ids*. Disposa també d'una bateria NimH (recarregable) amb una autonomia de 4 hores d'ús continuat. Al cap de dos minuts de no ser utilitzat s'apaga automàticament, fet que ajuda a allargar la vida útil de la bateria.

L'IDBlue és un dispositiu Bluetooth 1.1 de classe 2, amb un rang màxim teòric de 15 metres. Opera a freqüències d'entre 2.40 i 2.48 Ghz. Al seu torn, el lector RFID opera a una freqüència de 13.56 Mhz, amb un rang màxim teòric de 4 cm. A la pràctica, aquest rang disminueix sensiblement depenent de l'etiqueta RFID utilitzada, arribant a ser necessari realitzar un contacte físic entre l'aparell i l'RFID.

És capaç de llegir etiquetes ISO 15693-2, 15693-3, Tag-it™ HF/HFI i Philips I-Code SL. En el nostre cas, utilitzarem etiquetes passives Tag-it™, de Texas Instruments, ja que no requereixen alimentació i tenen un rang de lectura acceptable.

Connexió

Oficialment, el controlador de l'IDBlue només suporta plataformes MS Windows [12]. Gràcies al *datasheet* de l'aparell [13], però, sabem que implementa el *profile* Bluetooth SPP (*Serial Port Profile*) i que utilitza un protocol sèrie RFCOMM. Existeix, per tant, una possibilitat de connectar-nos-hi de la següent manera:

1. Utilitzar BlueZ (l'*stack* Bluetooth de Linux que també utilitza maemo) per establir una connexió sèrie des de l'N810 cap a l'IDBLue.
2. Enviar les comandes necessàries, disponibles en el *datasheet*, a través de la connexió sèrie.

Per realitzar el primer pas, és possible establir una connexió de forma manual utilitzant les comandes "`hcitool scan`" i "`rfcomm connect <MAC> <channel>`". Però aquesta solució no és viable perquè, òbviament, necessitem realitzar aquesta operació de forma automàtica des de la nostra aplicació en Java.

Malauradament, ens tornem a trobar en una situació similar al cas del GPS: la versió estàndard de Java no suporta per defecte cap tipus de comunicació amb dispositius Bluetooth, perquè són funcionalitats que depenen completament del host on s'executi la màquina virtual i, per tant, necessitem d'una implementació específica (generalment a través de JNI).

Les versions de Java 2 *Mobile Edition*, pel contrari, solen oferir aquesta funcionalitat perquè implementen l'especificació **JSR-82** [14], una extensió de Java per oferir accés genèric a dispositius Bluetooth. Però la llibreria de classes que utilitzem, GNU Classpath, està basada en les especificacions de Java 2 SE i no l'implementa.

La solució trobada ha estat AvetanaBT, una implementació per Linux (basada en BlueZ) i Java 2 SE del JSR-82. Es troba en fase de desenvolupament, però està ben documentada, és gratuïta i proporciona exactament el que necessitem. L'haurem de compilar manualment utilitzant l'*SDK* de maemo.

Un cop establerta la connexió, haurem de realitzar el segon i últim pas: enviar les comandes necessàries per llegir els RFID. Això ho aconseguirem utilitzant jIDBlue, una versió per Java 2 SE (i ME) del controlador de Cathexis i Baracoda. Es troba en un estat molt inicial de desenvolupament, no s'ha actualitzat des de l'any 2005 i no suporta totes les funcionalitats del controlador original per Windows, però gràcies a que és un projecte de codi obert podrem implementar tot allò que ens faci falta.

3.5 Estudi de viabilitat

3.5.1 Viabilitat tècnica

Els coneixements tècnics necessaris pel desenvolupament del projecte inclouen la programació en Java i en C/C++, així com els paradigmes bàsics de la programació orientada a objectes i les tècniques d'enginyeria del *software*. Tots ells han estat coberts per la formació rebuda durant la carrera.

L'equip necessari inclou estacions de treball x86, dispositius mòbils (Nokia N810, on s'instal·larà i es provarà l'aplicació) i lectors RFID Bluetooth. Tot l'equip ha estat proporcionat pel departament.

3.5.2 Viabilitat operativa

El personal d'emergències, usuari final del sistema, hauria de realitzar la tasca habitual de triatge. Substituir per complet les etiquetes de triatge pel nostre sistema seria viable en un panorama on tot el personal disposés de l'equip necessari per recuperar la informació recollida en temps real.

Aquesta situació, però, és una utopia. Els recursos davant d'una emergència mèdica de grans dimensions són limitats, els contratemps imprevisibles, i de ben segur que la nostra solució no és perfecte. Utilitzar-la de forma única és arriscar les vides dels pacients en cas que, per exemple, l'aplicació fallés o algun dispositiu *hardware* deixés de funcionar (bateria, accidents, etc).

Per tant, haurem de conviure amb el mètode de triatge tradicional. Proposem que les dades siguin recollides a través de la nostra interfície (i, per tant, emmagatzemades en els dispositius), però també siguin copiades a mà en les etiquetes clàssiques (si no en la totalitat, només les parts més rellevants).

L'única modificació serà l'ús d'una etiqueta RFID per tal de poder identificar el pacient en el nostre sistema. L'usuari, per tant, haurà de llegir aquesta etiqueta abans d'abandonar el pacient. Des de la nostra interfície recordarem que aquest pas ha de ser realitzat.

3.5.3 Viabilitat econòmica

Tot el *software* utilitzat és gratuït (i, en gran part, també de codi lliure) i no suposa cap despesa addicional. Farà falta adquirir els ja esmentats dispositius Nokia N810 i els lectors RFID; està previst fer-ho amb el pressupost del departament.

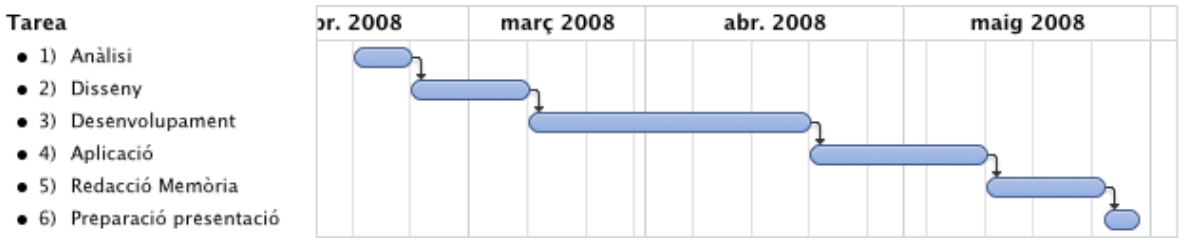
No és objectiu d'aquest projecte, però, fer un estudi del cost d'implantació de la nostra solució en situacions reals.

3.5.4 Viabilitat legal

En el sí del projecte no hem de tenir en compte qüestions legals. Les dades dels pacients són recollides i emmagatzemades en un dispositiu que no és accessibles des de fora. A més, part d'aquestes dades serien anònimes (la informació personal del pacient no sempre sol anotar-se, ja sigui perquè és desconeguda o perquè és irrellevant).

3.5.5 Planificació inicial de la proposta

Oferim una versió preliminar del diagrama de Gantt en relació al qual s'ha estructurat el projecte.



La data d'inici del projecte ha estat fixada el 18 de febrer, una setmana després de la finalització dels exàmens del primer semestre. Idealment, el projecte finalitzarà l'última setmana de maig.

L'entrega de la memòria està prevista pel dia 16 de juny. La data de la presentació serà el dia 7 de juliol.

Capítol 4. Disseny i implementació

En aquest capítol començarem realitzant una descripció global del sistema, identificant els mòduls principals en què l'hem dividit. Per cada components del projecte, anirem veient quines solucions s'han proposat per cobrir les necessitats extretes en la fase de disseny, així com els detalls de la seva implementació.

4.1 Disseny global de l'aplicació

4.1.1 Components

En el capítol d'anàlisi hem pogut identificar els components bàsics que formaran part de la nostra aplicació i el funcionament que en requerim:

- **Agent interfície gràfica**: l'encarregat d'oferir al personal d'emergències una interfície del protocol START que permeti realitzar la seva tasca de triatge sense tenir coneixement previ del mètode de triatge en sí.
- **Mòdul lector RFID**: l'encarregat de llegir l'identificador únic de l'RFID incorporat en cada etiqueta dels pacients.
- **Mòdul de localització GPS**: l'encarregat d'iniciar el receptor GPS de l'N810 i proporcionar-nos les dades de localització tant aviat com estiguin disponibles.
- **Agents d'emmagatzematge**: encarregats d'emmagatzemar les dades produïdes per la interfície, l'identificador únic RFID i la posició GPS del pacient en el moment de la diagnosi.

També hem decidit que Java serà el llenguatge utilitzat per realitzar el desenvolupament de tots ells, a excepció del mòdul de localització GPS que serà implementat parcialment en C per tal de interactuar utilitzant el *framework* JNI amb l'API de localització de maemo.

Amb aquesta distribució en ment, hem decidit organitzar l'aplicació en diferents paquets:

- **mabett**: l'arrel del projecte, contindrà les classes destinades a iniciar l'aplicació
  - **mabett.core**: contindrà les classes de control
  - **mabett.devices**: contindrà les classes corresponents als mòduls *hardware* de l'aplicació
    - **mabett.devices.bt**: pel mòdul de lectura de l'RFID
    - **mabett.devices.gps**: en el cas del mòdul de localització GPS
  - **mabett.gui**: contindrà totes les classes de la interfície gràfica
    - **mabett.gui.resources**: contindrà els recursos de la interfície (com per exemple les imatges) i les classes que els gestionen
    - **mabett.gui.triage**: per totes les classes de la interfície destinades a realitzar el procés de triatge
    - **mabett.gui.widgets**: contindrà les classes que implementin controls gràfics personalitzats
  - **mabett.triage**: en formaran part totes les classes de l'agent d'emmagatzematge
- **org.tzi.rfid.jidblue**: el controlador del lector d'RFID del projecte jIDBlue (és el nom original del paquet)

4.1.2 Arquitectura

En línies generals hem dissenyat la nostra aplicació seguint un patró arquitectural model-vista-controlador (*MVC*, *Model-View-Controller*).

- Model: representat per la classe mabett.core.Configuration, serà el component que emmagatzemarà les dades recuperades de cada sessió (preferències d'usuari) i tots aquells valors estàtics utilitzats per més d'una classe.
- Vista: representat, principalment, per la classe MetaInterface (interfície general del sistema), PreferencesComposite (vista de preferències) i TriageComposite (vista de triatge) del paquet mabett.gui.
- Controlador: representat per la classe mabett.core.Controller, serà el component encarregat de dur a terme tasques requerides per l'usuari i per altres classes

La principal raó d'utilitzar aquest patró ha estat, per una banda, la claredat que ofereix alhora d'analitzar el codi (donat que les classes estan fortament separades per funcionalitats), i per l'altra, la robustesa que dóna quan algun dels components s'ha de modificar (per exemple, podem canviar o eliminar una vista sense haver de retocar ni el model ni el controlador).

No tots els mòduls, però, han estat dissenyats amb aquest patró com a referència. L'hem adaptat a les nostres necessitats intentant aprofitar els seus beneficis sense que això ens compliqués excessivament l'etapa de desenvolupament.

4.1.3 JADE

Com hem explicat en el capítol d'estat de l'art, qualsevol agent necessita una plataforma on poder-se executar. JADE és la agència que utilitzarem, i necessitem per tant que es trobi en execució per poder iniciar l'agent interfície i, des d'aquí, anar llançant els agents d'emmagatzematge amb les dades de cada pacient.

Hem cregut convenient instal·lar JADE en l'N810 com un servei més, que s'iniciï quan encenem l'aparell. D'aquesta manera, podrem iniciar i aturar el nostre sistema en qualsevol moment amb la seguretat que JADE ja està en execució. Els detalls d'instal·lació es poden consultar a l'Annex 2.

Cal tenir en compte que la interfície podria ser aturada en tot moment, però els agents d'emmagatzematge continuaran executant-se en la plataforma de manera totalment independent. Aquest és un dels avantatges d'utilitzar la tecnologia d'agents.

4.1.4 SWT

SWT (*Standard Widget Toolkit*) és una API gràfica de codi lliure per la plataforma Java que actualment és desenvolupada (i utilitzada) pel projecte Eclipse. És una alternativa a AWT i Swing, desenvolupades per Sun Microsystems com a part de Java SE.

Utilitzarem SWT per implementar les nostres interfícies perquè, com hem pogut veure durant la fase d'anàlisi, és l'única API gràfica suportada per la llibreria de classes escollida. Des del punt de vista de disseny i implementació, haurem de tenir en compte:

- SWT recupera la filosofia original d'AWT d'utilitzar components nadius, intentant adoptar un estil consistent en totes les plataformes. Així, un mateix programa es veurà de forma diferent segons la plataforma on l'executem, però el seu comportament serà idèntic.
    - En el nostre cas, aquest primer punt implica que tots els components que es visualitzaran a l'N810 seran idèntics als que ofereix Hildon (la interfície gràfica de maemo).
- SWT no implementa l'arquitectura *MVC* de forma nadiua [15]. Per tant, l'haurem d'implementar manualment.
- SWT utilitza, en canvi, un model d'interfície d'usuari d'un sol *thread*. En aquest model, només el *thread* que ha creat la interfície pot accedir als seus mètodes. Si es vol accedir des de fora, es poden utilitzar dos mètodes que ofereix la classe Display: `syncExec(Runnable runnable)` i `asyncExec(Runnable runnable)`.

- Aquest fet l'haurem de tenir molt en compte alhora d'implementar els mòduls que es comuniquin amb components *hardware*, ja que es tracta de mòduls que s'executen de forma asíncrona i normalment han de notificar a la interfície de les dades que han recollit.

4.2 Agent interfície gràfica

En el moment en que s'inicia JADE (com a servei), es crea la plataforma amb un contenidor d'agents principal. En aquest contenidor s'hi llancen automàticament l'agent AMS (*Agent Management System*) i l'agent DF (*Directory Facilitator*). El primer és l'encarregat de controlar l'accés i la utilització de la plataforma per part de la resta d'agents; entre d'altres, ofereix un servei de pàgines blanques per conèixer quins agents s'estan executant a la plataforma. El DF és l'agent que ofereix el servei de pàgines grogues, és a dir, ens permet conèixer quins serveis ofereixen els agents que es troben en la plataforma.

Quan iniciem la nostra aplicació, ho farem demanant a JADE que es connecti a la plataforma existent i que hi llanci el nostre agent interfície, BootAgent. La seva única funció és la de iniciar la interfície gràfica, utilitzant el mètode `startWithinJade` del controlador Controller.

Per tal de dur a terme els seus objectius, els agents defineixen *behaviours*. El nostre comportament GuiBehaviour és molt simple i només s'ha de realitzar una vegada. Tenint en compte això, hem decidit instanciar un comportament ja definit a JADE anomenat OneShotBehaviour.

L'esquema de classes del nostre agent serà el següent:

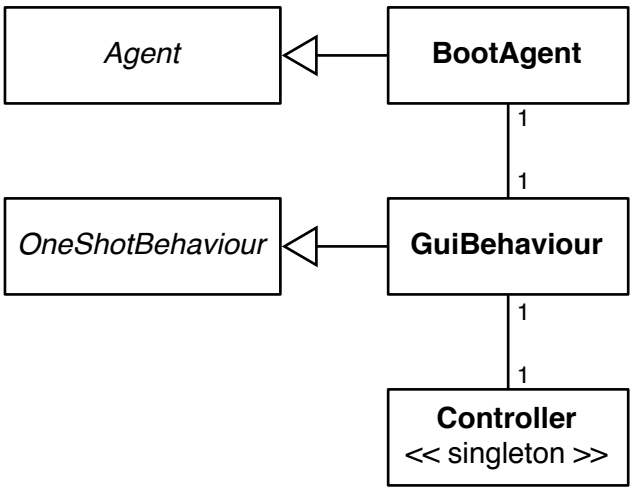


Diagrama de classes de l'Agent interfície. En cursiva les classes externes, en negreta les pròpies.

Com hem comentat a l'anàlisi, la nostra aplicació podria funcionar sense utilitzar agents. Prova d'això és la classe Boot, disponible en el mateix paquet que BootAgent, que inicia la interfície gràfica fora de JADE.

Si l'aplicació és iniciada d'aquesta manera no es llacen agents amb les dades dels pacients. De cara al desenvolupament i la fase de proves, però, ens ha estat molt útil.

4.3 Agent d'emmagatzematge

4.3.1 TriageDataAgent

Així com l'agent interfície és l'encarregat d'iniciar la interfície gràfica, les instàncies de l'agent d'emmagatzematge són creades per la pròpia interfície cada vegada que es diagnostica un pacient.

El paper d'aquest agent és absolutament trivial: un cop creat, romandrà a la plataforma sense fer res. Per aquesta raó, no implementa cap *behaviour*. Com comentàvem a la fase d'anàlisi, el nostre projecte no tractarà les dades creades, simplement les emmagatzemarà a la plataforma en forma d'agents.

4.3.2 TriageData

Internament, l'agent conté un objecte de la classe TriageData amb totes les dades d'un pacient que han estat omplertes per l'usuari del sistema utilitzant alguna de les vistes que veurem més endavant.

Aquesta classe és la nostra implementació de les etiquetes de triatge tradicionals, i per tant oferirà els camps dels quals vam parlar durant la fase d'anàlisi, a més d'alguns valors específics del nostre projecte com la localització GPS i l'etiqueta RFID del pacient.

Segons el número d'entrades d'un mateix camp podem dividir les dades en dues grans famílies:

- Camps estàtics: camps que són emplenats una sola vegada
- Camps dinàmics: camps que poden ser emplenats múltiples vegades

Segons les necessitats d'identificació dels camps, a més, també les podem dividir en dos grups:

- Camps anònims: camps dels quals no ens interessa qui els va emplenar
- Camps signats: camps dels quals volem saber qui els va emplenar i quan ho va fer.

Seguint aquesta classificació, hem definit tots els camps de la classe TriageData de la següent manera:

Valor	Estàtic	Dinàmic	Anònim	Signat	Tipus
Id del pacient (RFID)	√		√		atribut de la pròpia classe (String)
Posició GPS		√	√		classe GPSPosition
Dades personals	√		√		classe PersonalData
Signes vitals		√		√	classe VitalSignsEntry
Lesions del cos		√		√	classe InjuriesEntry
Anotacions		√		√	classe NoteEntry

Així, emmagatzemarem els camps dinàmics en vectors. Els camps signats, a més, hauran de ser una subclasse de AsRequired, el constructor de la qual emplenarà automàticament l'hora i l'identificador d'usuari quan es crei una instància la classe.

La pròpia classe TriageData serà també una subclasse de AsRequired, de manera que podrem conèixer quan va ser creat per primera vegada l'objecte i qui ho va fer.

Cal tenir en compte que el fet de permetre dades dinàmiques i/o signades no té una utilitat directa en el nostre projecte, perquè les dades són entrades una sola vegada al sistema.

Es tracta, però, de característiques que podrien ser utilitzades per altres projectes futurs, i ens ha semblat una bona idea implementar-les.

GPSPosition

Els atributs que ofereix aquesta classe són la latitud i la longitud, ambdós de tipus *long*.

PersonalData

Els atributs que ofereix aquesta classe són:

- Nom del pacient: *String*
- Sexe: enumeració Gender, de valors *MALE* i *FEMALE*.
- Edat: enter
- Adreça: *String*
- Identificació: representa qualsevol codi d'identificació que el pacient, com per exemple el seu DNI o equivalent. *String*.

VitalSignsEntry

Els atributs que ofereix són:

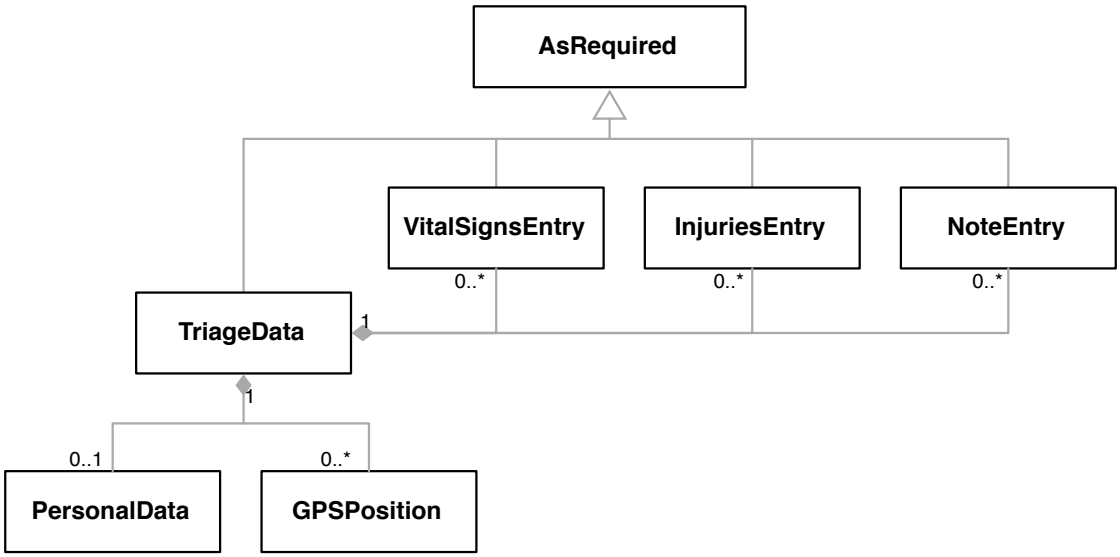
- Pressió sanguínia (en mm Hg, mil·límetres de mercuri): enter
- Pulsacions per minut: enter
- Tipus de pols: enumeració Pulse de valors *FULL*, *WEAK*, *REGULAR* i *IRREGULAR*.
- Respiracions per minut: enter
- Test capil·lar: enter
- Estat del pacient: enumeració LabelStatus de valors *DECEASED*, *IMMEDIATE*, *DELAYED* o *MINOR*.
- Estat mental: enumeració MentalStatus de valors *SANE* o *INSANE*.

InjuriesEntry

Ofereix un sol atribut, les parts del cos amb lesions: enumeració BodyPart de valors *HEAD*, *LEFT_ARM*, *RIGHT_ARM*, *CHEST*, *ABDOMEN*, *LEFT_LEG* i *RIGHT_LEG*.

NoteEntry

Ofereix també un sol atribut, una nota de tipus *String*.



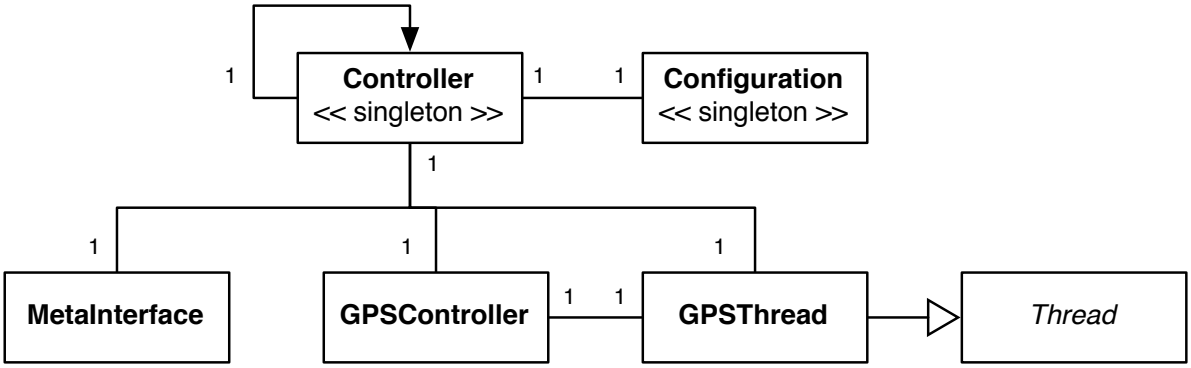
TriageData. Diagrama de classes.

La classe TriageTag, ofereix *setters* i *getters* per accedir als diferents camps i les diferents versions dels mateixos si són dinàmics. Les classes que implementen els camps, al seu torn, també ofereixen *getters* i *setters* per accedir a tots els atributs.

Totes les classes de l'agent d'emmagatzematge es troben en el paquet mabett.triage.

4.4 Controlador

El controlador, per la seva naturalesa, només s'instancia una vegada (de forma mandrosa) i és utilitzat per qualsevol classe que el necessiti. Per tal de reforçar aquest comportament, hem aplicat el patró de disseny conegut amb el nom de *Singleton*, que assegura l'existència d'una única instància per una classe i la creació d'un mecanisme d'accés global a aquesta instància. La classe que l'implementa és mabett.core.Controller.



Controlador. Diagrama de classes.

Els mètodes principals que haurà d'oferir el controlador són:

- `startWithinJade`: aquest mètode, cridat per l'agent, ha d'iniciar, per una banda, un *thread* amb el controlador GPS, que intentarà connectar-se al número mínim de satèl·lits tan aviat com sigui possible. Per altra banda, iniciarà la interfície gràfica utilitzant la classe MetaInterface. El mètode és bloquejant, i no finalitzarà fins que l'usuari tanqui l'aplicació.
- `start`: idèntic a l'anterior mètode, però utilitzat quan volem iniciar l'aplicació fora de JADE.
- `openPanel`: utilitzant aquest mètode qualsevol altra classe podrà demanar canviar la vista que actualment s'està mostrant.
- `getCurrentPanel`: de la mateixa manera, qualsevol classe podrà conèixer quina és la vista actual.
- `requiredPreferencesAreSet`: retornarà cert si les preferències d'usuari han estat entrades correctament, fals en cas que no sigui així; consultarà les dades necessàries del model de l'aplicació, la classe Configuration.
- `sendTriageData`: permetrà llançar agents emmagatzemadors TriageDataAgent del paquet mabbet.triage amb les dades recollides de la diagnosi d'un pacient.

- `getCurrentGPSPosition`: retornarà la posició actual en valors de latitud i longitud, encapsulats en una instància de la classe GPSPosition; en cas que no s'hagi pogut connectar amb el sistema GPS, retornarà la posició `0,0`.
- `quit`: cridat quan l'usuari tanqui l'aplicació, serà l'encarregat d'aturar el *thread* de captura de dades GPS i d'emmagatzemar les preferències d'usuari utilitzant el mètode `store` de la classe Configuration.

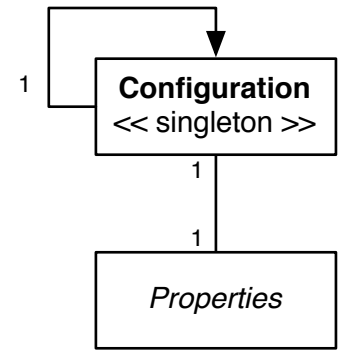
4.5 Model

El model, implementat en la classe Configuration del paquet mabett.core, és l'encarregat d'emmagatzemar dades que poden ser útils per altres classes. Tal com passava amb el controlador, aquesta classe només s'ha d'instanciar una vegada, i per tant implementa el patró de disseny *Singleton*.

Aquesta classe ha de ser capaç de recuperar (i mantenir carregats) els valors corresponents a les preferències d'usuari de l'aplicació. Java ofereix dues classes destinades a aquest fi: la classe Properties i la classe Preferences. L'última és la més completa, però malauradament no està disponible en la llibreria de classes que utilitzem a l'N810 [16]. Haurem d'utilitzar la primera; els valors que haurà d'emmagatzemar de sessió en sessió són:

- Mètode de triatge escollit (START o manual)
- Identificador i nom d'usuari
- Adreça Bluetooth del lector d'RFID IDBlue
- Idioma de l'aplicació (anglès, castellà o català)

El model també contindrà una quantitat elevada de valors utilitzats per més d'una classe; al estar definits en un sol lloc els podrem canviar de manera consistent. Seran valors constants que no es podran modificar en temps d'execució (fet que reforçarem declarant totes les variables amb els modificadors `static` i `final`).



Model. Diagrama de classes.

Els principals mètodes a implementar són:

- `getProperty`: recupera un valor de les preferències. Necessita com a paràmetres la clau (*String*) del valor a recuperar i el valor per defecte que es desitja en cas que no es trobi la clau (d'aquesta manera ens assegurem que el mètode sempre retorni un valor consistent). Internament, la classe *Properties* emmagatzema els valors en una taula *hash*.
- `setPropperty`: permet modificar un valor determinat de les preferències.
- `store`: desa les dades a disc, en el fitxer $HOME/.mabett/mabett.properties. L'estructura de l'arxiu ve determinada per la classe *Properties*.

```
#MABETT
#Fri Jun 13 13:24:45 CEST 2008
triage_method=START
userid=12344415G
idBlue_MAC=00\:A0\:96\:17\:B2\:B9
username=Xavier
language=0
```

Model. Contingut de l'arxiu de preferències

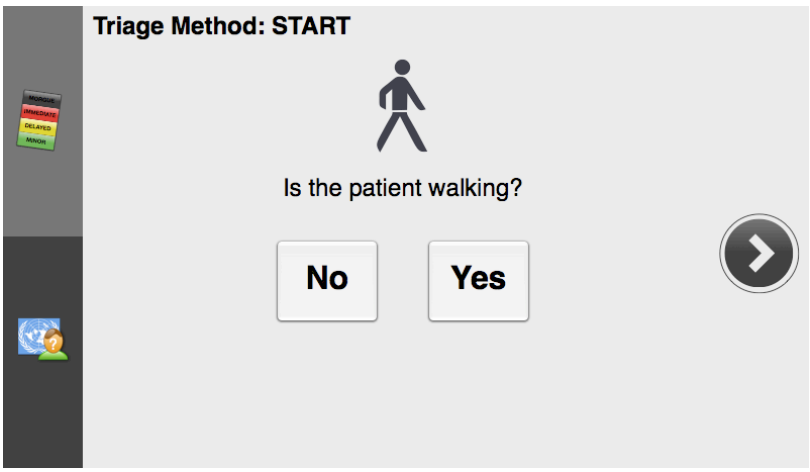
4.6 Vista general

La nostra interfície gràfica serà capaç de mostrar les següents vistes:

- Vista de preferències: per consultar o modificar les preferències de l'aplicació
- Vista de triatge: segons les preferències de l'usuari, mostrarà:
  - Vista de triatge START, per diagnosticar un pacient utilitzant l'algorisme START
  - Vista de triatge manual, per diagnosticar un pacient de forma manual

A l'inici de l'aplicació, si les preferències s'han pogut carregar es mostrarà la vista de triatge escollida. En cas contrari, es mostrarà la vista de preferències fins que les dades no hagin estat entrades.

Si l'usuari prem en qualsevol moment la tecla *ESC*, que en l'N810 està localitzada en el cantó esquerra de la pantalla, es cancel·larà el procés de triatge actual o bé, si es troba en la vista de preferències, se sortirà de l'aplicació. En tots dos casos, es demanarà una confirmació de l'operació utilitzant un *popup*.



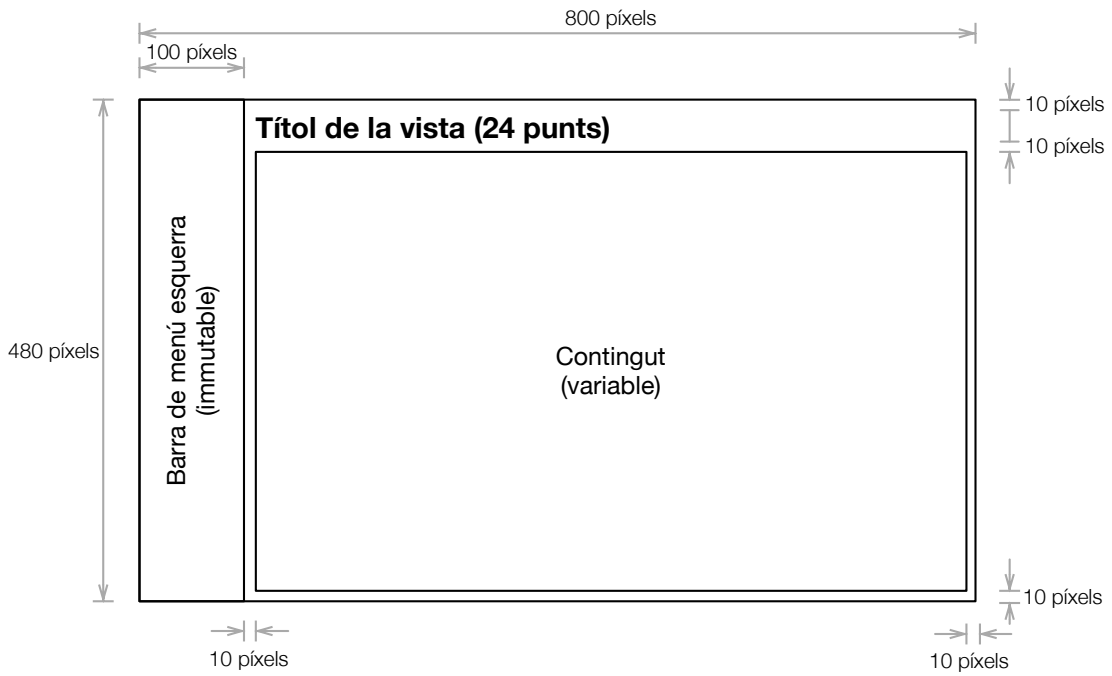
Vista general. Inici de l'aplicació.

4.6.1 Layout

Estem executant la nostra aplicació Java en un dispositiu mòbil amb un processador a 400 Mhz i 128 MB de RAM, utilitzant una versió no oficial de la màquina virtual de Java i de les llibreries de classe. La suma de tots aquests factors provoca que la velocitat d'execució del sistema (de la qual parlarem amb més detall en el següent capítol) no sigui molt elevada en comparació amb un ordinador de sobretaula.

Els punts crítics detectats en les nostres proves han estat la inicialització de les aplicacions i l'obertura de noves finestres. Per aquesta raó, hem pres la decisió d'integrar tota la interfície gràfica en una sola finestra, en pantalla completa per aprofitar al màxim l'espai disponible.

Per tal de poder mostrar les diferents vistes de què consta la nostra aplicació, actualitzarem el contingut (d'una part) de l'única finestra existent, implementada per la classe MetaInterface. El disseny d'aquesta vista es mantindrà constant.



Vista general. Disseny.

La disposició dels elements no ha estat escollida de forma arbitrària. El menú lateral està en una posició fàcilment accessible per una de les dues mans agafades al dispositiu. El contingut de cada vista ocupa una zona central, i el títol permet descobrir ràpidament quina vista està activa.

Per tal de poder mostrar més d'un element en un sol contingut, totes les vistes seran una subclasse de Composite, una classe d'SWT que expressa un conjunt de controls agrupats amb una certa estructura i una funcionalitat comuna.

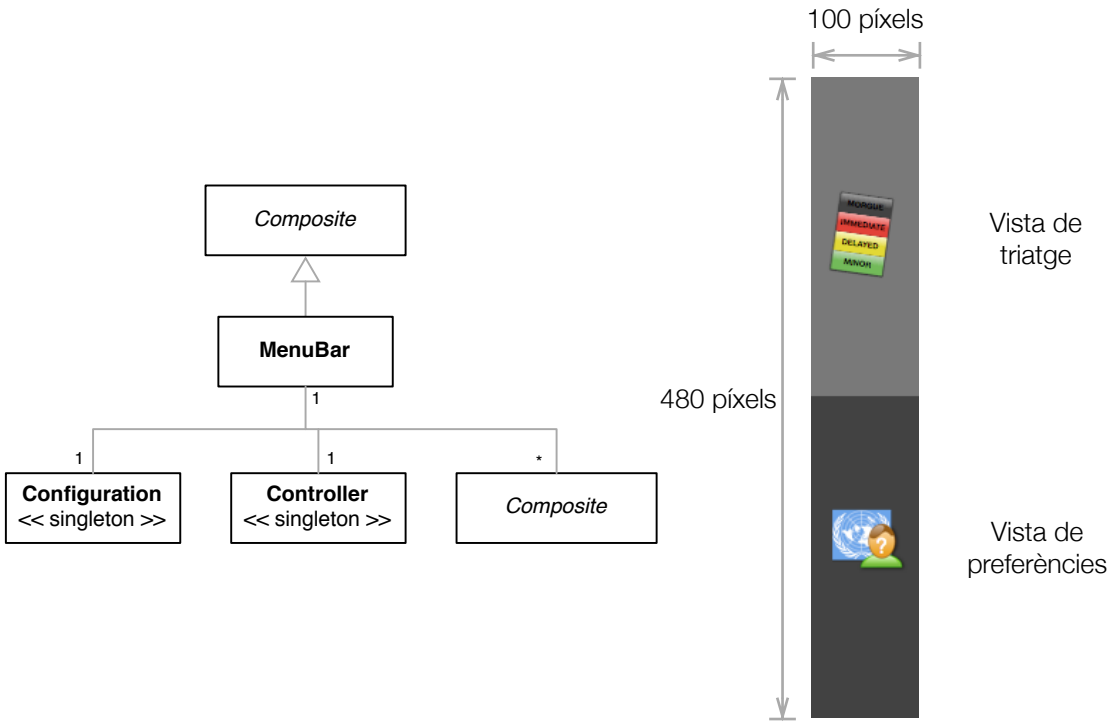
4.6.2 Menú

Ja que integrem tota la interfície gràfica en una sola finestra, hem d'oferir algun mètode a l'usuari perquè pugui canviar de vista de forma àgil amb les mans. La solució proposada és implementar una barra lateral que farà de menú.

La implementació d'aquesta funcionalitat la trobem a la classe mabett.gui.MenuBar. Es tracta també d'una subclasse de Composite dividida en dues parts iguals:

- Una part superior per accedir a les vistes de triatge
- Una part inferior per accedir a les preferències

Ambdós parts es comporten d'una manera semblant a com ho faria un botó tradicional; són també una subclasse de Composite, amb un fons i una imatge determinades, que varien de color segons si es troben o no seleccionats.



Menú. Diagrama de classes i implementació.

Com podem veure, el botó que es troba seleccionat és d'un color més clar al que no ho està per ajudar a identificar quina és la vista activa; un dels dos botons sempre estarà actiu. Per tal de permetre al controlador canviar la vista, la classe implementa un sol mètode:

- `selectButton`: canvia la vista actual per la que li passem per paràmetre. Els identificadors de vista es troben definits a la classe Configuration.

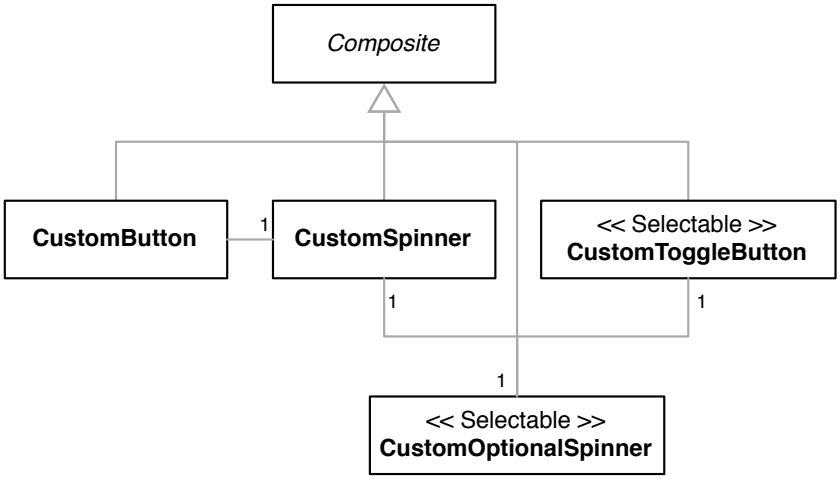
4.6.3 Controls específics

Tal com hem comentat, SWT utilitza els controls que de forma nadiua estan disponibles en cada sistema. Malauradament, tots els controls de Hildon estan dissenyats per ser utilitzats amb el llapis tàctil, i el requeriment de poder utilitzar l'aplicació amb els dits ens obligarà a implementar els nostres propis controls.

SWT desaconsella [17] crear subclasses dels controls estàndard. Per tant, tots els controls hauran de ser subclasse d'un Composite i hauran d'implementar manualment tots els mètodes que necessitem.

Hem decidit dissenyar un tipus de control simple i tres de compostos. Els controls que anomenem simples són aquells que hem creat des de zero, mentre que els compostos els hem creat utilitzant dos o més controls existents (propis o d'SWT).

Tots ells tindran una mida mínima aproximada d'un centímetre quadrat perquè puguin ser seleccionats fàcilment amb el dit. En la pantalla de l'N810, això es tradueix en uns 80 píxels quadrats. Les classes dels controls es troben en el paquet **mabett.gui.widgets**.



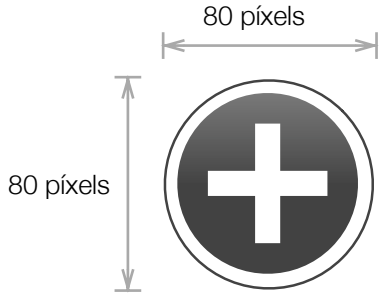
Controls específics. Diagrama de classes.

La interfície Selectable que podem veure a la figura ha estat creada per identificar controls amb dos botons i tres estats: un estat positiu (un botó pres), un de negatiu (l'altre botó pres), i un de neutre (cap botó seleccionat). Els mètodes que proposa aquesta interfície són:

- `getSelection`: retorna una enumeració de tipus ButtonSelection amb tres valors possibles: *YES*, *NO* o *NONE*, per indicar l'estat del control.
- `setSelection`: selecciona l'estat del control indicat, activant i desactivant els botons necessaris.

CustomButton

Es tracta d'un control simple que representa un botó sense estat que només serveix per llençar accions quan es premut. La particularitat que ens ha obligat a implementar aquest control ha estat el fet que necessitàvem poder incloure imatges de fons en aquests botons. SWT suporta aquesta opció, però no assegura que els controls nadius la respectin, i Hildon no ho fa.



CustomButton. Disseny i proporcions.

A més del disseny de la figura, aquest control també serà capaç de mostrar una imatge amb un signe menys i imatges amb el signe “>” en les quatre direccions cardinals (veure Annex 3).

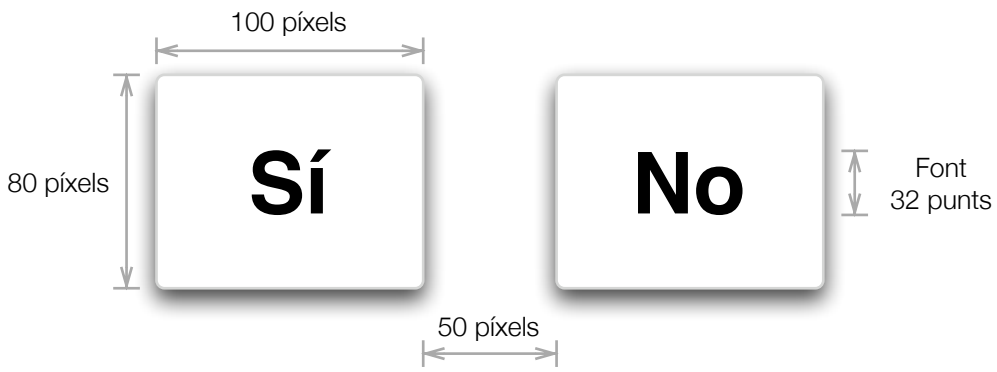
Aquest control sobrecarrega i implementa un únic mètode:

- `addMouseListener`: per defecte, aquest mètode d’SWT s’utilitza per afegir un esdeveniment en el gestor d’esdeveniments del ratolí. D’aquesta manera es poden executar mètodes propis cada cop que l’usuari interacciona amb la zona de la pantalla on es troba el control. L’hem de sobrecarregar per dirigir l'esdeveniment a la imatge del botó i no al Composite.

CustomToggleButton

Aquest control compost està format per dos *toggle buttons* botons amb el text *Yes* i *No*. Aquest tipus de botons mantenen l’estat (activats o desactivats) i el canvien quan l’usuari els prem.

Ara bé, nosaltres necessitem que quan un botó sigui seleccionat, l’altre deixi d’estar-ho automàticament; els dos podran romandre, però, sense seleccionar. En resum, s’han de comportar com una sola entitat, i aquesta és la raó de ser d’aquest control compost.



CustomToggleButton. Disseny i proporcions.

Aquesta classe implementa els mètodes de la interfície Selectable, i com passava amb l’anterior control, també sobrecarrega el mètode `addMouseListener`.

CustomSpinner

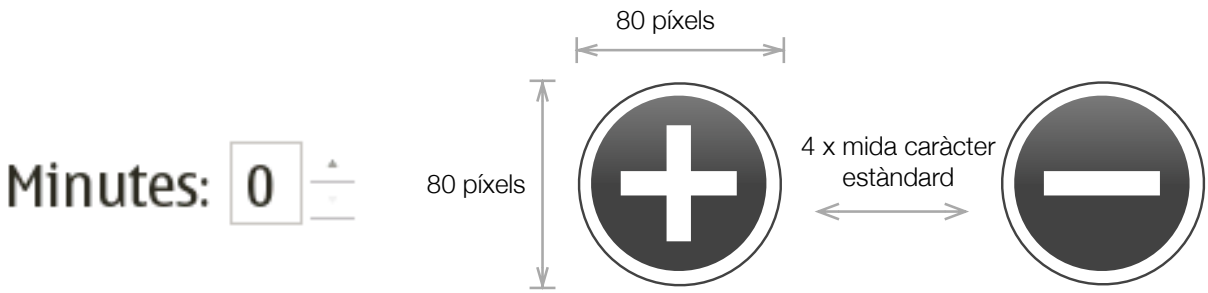
Els *spinner* són controls que es caracteritzen per tenir dos botons que permeten incrementar o disminuir el valor d’un camp de text. En Hildon, la mida d’aquests controls és estàtica, i estan pensats per ser utilitzats amb el llapis.

Per aquesta raó, hem de dissenyar un control compost de tipus *spinner* amb els dos botons i el camp de text d’una mida acceptable. No es podrà editar el valor del camp manualment (utilitzant el teclat).

Utilitzarem dos botons de la classe CustomButton amb les imatges corresponents d'afegir i sostraure, separats per un camp de text amb el valor seleccionat. La separació serà variable, i serà el triple de la mida d'un sol caràcter de la font que s'està utilitzant. Aquest valor es pot aconseguir utilitzant el mètode `getFontMetrics().getAverageCharWidth()` de la classe GC (*Graphic Canvas*) de SWT.

Els principals mètodes que implementa aquesta classe són:

- `setMinium`: estableix el valor mínim del control, per sota del qual les peticions de decrement no seran ateses
- `setMaximum`: estableix el valor màxim del control
- `setIncrement`: estableix quantes unitats s'incrementen o es disminueixen cada cop que es prem el botó
- `getSelection`: retorna el valor actual del control
- `setSelection`: estableix el valor actual del control, sempre que estigui entre els límits establerts



Spinner estàndard de Hildon a l'esquerra, i CustomSpinner a la dreta. Les proporcions entre els dos controls són reals.

CustomOptionalSpinner

Aquest control compost combina un CustomToggleButton a la part superior amb un CustomSpinner a l'inferior que només és visible quan l'usuari selecciona el botó afirmatiu del primer. Implementarà també la interfície Selectable, i el següent mètode:

- `getSpinner`: retornarà un objecte de tipus CustomSpinner per tal de poder utilitzar tots els seus mètodes.

4.7 Vista de preferències

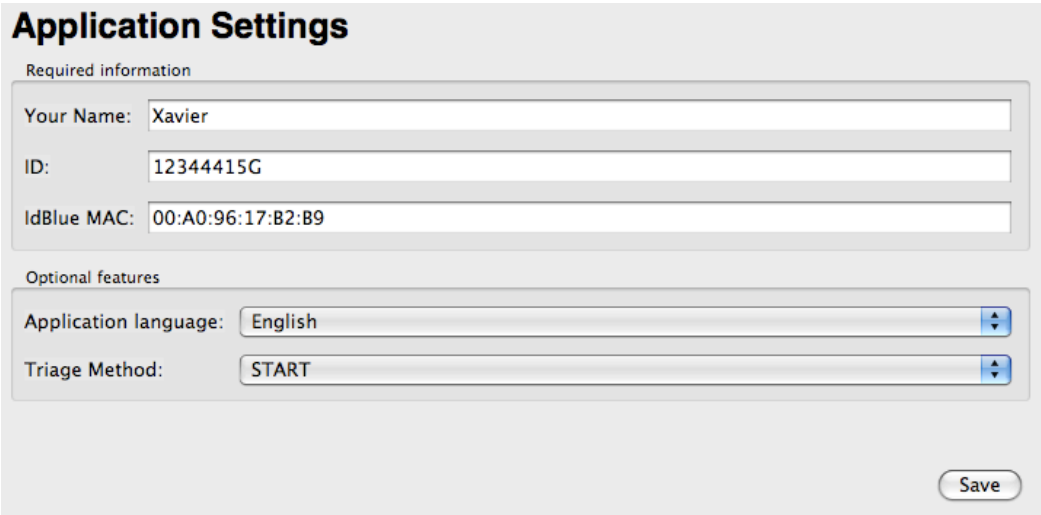
4.7.1 Descripció

La vista de preferències ens ha de servir per demanar a l'usuari certs valors necessaris pel correcte funcionament de l'aplicació. Aquests valors coincideixen amb els valors que hem definit en el model:

- Nom de l'usuari: utilitzant un camp de text, hem de demanar el nom de l'usuari que està utilitzant el sistema. És un camp obligatori.
- Identificador d'usuari: en un altre camp de text demanarem l'identificador d'usuari; cada membre de l'equip d'emergències disposarà d'algun identificador que haurà d'introduir en aquest camp. És també obligatori.
- Adreça del dispositiu IDBlue: l'últim camp obligatori, el necessitem per conèixer l'adreça física del lector RFID a utilitzar.
- Idioma de l'aplicació: permetrà escollir entre els idiomes suportats (anglès, català i castellà). Per defecte, estarà en anglès. Implementat amb un menú desplegable, és un camp opcional.
- Mètode de triatge: START o manual (cap). Per defecte, estarà seleccionat el mètode START. Implementat també amb un menú desplegable, és un camp opcional.

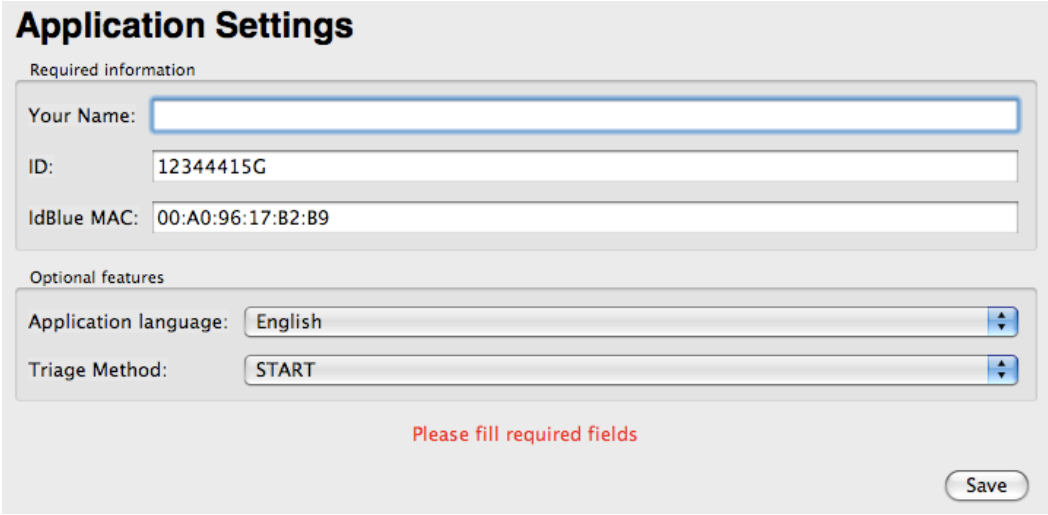
4.7.2 Disseny

Aquesta vista està pensada per ser utilitzada amb el llapis, no amb les mans. La raó és que les dades que s'hi presenten han de ser introduïdes un sol cop, i mai durant el procés de triatge en si. A més a més, els tres camps obligatoris necessiten dades que s'han d'entrar obligatòriament amb el teclat, raó per la qual no podríem utilitzar únicament els dits encara que volguéssim.



Vista de preferències. Implementació.

La interfície s'ha d'assegurar que l'usuari ha entrat les dades obligatòries. Per fer-ho comprova que els camps no estiguin buits i que l'adreça Bluetooth sigui de la forma XX:XX:XX:XX:XX:XX (utilitzant l'expressió regular (`[a-fA-F0-9]{2}:){5}[a-fA-F0-9]{2}`). Si no és així, mostra un avís a l'usuari quan aquest intenta desar els canvis.



Vista de preferències. Avís de dades incorrectes

4.8 Vista de triatge

4.8.1 Descripció

L'objectiu principal del projecte era crear una interfície pel protocol de triatge simple START. Així i tot hem volgut preparar la nostra aplicació per poder ser utilitzada amb qualsevol altre mètode de triatge.

La raó és senzilla: START no és l'únic protocol existent, ni és aplicable a totes els situacions. Hi han mètodes creats específicament per situacions bèl·liques (triatge invers), d'altres pensats per a nens o fins i tot mètodes dissenyats per ser aplicats en instal·lacions mèdiques quan els recursos disponibles són inferiors als necessaris per atendre a tots els pacients.

Per tant, hem volgut preparar el nostre sistema per poder-hi afegir mètodes de triatge simplement implementant la vista que correspongui al nou mètode.

4.8.2 Disseny

TriageComposite

Aquesta classe del paquet mabett.gui serà l'encarregada de crear totes les vistes de triatge juntament amb la vista d'edició manual. Internament, estarà formada per un sol Composite on hi mostrarà la vista de triatge. Quan aquesta finalitzi, i si així ho desitja, mostrarà la vista d'edició manual (implementada a la classe TTBuilder) per poder modificar o afegir qualsevol dada abans de ser emmagatzemada.

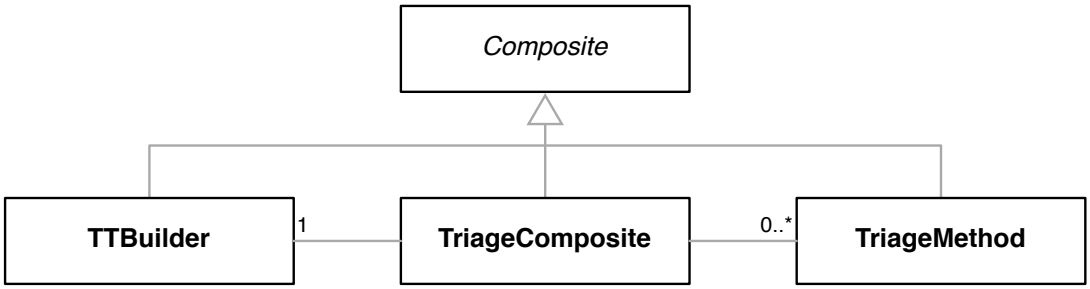
Els mètodes que haurà d'implementar aquesta classe són:

- `newCase`: inicia un nou procés de triatge mostrant la vista del mètode escollit a les preferències. En cas que l'usuari no hagi escollit cap mètode, mostrarà directament la vista d'edició manual
- `openBuilder`: obre la vista d'edició manual

TriageMethod

Es tracta d'una classe abstracta del paquet mabett.gui.triage que haurà de ser implementada per qualsevol protocol de triatge que es vulgui afegir a l'aplicació. Consta d'un sol mètodes:

- `newTriageProcess`: mètode cridat per TriageComposite quan l'usuari desitgi iniciar un nou procés de triatge amb el nostre protocol. La crida té un sol paràmetre de la classe TriageData, que és on haurem d'emmagatzemar tota la informació del pacient recollida.



Vista de triatge. Diagrama de classes.

4.9 Vista START

4.9.1 Descripció

Hem d'implementar una interfície interactiva pel protocol START. Aquesta és, juntament amb la vista manual que analitzarem en el següent punt, la part més important de la nostra aplicació.

Durant l'anàlisi vam definir tots els passos que l'algorisme du a terme per assignar un estat a cada pacient segons les seves constants vitals. Ja que volem que una persona sense coneixements del protocol START pugui fer servir la nostra aplicació, creiem que la millor forma d'implementar aquest algorisme és utilitzant un assistent.

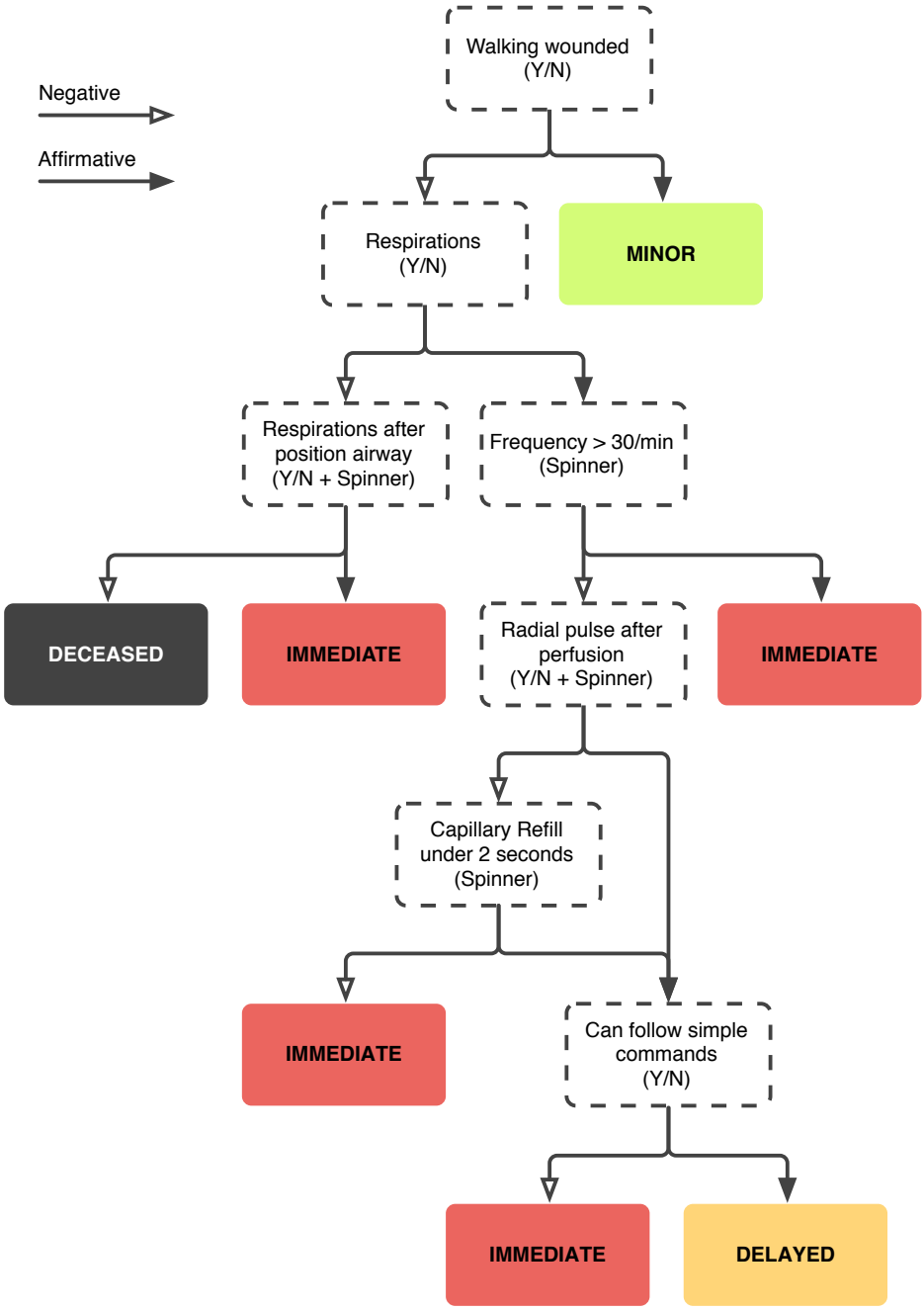
L'assistent preguntarà les dades del pacient en l'ordre especificat per l'algorisme. Segons la resposta a cada pregunta, demanarà una o altra informació fins a poder assignar-li un dels quatre estats.

Al finalitzar l'algorisme, la vista START es tancarà i en el seu lloc es mostrarà la vista manual, on el personal mèdic podrà modificar qualsevol dada (inclòs l'estat del pacient assignat automàticament) o afegir-ne de noves. Serà aquesta classe la que, finalment, cridarà al controlador per tal que llanci un nou agent d'emmagatzematge amb les dades recollides.

4.9.2 Disseny

Creiem que la millor forma de dissenyar l'assistent és utilitzant un arbre de decisió binari, on cada node representarà una pregunta a realitzar i les fulles l'estat escollit. L'esquema resultant el podem observar en la figura de la pàgina següent, on a més a més hi hem inclòs el tipus de pregunta que se'ns demana a cada node:

- Per conèixer la resposta a una pregunta de tipus afirmatiu/negatiu (*Y/N*), utilitzarem un control de la classe CustomToggleButton.
- En el cas de preguntes quantitatives (*Spinner*), utilitzarem un CustomSpinner.
- Finalment, en el cas de preguntes quantitatives opcionals (*Y/N + Spinner*), utilitzarem un control de la classe CustomOptionalSpinner.



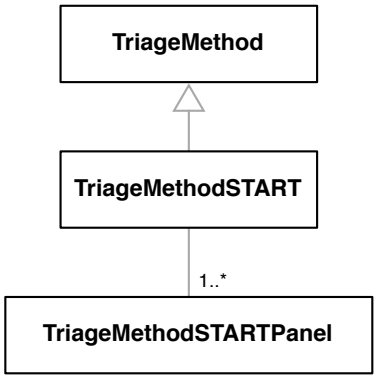
Vista START. Arbre binari de decisió de l'algorisme START.

Podem dissenyar aquest arbre de dues formes ben diferents:

- Creant tantes vistes com nodes té l'arbre (sense contar les fulles), i anar mostrant la vista adequada a mesura que l'usuari respongui a les preguntes.
- Emmagatzemar en l'arbre la pregunta en qüestió, implementar una sola vista i anar actualitzant-la.

La segona opció ens sembla molt més recomanable per les limitacions de memòria i potència que ja hem comentat anteriorment.

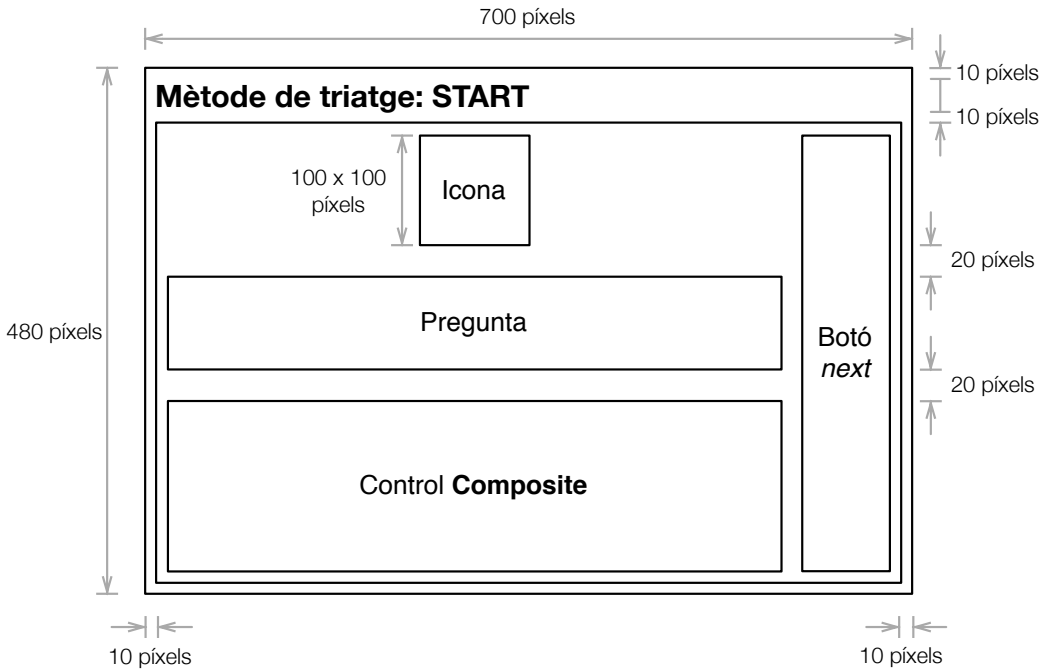
TriageMethodSTART



Vista START. Diagrama de classes

Aquesta classe, subclasse de TriageMethod**,** representarà l'assistent del mètode START. Volem mantenir una interfície el més simple possible, i per tant l'únic que mostrarem serà:

- Una icona representativa de cada pregunta, que permetrà a aquells usuaris que ja hagin utilitzat l'aplicació identificar ràpidament què s'està preguntant a partir d'una imatge (imatges disponibles a l'Annex 3).
- El text de la pregunta en sí.
- El control necessari per respondre la pregunta.
- Un botó a prémer per passar al següent nivell de l'algorisme (serà un botó de la classe CustomButton).



Vista START. Disseny general.

Internament, aquesta classe crearà l'arbre de decisió binari i l'anirà recorrent des de l'arrel fins a les fulles, mostrant el contingut de cada node en la seva pròpia vista.

Els principals mètodes que implementa són:

- `buildTree`: crea l'arbre de decisió binari, amb tants TriageMethodSTARTPanel com nodes hi han a l'arbre.
- `showPanel`: mostra un dels nodes de l'arbre.
- `next`: descendeix un nivell en l'arbre de decisió binari; si el node actual és una fulla, demanarà a a TriageComposite que mostri la vista manual TTBuilder i finalitza l'assistent. En cas contrari, crida al mètode `showPanel` perquè mostri el node fill.
- `newTriageProcess`: inicia de nou l'algorisme de decisió mostrant l'arrel de l'arbre.

TriageMethodSTARTPanel

Aquesta classe abstracta representa un node de l'arbre de decisió. Haurà de contenir la icona, la pregunta i el tipus de control a mostrar. I diem el tipus i no el control en sí perquè aquests es crearan en la classe TriageMethodSTART, que els anirà mostrant segons ho demanem.

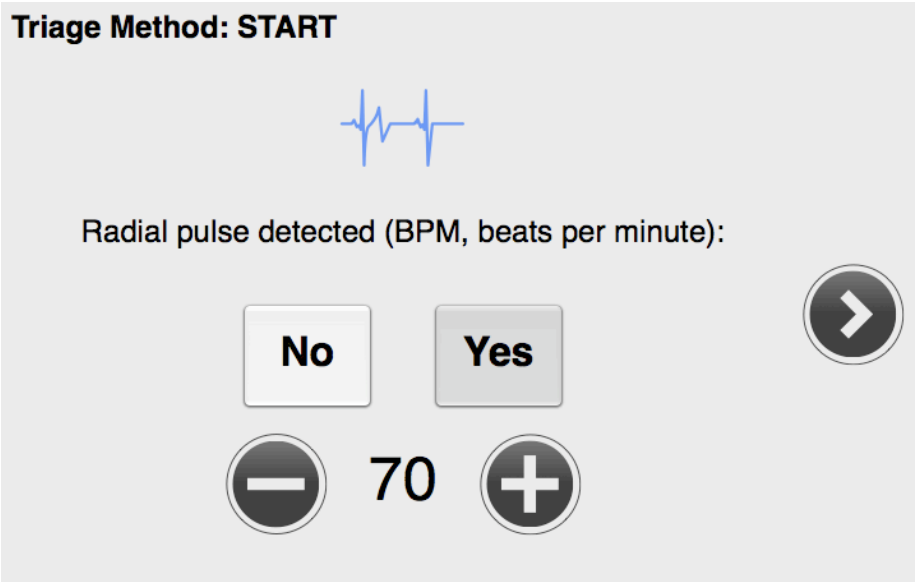
A més a més, cada node haurà de conèixer quin són els seus dos fills per tal de comunicar a l'algorisme que recorre l'arbre quin dels dos és el següent node a visitar, tenint en compte les dades introduïdes per l'usuari.

La classe haurà d'oferir els següents mètodes:

- `setAffirmativeNode`: enllaça amb el node corresponent a una avaluació positiva.
- `setNegativeNode`: enllaça amb el node corresponent a una avaluació negativa.
- `getImage`: retorna la imatge a mostrar.
- `getQuestion`: retorna el text de la pregunta a mostrar.
- `getControlType`: retorna el tipus de control que s'ajusta a la pregunta.
- `preEvalute`: aquest mètode serà cridat per TriageMethodSTART abans de mostrar un determinat node, i permet que inicialitzem els controladors amb valors personalitzats.
- `evaluate`: aquest mètode serà cridat quan l'usuari decideixi prémer el botó de "següent". Ha de retornar *true* si l'usuari ha entrat totes les dades necessàries per poder decidir quin dels dos fills és l'escollit.
- `postEvaluate`: serà cridat només si l'anterior mètode s'avalua a cert, i haurà de retornar el node escollit.

Els tres últims mètodes són abstractes i han de ser implementats per cada instància.

Amb tot, la classe TriageMethodSTART i TriageMethodSTARTPanel utilitzen el patró de disseny Template, perquè en la primera classe definim l'estructura de l'algoritme a utilitzar i en les diferents instàncies de la segona classe implementem cada node de l'arbre de decisió binari.



Vista START. Implementació d'un dels nodes amb un control de tipus CustomOptionalSpinner

4.10 Vista d'edició manual

4.10.1 Descripció

La vista d'edició manual ha de mostrar una interfície amb tots aquells camps habituals de les etiquetes de triatge vists durant l'anàlisi. A més, en cas que l'usuari hagi executat amb anterioritat un algorisme de triatge, la vista ha ser capaç de llegir les dades ja recollides i mostrar-les en els camps adequats. Totes les dades de les vistes s'emmagatzemaran en un objecte de la classe TriageData compartit entre totes les vistes.

Per altra banda, aquesta vista serà l'últim pas abans de crear l'agent d'emmagatzematge; per tant, també haurà d'oferir una interfície per llegir l'etiqueta RFID col·locada en el pacient.

Amb tot, ens trobem que hem d'oferir un número elevat de dades, que hem decidit dividir en cinc vistes diferents:

- Informació general: permetrà afegir o modificar dades relacionades amb els signes vitals del pacient.
- Lesions localitzades: oferirà un esquema del cos humà on es podran seleccionar parts del cos lesionades.

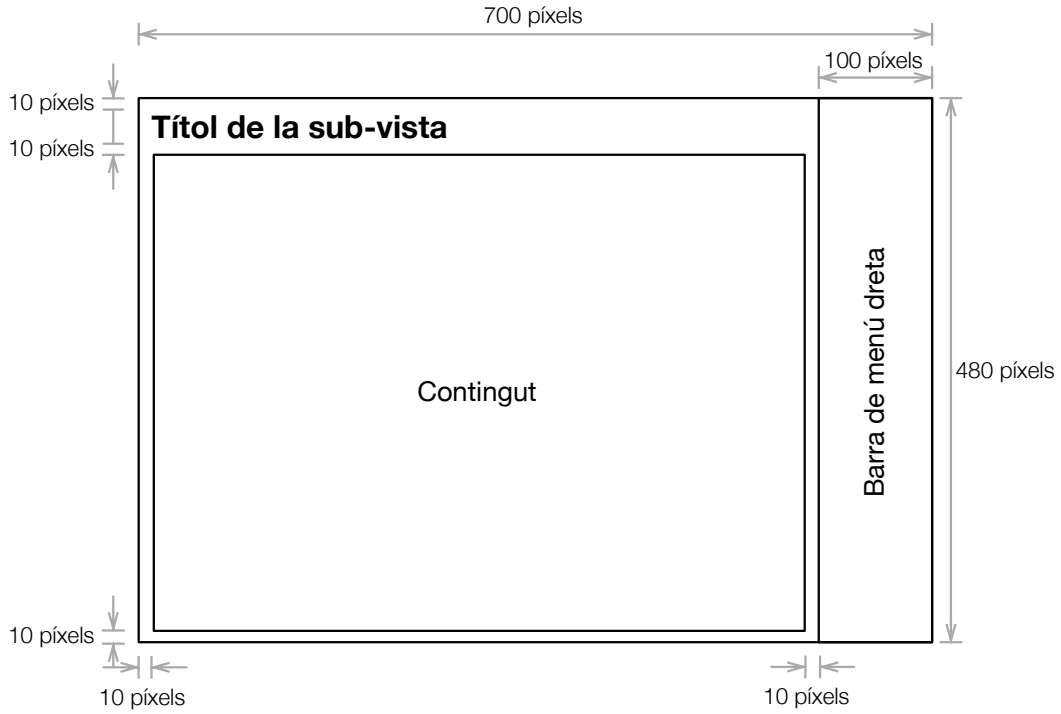
- Informació personal del pacient: permetrà introduir dades personals com el nom o l'adreça del pacient.
- Notes addicionals: oferirà un espai en blanc perquè el personal d'emergències pugui escriure notes.
- Confirmació: oferirà un petit assistent per tal de recuperar l'identificador de l'etiqueta RFID del pacient.

Tal com passava amb la vista de preferències, alguns d'aquests camps hauran de ser introduïts a través del teclat, així que la vista no podrà ser utilitzada exclusivament amb els dits.

4.10.2 Disseny

TTBuilder

Aquesta classe tindrà una funcionalitat molt semblant a la de la classe MetaInterface. Definirà el *layout* de tota la vista d'edició manual i mostrarà diferent vistes utilitzant un menú lateral.



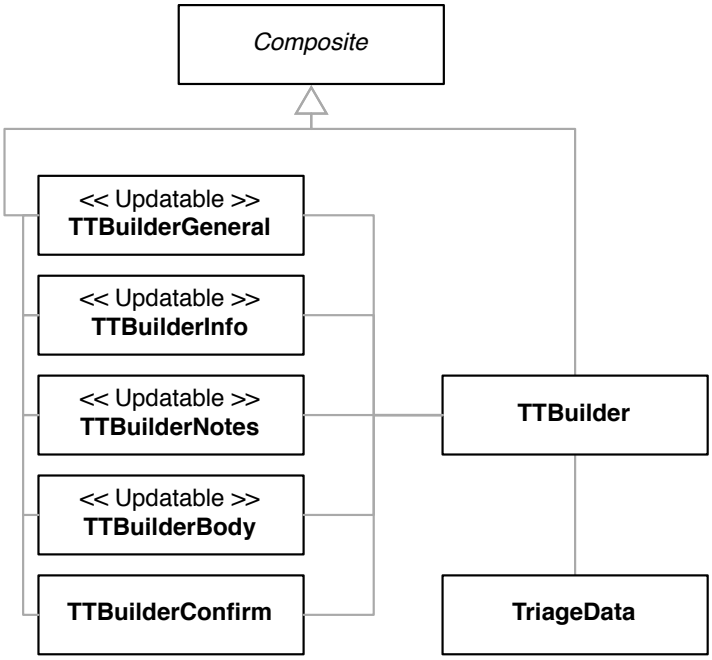
Vista d'edició manual. Disseny.

Al contrari que la classe MetaInterface, però, el menú lateral estarà integrat en la pròpia classe. Deixant de banda que haurà de mostrar cinc elements, el seu funcionament serà idèntic.

Els mètodes que implementarà aquesta classe són:

- `setTriageData`: demanarà a totes les vistes que actualitzin les seves dades i mostrarà la vista d'informació general.
- `showPanel`: permetrà canviar de vista.
- `saveData`: informarà a totes les vistes que desin les dades introduïdes en els seus camps. Serà cridada just abans d'enviar un nou agent d'emmagatzematge.

Per facilitar la feina d'informar a totes les sub-vistes existents de que han d'actualitzar o desar les dades en un moment donat, hem creat la interfície Updatable, amb els mètodes `saveData` i `loadData`.



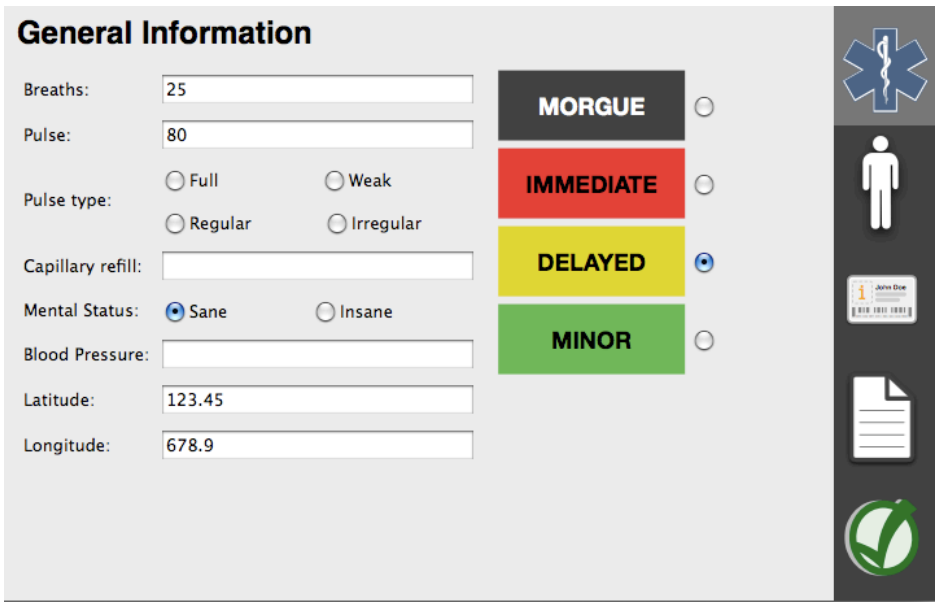
Vista d'edició manual. Diagrama de classes.

TTBuilderGeneral

Serà la vista encarregada de mostrar els signes vitals del pacient així com l'estat de triatge assignat.

Alguns camps d'aquesta vista, com per exemple les respiracions per minut, el pols o la pròpia categoria poden haver estat introduïts durant el procés de triatge; lògicament, es podran modificar si així ho desitja l'usuari.

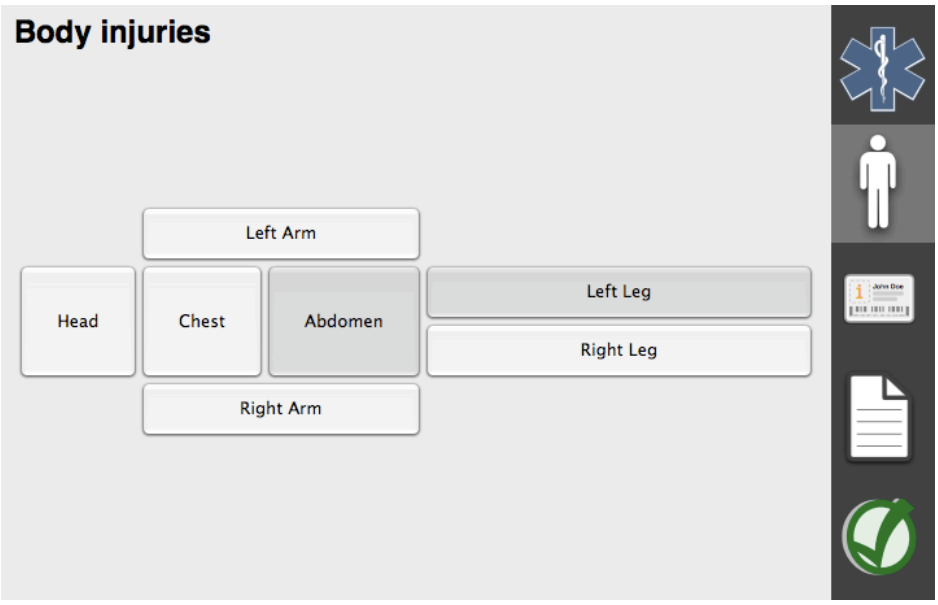
El camp de localització GPS es mostra a caire informatiu, ja que en aquesta vista s'intentarà recuperar la posició actual amb la crida a `getCurrentGPSPosition` al controlador de l'aplicació i emmagatzemar-la en l'objecte TriageData.



Vista d'edició manual. Vista d'informació general.

TTBuilderBody

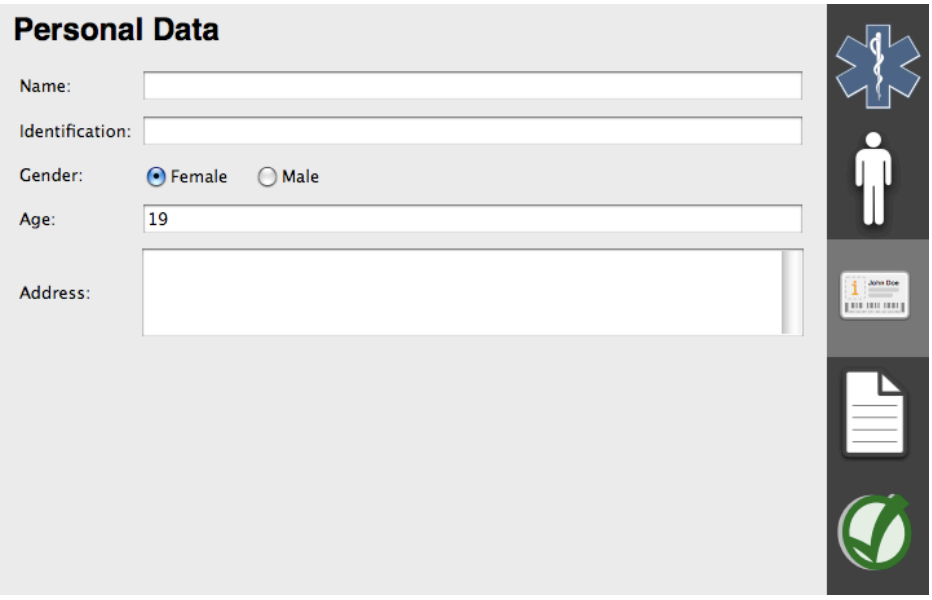
En aquesta vista mostrarem l'esquema d'un cos creat amb botons de tipus *toggle*. Podran ser seleccionats de manera individual i serviran per que el personal mèdic pugui indicar zones del cos del malalt que requereixen especial atenció.



Vista d'edició manual. Vista de lesions localitzades.

TTBuilderInfo

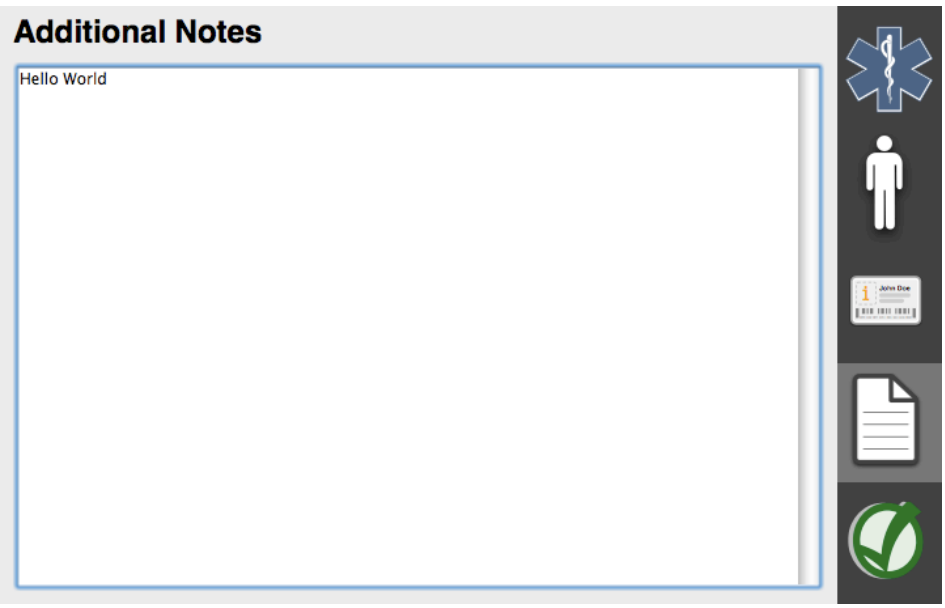
Aquesta vista mostrarà camps d'informació personal del pacient.



Vista d'edició manual. Vista d'informació personal.

TTBuilderNotes

Mostrarà un camp de text per ser emplenat amb la informació que es consideri rellevant.



Vista d'edició manual. Vista de notes.

TTBuilderConfirm

La funcionalitat d'aquesta vista difereix de la resta. Aquest serà el punt final de qualsevol procés de triatge, i el seu objectiu és llegir l'identificador únic RFID de l'etiqueta del pacient i emmagatzemar-lo juntament amb les dades recollides en la instància compartida de la classe TriageData.

El procés de lectura de l'RFID es durà a terme en un *thread* separat utilitzant la nostra classe IDBlueController. Es realitzarà una connexió amb l'IDBlue, es demanarà que ens retorni el pròxim identificador RFID que sigui llegit i finalment ens desconnectarem. El procés complet es pot realitzar en menys de 5 segons.

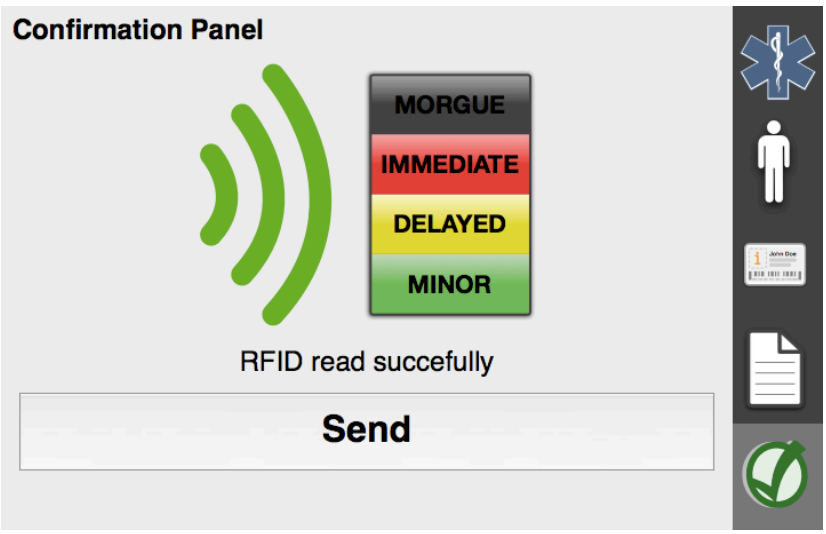
Al contrari que el sistema GPS, que està funcionant contínuament, per cada nou procés de triatge ens connectarem i desconnectarem de l'IDBlue. D'aquesta manera, intentarem reduir el consum de les bateries del lector RFID i de l'N810.

La vista, al mateix temps, anirà mostrant en temps real els passos que ha de seguir l'usuari. Tant si s'aconsegueix llegir l'RFID com si no, al final del procés es permetrà "enviar" les dades.

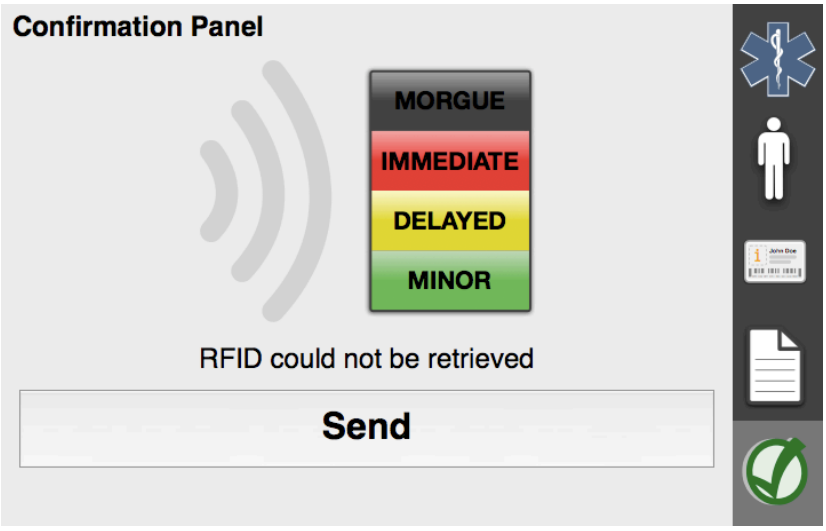
Com ja hem anat realitzant en altres casos, crearem una sola vista i anirem actualitzant els seus camps segons convingui: en aquest cas, només haurem de mostrar una imatge representativa del procés de lectura RFID, un missatge que indicarà què ha de fer l'usuari i un botó perquè ens comuniqui quan ho ha fet. El procés de lectura RFID serà el següent:

1. Al iniciar la vista, demanem a l'usuari que encengui el seu lector RFID. Quan premi el botó de la pantalla iniciarem el *thread* del mòdul RFID.
   1.1. Si no aconseguim iniciar el mòdul RFID (per exemple, si Bluetooth està desactivat), o bé si no podem contactar amb l'IDBlue, anirem a B.
   1.2. Si hem contactat, demanarem a l'usuari que llegeixi l'etiqueta RFID
      1.2.1. Si rebem el seu valor, anirem a A
      1.2.2. Si no el rebem, anirem a B

A. Mostrem un missatge conforme hem llegit l'etiqueta i el botó mostrarà la paraula "Enviar". Quan l'usuari el premi, cridarem al mètode `sendTriageData` del controlador enviant-li l'objecte TriageData emplenat i finalitzarem el procés de diagnosi del pacient cridant al mètode `newCase` de la classe TriageComposite.
B. Mostrem un missatge conforme no hem pogut llegir l'etiqueta i el botó mostrarà el missatge "Enviar". A partir d'aquí, realitzarà el mateix procés que A.



Vista d'edició manual. Vista de confirmació amb lectura d'RFID correcta (cas A)



Vista d'edició manual. Vista de confirmació amb lectura d'RFID incorrecta (cas B)

4.11 Mòdul GPS

4.11.1 Descripció

Tal com hem vist durant la fase d'anàlisi, aquest mòdul serà l'encarregat de gestionar el receptor GPS de l'N810. Per fer-ho, hem decidit utilitzar el *framework* JNI de Java, perquè l'API de maemo pel control del GPS està escrita en C.

El disseny i la implementació d'aquest mòdul es troba dividit, per tant, en una classe Java disponible en el paquet mabett.devices.gps, i en una llibreria JNI escrita en C que implementa els mètodes nadius declarats en Java (veure Annex 1).

4.11.2 GPSController.java

GPSController és la classe que ens permetrà interactuar amb el receptor GPS des de Java. Com qualsevol classe que fa ús de JNI, defineix tots aquells mètodes nadius que s'han d'implementar en C utilitzant el modificador *native*. Aquests mètodes són:

- `native private void gpsdInit()`: serà l'encarregat d'iniciar el receptor GPS.
- `native private boolean gpsdCheckConnection()`: retornarà cert si ja es disposen de dades de localització, fals en cas contrari.
- `native private GPSPosition gpsdGetLocation()`: retornarà la posició actual en una instància de la classe GPSPosition.
- `native private void gpsdFree()`: aturarà el receptor GPS i alliberarà tots els recursos de memòria de C.

Degut a la complexitat de JNI, els mètodes nadius implementen funcionalitats molt simples. Com podem observar, no hi ha cap mètode que explícitament es connecti al receptor GPS. Hem optat per anar comprovant si hi han dades disponibles cridant el mètode `gpsdCheckConnection` des de Java:

- `GPSController`: constructor per defecte de la classe que cridarà al mètode nadiu `gpsdInit`.
- `connect`: mètode bloquejant que iniciarà un *timer* per anar comprovant si hi han dades disponibles a través del mètode nadiu `gpsdCheckConnection`. Retornarà quan aquest mètode retorni cert, o bé quan hagin passat 5 minuts (*timeout*).
- `getCurrentPosition`: si el sistema està connectat, retornarà l'últim valor del receptor GPS.
- `isConnected`: mètode que ens permetrà conèixer si `connect` ha retornat perquè efectivament hem pogut detectar suficients satèl·lits i obtenir la nostra posició, o be perquè s'ha esgotat el *timeout*.
- `finalize`: mètode privat cridat pel recol·lector de brossa de Java quan un objecte està a punt de ser esborrat. L'utilitzarem per cridar a `gpsdFree`.

4.11.3 GPSController.c

Aquesta és l'única part del nostre projecte que no està escrita en Java. Consta d'un conjunt de funcions que han d'implementar els mètodes que s'han definit amb el modificar nadiu en la classe GPSController.

Consta d'un sol arxiu que ha de ser compilat utilitzant l'SDK de maemo i instal·lat en el directori `/usr/lib/jni/` de l'N810.

Com hem comentat durant l'anàlisi, Liblocation és l'API amb la que haurem d'interactuar per comunicar-nos amb el receptor GPS. Utilitza GLib, fet que ens obliga a realitzar certes crides a funcions d'aquesta llibreria.

gpsdInit
En aquesta funció inicialitzarem la llibreria GLib i recuperarem una instància del controlador GPS que ofereix LibLocation. També calcularem els identificadors de classe i mètode per poder crear, més endavant, instàncies de la classe GPSPosition des de JNI.

Crearem també un *thread* que executarà un *main loop* de GLib en un context propi. Això és necessari perquè GLib està dissenyada per ser utilitzades en programes amb un bucle de gestió d'esdeveniments. Sense aquest *thread* no rebríem les dades de posició perquè no s'actualitzarien mai.

gpsdCheckConnection
Comprovarà si l'estructura de dades que ens ofereix LibLocation ja disposa de valors actuals de localització. Retornarà cert si és així, o fals en qualsevol altre cas.

gpsdGetLocation
Crearà una nova instància de la classe GPSPosition utilitzant els identificadors calculats anteriorment, amb els valors de latitud i longitud extrets de l'estructura de dades de LibLocation.

gpsdFree
Aturarà el *thread* amb el *main-loop* i el controlador GPS. Alliberarà també els recursos de GLib necessaris.

4.12 Mòdul RFID

4.12.1 SerialBluetoothController

Tal com hem indicat en la fase d'anàlisi, el primer pas a realitzar és connectar-se físicament amb l'IDBlue utilitzant els mètodes de control de Bluetooth que ofereix l'especificació JSR-82.

Aquesta primera part ha resultat un procés trivial si coneixem l'adreça MAC de l'aparell, valor que podem recuperar des del controlador de la nostra aplicació ja que forma part dels camps obligatoris que l'usuari haurà d'omplir per poder utilitzar el sistema.

Els mètodes necessaris per dur a terme aquest procés es troben en la classe SerialBluetoothController:

- `SerialBluetoothController(String connectionURL)`: constructor de la classe, necessita l'anomenada URL de connexió. Aquesta no és més que un *string* amb la següent estructura: `btspp://AABBCCDDEEFF:1;master=true`.
  - `btsp://` indica que es tracta d'una connexió sèrie sobre bluetooth.
  - `AABBCCDDEEFF` haurà de ser substituït pel valor de l'adreça MAC del dispositiu al qual ens volem connectar.
  - `:1` és el canal de connexió.
  - `master=true` indica que nosaltres som els iniciadors de la connexió.
- `getOutputStream`: retorna un OutputStream de la connexió sèrie establerta a partir de l'URL subministrada.
- `getInputStream`: retorna un InputStream de la connexió sèrie.
- `closeSerialConnection`: tanca la connexió sèrie.

Com podem veure, aquesta és una classe totalment independent de l'IDBlue, i podria ser utilitzada per establir una connexió amb qualsevol dispositiu Bluetooth que utilitzés un *profile* sèrie.

4.12.2 IDBlueController

Aquesta classe serà la responsable d'enviar i rebre les comandes necessàries per llegir els identificadors RFID. Utilitzarà **org.tzi.rfid.jidblue.IDBlue**, el controlador existent de codi lliure en Java per l'IDBlue.

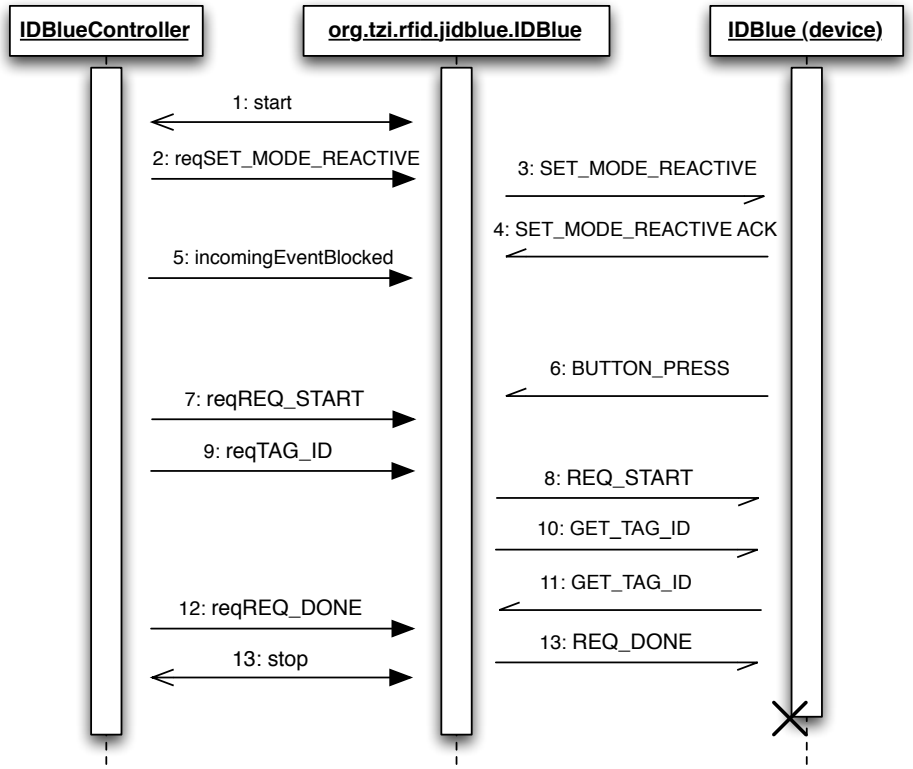
El controlador ofereix un mecanisme d'enviament i recepció d'una part dels missatges que apareixen en les especificacions del producte [13]. Per sort, amb la part implementada en tenim prou per poder recuperar un RFID en mode reactiu, és a dir, de manera asíncrona quan l'usuari premi el botó de l'IDBlue.

Precisament, hem hagut d'implementar un nou mètode d'inicialització del mode reactiu (`reqSET_MODE_REACTIVE`) que en el controlador original estava pensat per les primeres versions de l'IDBlue i no funcionava correctament amb la versió 2.4 del *firmware* que té el nostre dispositiu.

Un cop realitzat aquest canvi en el controlador, només ens resta implementar els següents mètodes:

- `connect`: obrirà una connexió amb el dispositiu utilitzant els *streams* que ens proporcioni la classe SerialBluetoothController.
- `readRFID`: el principal mètode de la classe, ens permetrà recuperar el següent RFID que llegeixi l'IDBlue de forma asíncrona.

- `disconnect`: tancarà la connexió amb l'IDBlue



Mòdul RFID. Diagrama de seqüència d'una operació de lectura `readRFID`

Capítol 5. Proves i integració

En aquest capítol es descriuen tot un seguit de proves que s'han dut a terme per avaluar el sistema i comprovar que compleix amb els requisits exposats. Es provaran els diferents components de forma individual, per acabar realitzant un test d'integració de tot el sistema.

5.1 Proves

5.1.1 Mòdul GPS

Les proves del mòdul GPS han anat orientades a comprovar dos aspectes principals: que el sistema era capaç de connectar-se al receptor GPS, i que les dades que ens proporcionava són correctes.

Per comprovar el primer punt, vam crear la classe mabett.devices.gps.GPSTest, que realitzava una crida seqüencial als mètodes oferts per GPSController. Si ens trobàvem en una zona a l'aire lliure, el sistema era capaç de retornar la nostra posició en aproximadament uns 4 minuts [11]. Execucions consecutives de la classe de prova un cop havíem aconseguit recuperar la nostra localització, es completaven en pocs segons.

En proves realitzades dins d'edificis, en canvi, s'exhauria el *timeout* sense haver recuperat cap dada.

Per tal de comprovar el segon punt, vam comparar la posició calculada per l'N810 amb la posició aproximada extreta de Google Maps (ja que no disposàvem de cap altre receptor GPS); coincidien amb un alt grau de precisió:

	N810	**Google Maps**
Latitud	41.563381	41.563854
Longitud	1.996530	1.996508

5.1.2 Mòdul IDBlue/RFID

Les proves del mòdul de lectura de RFID utilitzant l'IDBlue s'han centrat, novament, en os punts: comprovar que es podia establir una connexió amb el dispositiu a distàncies prudencials i que els identificadors recuperats eren correctes.

Per comprovar el primer punt, vam crear una classe de prova, mabett.devices.bt.BTTest, que cridava els mètodes de la classe IDBlueController. Els resultats han estat satisfactoris, ja que hem estat capaços de connectar-nos a un IDBlue situat a tres metres de distància, un valor més que suficient per cobrir les nostres necessitats.

La pròpia lectura de l'RFID, en canvi, necessita que l'IDBlue i l'etiqueta es trobin a no més d'un o dos centímetres. És un fet habitual en les etiquetes amb tecnologia de connexió passiva.

L'únic problema que hem tingut en el procés de connexió ha estat quan la bateria de l'IDBlue es trobava baixa; en aquesta situació, el procés de connexió fallava o s'interrompia al cap d'uns segons. El problema se soluciona carregant de nou el dispositiu.

Per comprovar el segon punt, donat que no disposem de cap altre lector RFID, hem realitzat una prova molt simple: hem comprovat que, una mateixa etiqueta llegida múltiples vegades retorna sempre el mateix identificador. Al seu torn, també hem comprovat que dues etiquetes diferents tenen valors també diferents. Els resultats, en ambdós casos, han estat satisfactoris.

5.1.3 Interfície gràfica

Per tal de provar la interfície gràfica d'una manera aïllada, l'hem iniciat fora de la plataforma JADE utilitzant la classe mabett.Boot. Al seu torn, hem desactivat el suport GPS i Bluetooth i ens hem centrat en comprovar els següents punts:

- L'aplicació és capaç d'iniciar-se sense cap fitxer de preferències en el sistema.
- L'aplicació és capaç d'emmagatzemar les dades introduïdes en la vista de preferències (nom d'usuari, identificador, mètode de triatge escollit, etcètera) i recuperar-les en una nova sessió.
- Totes les dades introduïdes durant un triatge amb mètode START són visibles en la vista manual i corresponen efectivament als valors entrats.
- L'algorisme START se segueix de forma estricta, i l'estat assignat coincideix amb l'estat que indica el protocol manual
- Es pot realitzar un triatge amb el mètode START utilitzant només els dits.

Tots els tests van ser satisfactoris.

5.2 Test d'integració

Un cop comprovat el funcionament dels mòduls per separat, hem realitzat un conjunt de proves destinades a tot el sistema per observar que els mòduls interactuen de manera adequada dins de l'entorn de producció.

S'han comprovat específicament els següents aspectes:

- Un error en qualsevol dels mòduls externs (RFID i GPS) en forma d'excepció, no provoca que l'aplicació deixi de funcionar (per exemple, si no tenim dades GPS, senzillament no seran emplenades; o si no aconseguim llegir l'RFID del pacient, podrem llançar igualment l'agent TriageDataAgent).
- Els agents amb les dades dels pacients són llançats a la plataforma
- Es pugui realitzar un procés de triatge amb el mètode START en un temps inferior als 60 segons.
- L'aplicació respon a totes les peticions de l'usuari de forma ràpida (canviar de vista, llançar l'agent, etc).

L'únic punt a comentar és el relacionat amb la velocitat del sistema. L'aplicació tarda uns 10 segons en obrir-se per primera vegada, i uns 5 segons més en mostrar la vista de triatge per primera vegada. Un cop iniciada l'aplicació, però, la navegació per totes les vistes és fluïda i no s'experimenten més alentiments.

La raó d'aquests alentiments l'hem de buscar a la màquina de Java utilitzada, cacaovm, o en la velocitat de processament de l'N810, perquè no s'experimenta aquest problema executant l'aplicació en un ordinador de sobretaula. En qualsevol cas, són factors on no hi podem intervenir.

Per tant, creiem que novament tots els resultats han estat satisfactoris.

Capítol 6. Conclusions i línies d'ampliació

Hem arribat a l'últim capítol de la memòria i és moment de fer un petit resum de tota la feina feta.

A l'inici del projecte hem plantejat algunes de les limitacions que, des del nostre punt de vista, presenten els mètodes de triatge tradicionals. N'hem escollit un d'ells, START, i ens hem proposat integrar-lo en un sistema informàtic capaç d'aprofitar els avantatges que ofereixen els dispositius mòbils i altres tecnologies com la localització GPS, l'etiquetatge RFID o els sistemes d'agents.

Per desenvolupar aquesta idea, hem exposat els conceptes clau al llarg del capítol 2, que al seu torn ens han permès afrontar l'etapa d'anàlisi amb uns coneixements sobre la matèria suficients. Així, en el capítol 3 hem estat capaços d'enumerar els requeriments funcionals i no funcionals que han començat a donar forma al nostre projecte.

Ha estat també durant la fase d'anàlisi quan hem dividit el sistema en diferents mòduls per facilitar la feina que vindria a posteriori.

D'aquesta manera, en el quart capítol hem dissenyat i implementat un sistema en base als requeriments exposats. Hem començat dissenyant els aspectes més generals del l'aplicació, com la seva arquitectura, i hem acabat implementat tots i cadascun dels mòduls exposats.

En el cinquè capítol hem recollit tota la feina feta, l'hem provada de forma individual, respectant les divisions establertes ja des de la fase d'anàlisi, i finalment també de forma íntegra.

Ha arribat el moment, per tant, d'avaluar la feina feta.

- Hem implementat un assistent per al protocol de triatge START que és capaç de preguntar de forma interactiva les dades que necessita per assignar un estat de triatge a la persona que s'està diagnosticant. L'usuari del sistema no ha de conèixer, per tant, l'algorisme en sí, només s'ha de limitar a contestar les preguntes.
- Hem implementat una segona interfície en la qual un usuari pot entrar de forma manual totes les dades relacionades amb el pacient. Hem inclòs una col·lecció de camps molt àmplia, que no sempre estan presents en les etiquetes de triatge tradicionals. A més, hem inclòs automàticament les dades de localització GPS del pacient si aquestes es troben disponibles.
- Hem incorporat les dues interfícies en una sola aplicació, dissenyada a més per poder suportar altres mètodes de triatge. La usabilitat ha estat un dels objectius que ens hem marcat en el disseny de les interfícies gràfiques, i creiem que l'hem complert creant interfícies estructurades, minimitzant l'entrada de text i incorporant elements de referència visual sempre que ens ha estat possible.
- Hem creat una estructura capaç d'emmagatzemar les dades de cada pacient, independentment del mètode amb el qual hagin estat recollides. No només podem guardar un conjunt de dades, sinó tot un historial de valors que poden anar variant al llarg del temps. Malgrat no ser una funcionalitat que aprofitem en el nostre projecte, tota aquesta informació s'ha incorporat en un sistema intel·ligent com és un agent, facilitant així una futura integració amb altres sistemes mèdics basats en el mateix paradigma.
- Per tal d'identificar de manera única cada pacient, hem incorporat també la tecnologia RFID, tant a la interfície gràfica com a l'estructura de dades.
- Finalment, tot el sistema descrit ha esta dissenyat pensant en l'N810, tenint en compte els avantatges (GPS, pantalla tàctil, etc) i els inconvenients (recursos limitats). Tot i així el producte resultant es pot executar en qualsevol altra plataforma que tingui una màquina virtual de Java. Bona proba d'això és que tot el desenvolupament s'ha realitzat en un ordinador de sobretaula que és capaç d'executar l'aplicació de la mateixa manera en com ho fa l'N810.

Arribats a aquest punt podem concloure que els objectius inicials s'han complert satisfactòriament.

La feina, però, no s'acaba aquí, i a continuació mostrem algunes línies d'investigació que han sorgit durant el desenvolupament del nostre projecte:

1. Creació d'interfícies de protocols de triatge diferents a START. Existeixen situacions on s'han d'aplicar mètodes de triage específics, i el nostre sistema està preparat per soportar-los. Un exemple podria ser implementar el protocol START pediàtric, una versió d'START adaptada a menors, que té en compte símptomes diferents dels adults.
2. Modificar la estructura de dades dels pacients per afegir-hi camps dinàmics sobre els tractaments preventius aplicats, o informació sobre l'estat de radiació i contaminació química del pacient.
3. Tractar totes les dades que han estat recollides: implementar un servidor d'agents capaç d'analitzar i posar en comú la informació de tots equips d'emergència. Una aplicació immediata podria ser, a partir de les dades de localització GPS de tots els pacients i del seu estat, elaborar camins òptims de recollida i atenció.
4. Integrar la nostra aplicació en sistemes mèdics existents.

Bibliografia

(1) W. Sacco, D. Navin, K. Fiedler, R. Waddell, W. Long, and R. Buckman. Precise formulation and evidence-based application of resource-constrained triage. *Acad Emerg Med*, 12:759–770, Aug 2005.

(2) A. Garner, A. Lee, K. Harrison, and C. Schultz. Comparative analysis of multiple-casualty incident triage algorithms. *Ann Emerg Med*, 38:541–548, Nov 2001.

(3) Wikipedia. Global Positioning System [online]. June 2008. Available from: http://en.wikipedia.org/wiki/Global_Positioning_System.

(4) Wikipedia. Radio-frequency identification [online]. June 2008. Available from: http://en.wikipedia.org/wiki/RFID.

(5) Wikipedia. Software agent [online]. June 2008. Available from: http://en.wikipedia.org/wiki/Software_agent.

(6) [M. Benson, K. Koenig, and C. Schultz. Disaster triage: START, then SAVE–a new method of dynamic triage for victims of a catastrophic earthquake. *Prehosp Disaster Med*, 11:117–124, 1996.

(7) B. Spencer. *Application User Interface Design Guide*. Available from: http://www.moblin.org/pdfs/MID_app_design_guide.pdf.

(8) P. M. Vieira-Marques, S. Robles, J. Cucurull, R. J. Cruz-Correia, G. Navarro, and R. Martì. Secure integration of distributed medical data using mobile agents. *IEEE Intelligent Systems*, 21(6):47–54, 2006.

(9) Maemo 4.0 architecture [online]. June 2008. Available from: http://maemo.org/development/documentation/how-tos/4-x/maemo_architecture.html.

(10) Maemo 4.0 api references [online]. June 2008. Available from: http://maemo.org/development/documentation/apis/4-0-x/.

(11) Maemo Bugzilla bug 2878 [online]. June 2008. Available from: https://bugs.maemo.org/show_bug.cgi?id=2878.

(12) Baracoda. *IDBlue. An efficient way to add RFID reader/encoder to bluetooth PDA & mobile phones.*, December 2004. Available from: http://www.baracoda.ch/Bluetooth-Barcodescanner/IDBlue.pdf.

(13) Baracoda. *IDBlue. Programmers Serial Interface Specification*, 2.4.2 edition, March 2006. Available from: http://baracoda.com/baracoda/librairie-doc/ID-Blue-Communication-Protocolo_820.pdf

(14) JSR 82: Java™ APIs for Bluetooth [online]. Available from: http://jcp.org/en/jsr/detail?id=82.

(15) Standard widget toolkit [online]. June 2008. Available from: http://en.wikipedia.org/wiki/Standard_Widget_Toolkit#Design.

(16) Jalimo on Maemo [online]. Available from: https://wiki.evolvis.org/jalimo/index.php/Maemo#Limitations [cited June 2008].

(17) S. Northover and C. MacLeod. *Creating Your Own Widgets using SWT*. Object Technology International, March 2001. Available from: http://www.eclipse.org/articles/Article-Writing%20Your%20Own%20Widget/Writing%20Your%20Own%20Widget.htm.

(18) S. Liang. *The Java™ Native Interface. Programmer's Guide and Specification*. Sun Microsystems, 1999. Available from: http://java.sun.com/docs/books/jni/download/jni.pdf.

Annex 1. Java

Java és el nom amb què es coneix el conjunt de *software* i especificacions desenvolupades per Sun Microsystems que ofereixen un entorn de desenvolupament i implantació multi plataforma. Es tracta d'una tecnologia present en gairebé tot l'espectre de sistemes informàtics, des de plataformes encastades fins a servidors, passant per telèfons mòbils (on és especialment utilitzada) i ordinadors personals.

Plataforma

La plataforma Java és el conjunt de *software* que Sun ofereix per desenvolupar i/o executar programes escrits, principalment, en el llenguatge de programació Java. No és específica de cap arquitectura o sistema operatiu, però necessita que tant el motor d'execució (anomenat màquina virtual) com el compilador (i una sèrie de llibreries estàndard relacionades) sí estiguin disponibles pel sistema. Complerts aquests requeriments, un programa Java s'executarà de forma idèntica en qualsevol sistema.

Existeixen diferents edicions de la plataforma:

- **Java ME** (*Micro Edition*): Especifica diferents *profiles* per dispositius amb limitacions tècniques que impedeixen el suport a la totalitat de les llibreries de Java (per raons de memòria, capacitat de procés, etc). És habitual trobar implementacions de Java ME en telèfons mòbils.
- **Java SE** (*Standard Edition*): Es tracta de la versió de propòsit general que utilitzen tant ordinadors personals com servidors.
- **Java EE** (*Enterprise Edition*): Combina Java SE amb APIs útils per aplicacions client servidor complexes en entorns empresarials.

Llenguatge de programació Java

La paraula Java s'utilitza també per referir-se al llenguatge de programació Java, que va ser presentat conjuntament amb la plataforma l'any 1995. La seva filosofia es pot resumir en 5 punts:

- Presentar una metodologia orientada a objectes
- "*Write Once, Run Anywhere*": permetre que un mateix programa, un cop compilat, sigui executat en múltiples sistemes operatius i arquitectures
- Oferir suport integrat d'accés a xarxes
- Permetre executar codi remot de forma segura
- Integrar conceptes d'altres llenguatges de programació orientat a objectes (especialment C++).

El llenguatge no ofereix elements de baix nivell com punters. Té un model de memòria relativament senzill gestionat de forma automàtica per un *garbage collector*, el que fa innecessari l'existència de mètodes manuals per reservar o alliberar memòria.

Compilador de Java

La seva tasca és compilar codi font en l'anomenat *bytecode*, per ser interpretat més tard a la VM (*Virtual Machine*) de Java. Aquest codi màquina intermedi presenta una semàntica de baix nivell que recorda al codi assemblador. Generalment, els compiladors només processen codi escrit en Java, però n'hi han que accepten altres llenguatges com Python, Ruby o Groovy.

Existeixen també compiladors de Java que compilen el codi de forma estàtica, és a dir, directament a codi nadiu, com ho faria un compilador de C o C++. GCJ (*GNU Compiler for Java*) n'és un exemple. Ofereixen un millor rendiment en comparació amb les aplicacions executades a la màquina virtual. Per altra banda, però, es perd la possibilitat d'executar el mateix binari en cap altre arquitectura, fet que entra en conflicte amb la filosofia de Java.

Màquina virtual de Java (Java VM)

La màquina virtual és considerada el nucli de la plataforma, ja que té la capacitat d'interpretar i executar arxius *bytecode*. Les primeres implementacions tan sols interpretaven *bytecode*, de manera que el rendiment dels programes era inferior a l'obtingut amb altres llenguatges com C o C++. Versions més actuals utilitzen diverses tècniques d'optimització per solucionar el problema.

Una d'aquestes tècniques, l'anomenada compilació JIT (*just-in-time*), permet traduir *bytecode* en codi nadiu en temps d'execució (de la mateixa manera que ho faria un compilador estàtic de Java). Això provoca que, després d'un període inicial, el rendiment de l'aplicació sigui comparable al de programes nadius.

Una altra tècnica, anomenada recompilació dinàmica, és capaç d'analitzar el funcionament de l'aplicació en temps d'execució, i optimitzar aquelles parts del codi que considera crítiques (generalment bucles). Generalment, aquest mètode permet un rendiment superior a l'obtingut amb una compilació estàtica.

Llibreries de classes Java

Degut al fet que Java és independent del sistema operatiu on s'executi, les aplicacions Java no poden dependre de llibreries existents. En comptes d'això, la plataforma oferix un conjunt propi de llibreries estàndard (API) que duen a terme les següents funcions:

- Proveir als programadors d'un conjunt d'estructures de dades, tipus de dades, i mètodes associats típics de qualsevol llenguatge modern, tals com mantenir un llistat d'elements o treballar de forma fluïda amb *strings*.
- Oferir una interfície abstracta a tasques especialment dependents del maquinari i el sistema operatiu, com l'accés a la xarxa (java.net) o al sistema de fitxers (java.io).
- Quan el sistema on corre la plataforma real no suporta algunes característiques de Java, permetre emular-ne el comportament amb els recursos disponibles, o bé oferir un mecanisme consistent per tal de conèixer les limitacions.

JNI

JNI (*Java Native Interface*) és un *framework* que permet que codi Java executant-se en una JVM pugui cridar i ser cridat per aplicacions nadiues (programes específics d'un sistema operatiu i una arquitectura) i llibreries escrites en altres llenguatges, com C i C++.

JNI s'acostuma a utilitzar per accedir a recursos que no ofereixen a les llibreries de classe, o bé per implementar funcions d'alt rendiment utilitzant llenguatges de baix nivell (C o assemblador). Internament, algunes classes de Java depenen d'aquestes funcions (I/O, xarxa o so entra d'altres), però és un fet transparent a l'usuari.

Abans d'utilitzar JNI, haurem de tenir en compte que:

- No és trivial i necessita un esforç considerable per entendre el seu funcionament.

- Errors en la seva utilització poden desestabilitzar tota la JVM i causar problemes molt difícils de reproduir i de depurar.
- Una aplicació que necessiti utilitzar JNI perd la portabilitat que ofereix Java.
- Dins del codi JNI no hi ha recol·lecció de memòria automàtica, amb la qual cosa fa falta gestionar-la de manera manual.

Accés de Java a JNI

En JNI, els mètodes nadius són implementat en forma de funcions en fitxers C o C++. Quan des de Java invoquem aquests mètodes, hi passem de forma automàtica un punter a una interfície per accedir a la JVM, un altre punter que fa referència a l'objecte que ha fet la crida i tots aquells arguments que necessitem.

Per exemple, aquest seria el *template* que hauríem d'utilitzar per implementar un mètode "`methodName`" de la classe "ClassName" del paquet "package.name" al qual li passem un *string*.

```
//C code
JNIEXPORT void JNICALL Java_ package_name_ClassName_methodName
    (JNIEnv *env, jobject obj, jstring javaString) {
    //Get the native string from javaString
    const char *nativeString = (*env)->GetStringUTFChars(env, javaString, 0);

    //Do something with the nativeString

    //DON'T FORGET THIS LINE!!!
    (*env)->ReleaseStringUTFChars(env, javaString, nativeString);
}
```

Per a més informació, consulteu el manual de referència de JNI [18].

Versions

A Abril de 2008, Java 6 és l'última versió estable. Sun ofereix dues versions de cadascuna de les plataformes (SE, ME, EE):

- **Java Runtime Environment** (JRE): conjunt de components format per la màquina virtual i les llibreries. És tot el que necessitem per executar aplicacions Java al nostre sistema.
- **Java Development Kit** (JDK): conté totes les eines necessàries per desenvolupar aplicacions en Java. Està format pel compilador, depurador i altres eines de utilitat diversa, juntament amb una còpia completa del JRE.

Llicència

La plataforma Java de Sun havia estat una plataforma propietària des dels seus inicis. Malgrat part del codi font estava disponible, els termes de la llicència restringien en gran mesura la seva utilització.

L'any 2006, Sun va anunciar que Java esdevindria *software* gratuït i de codi obert. En una primera instància, va alliberar el codi de la seva màquina virtual (HotSpot). A començaments de l'any 2007, Sun va fer el mateix amb gairebé la totalitat de la llibreria de classes (en el que va anomenar *OpenJDK Class Library*). Algunes parts de la plataforma, però, encara no han estat alliberades ja que Sun no en té tots els drets. Actualment, tant Sun com la comunitat *Open Source* està treballant en substituir aquests components per altres de codi obert.

Annex 2. Instal·lació i configuració de l'N810

En aquest annex volem descriure els passos necessaris per configurar des de zero un dispositiu N810 per poder ser utilitzat en el nostre projecte.

Atenció: aquest procés elimina tota la informació de la partició principal de l'N810, procediu amb precaució.

Actualització del *firmware*

Les actualitzacions del sistema operatiu de l'N810 es realitzen actualitzant el *firmware* de l'aparell, que correspon a la partició principal de 256 MB on resideix tot el sistema. Si ve és possible instal·lar i realitzar actualitzacions menors utilitzant el gestor de paquets de maemo, les grans actualitzacions requereixen una actualització del *firmware*.

Descàrrega del *firmware*

En primer lloc he de dirigir-nos a la pàgina que Nokia ofereix per descarregar-nos la última versió del sistema: http://tablets-dev.nokia.com/nokia_N810.php

Se'ns demanarà que introduïm l'adreça MAC de la tarja de xarxa de l'N810 abans de procedir. Un cop introduïda, se'ns mostrarà un llistat amb totes les versions del firmware publicades; escollirem al més actual i la descarregarem (a juny del 2008, l'última versió és *OS 2008 2.2007.51-3*, i ocupa 135 MB).

Descàrrega de la utilitat d'actualització

És necessari disposar d'una aplicació, disponible també des del mateix web http://tablets-dev.nokia.com/d3.php, per *flashejar* l'N810 amb el *firmware* descarregat. Només fa falta que descarreguem la versió que es correspongui amb el nostre sistema operatiu i la desem al mateix directori on hem descarregat el *firmware*.

Actualització

Amb la bateria de l'N810 ben carregada, apagar el dispositiu i connectar-lo al port USB de l'ordinador. A continuació, executar la comanda:

```
$ ./flasher-2.0 -F <<nom de l'arxiu de firmware >> -f --enable-rd-mode -R
```

Quan se'ns mostri el missatge "*Suitable USB device not found, waiting*", encendre l'N810 tot prement el botó de canvi de finestra. Si tot ha anat bé, hauria d'aparèixer un *output* semblant al següent:

```
SW version in image: RX-44_2008SE_2.2007.51-3_PR_MR0
Image 'kernel', size 1529984 bytes
    Version 2.6.21.0-200749osso2
Image 'initfs', size 1954560 bytes
    Version 0.95-70
Image 'rootfs', size 137625600 bytes
    Version RX-34+RX-44_2008SE_2.2007.51-3_PR_MR0
Image '2nd', size 8192 bytes
    Valid for RX-44: 0801, 0802, 0803, 0804, 0805, 0806, 0901, 0902
    Version 1.1.11-1
Image 'xloader', size 9216 bytes
    Valid for RX-44: 0801, 0802, 0803, 0804, 0805, 0806, 0901, 0902
    Version 1.1.11-1
Image 'secondary', size 99968 bytes
    Valid for RX-44: 0801, 0802, 0803, 0804, 0805, 0806, 0901, 0902
    Version 1.1.11-1
Suitable USB device not found, waiting
USB device found found at bus 005, device address 006-0421-0105-02-00
Found device RX-44, hardware revision 0805
NOLO version 1.1.7
Version of 'sw-release': RX-44_2008SE_2.2007.50-2_PR_MR0
Sending xloader image (9 kB)...
100% (9 of 9 kB, avg. 3000 kB/s)
Sending secondary image (97 kB)...
100% (97 of 97 kB, avg. 10847 kB/s)
Flashing bootloader... done.
Sending kernel image (1494 kB)...
100% (1494 of 1494 kB, avg. 12555 kB/s)
Flashing kernel... done.
Sending initfs image (1908 kB)...
100% (1908 of 1908 kB, avg. 12896 kB/s)
Flashing initfs... done.
Sending and flashing rootfs image (134400 kB)...
100% (134400 of 134400 kB, avg. 7033 kB/s)
Finishing flashing... done
The device is now in R&D mode
```

Root

La primera vegada que encenem l'N810 després d'actualitzar el sistema, ens demanarà un conjunt de dades de configuració d'idioma, hora, xarxa, etc.

Com que hem activat el mode R+D en el dispositiu, podrem entrar com a *root* utilitzant la comanda "`sudo gainroot`". No és aconsellable deixar activat aquest mode, així que el primer qua farem serà modificar *l'script gainroot* per poder utilitzar-lo també sense el mode R+D.

Obrirem l'arxiu `/usr/sbin/gainroot` i canviarem la línia:

```
MODE=enabled `/usr/sbin/chroot /mnt/initfs cal-tool --get-rd-mode`
```

per:

```
MODE=enabled
```

Finalment, des del nostre ordinador desactivarem el mode R+D de la mateixa manera que hem actualitzat el *firmware*, però utilitzant la comanda:

```
$ ./flasher-2.0 --disable-rd-mode
```

Instal·lant nous paquets de software

Des de l'N810, podem visitar la pàgina http://gronmayer.com/it per instal·lar automàticament tots aquells repositoris que ens interessin. En el nostre cas, necessitem instal·lar el del projecte Jalimo per instal·lar Java. Ho podem fer a través d'aquest web o visitant la pàgina del projecte: https://wiki.evolvis.org/jalimo/index.php/Maemo.

Jalimo

Podem instal·lar tots els paquets necessaris per tenir un entorn operatiu de Java (cacaovm, GNU classpath i llibreries SWT) amb la comanda:

```
$ apt-get install jalimo-swt-example
```

AvetanaBT

Malauradament no hi ha cap repositori amb una versió compilada d'AvetanaBT. Ens haurem de descarregar les fonts i compilar-les en l'SDK de maemo. La pàgina del projecte és http://sourceforge.net/projects/avetanabt/, i un cop compilat hauríem d'obtenir dos arxius que haurem de copiar al nostre N810:

- avetanaBT.jar: el copiarem a `/usr/share/java/`
- libavetanaBT.so: el copiarem a `/usr/lib/jni/`

JADE

Per instal·lar JADE, simplement haurem de descarregar la versió estàndard compilada del *framework* des de la pàgina web del projecte: http://jade.tilab.com/download.php

Un cop descarregat, necessitarem copiar a l'N810 els següents arxius, preferiblement a un directori de l'estil `/usr/local/jade`:

```
commons-codec-1.3.jar
http.jar
iiop.jar
jade.jar
jadeTools.jar
```

Script d'execució

Per tal d'executar JADE a l'inici del sistema només haurem de crear un *script* a /etc/rc5.d/ que executi la següent comanda:

```
java -cp  /usr/local/jade/commons-codec-1.3.jar: \
          /usr/local/jade/http.jar:               \
          /usr/local/jade/iiop.jar:               \
          /usr/local/jade/jade.jar:               \
          /usr/local/jade/jadeTools.jar           \
          jade.Boot >/var/log/jade.log &
```

Problemes amb missatges de consola incomplets

Els missatges de registre que JADE imprimeix per pantalla apareixen incomplets, hi manca la informació de la classe i el mètode que ha realitzat la petició de *log*.

```
11/06/2008 12:12:18 <unknown> <unknown>
```

No és un problema greu, però l'explicació és la següent (informació extreta de [http://java.sun.com/j2se/1.5.0/docs/guide/logging/overview.html):

"[…] For this second set of methods [void warning(String msg)], the Logging framework will make a "best effort" to determine which class and method called into the logging framework and will add this information into the !LogRecord. However, it is important to realize that this automatically inferred information may only be approximate. The latest generation of virtual machines perform extensive optimizations when JITing and may entirely remove stack frames, making it impossible to reliably locate the calling class and method."

La nostra màquina virtual utilitza, efectivament, la compilació JIT per executar els mètodes de java de forma nadiua. Per tant, aquesta podria ser la causa.

Annex 3. Disseny d'icones per la interfície

En el desenvolupament de l'aplicació hem dissenyat un conjunt nombrós d'icones i imatges amb l'objectiu de facilitar als usuaris la utilització de la nostra interfície. En aquest annex mostrem totes les icones creades, classificades segons la vista on s'han utilitzat.

3.1 Icones dels menús

Notes

General Information

Personal Data

Body injuries

Confirm

Triage Tag

Preferences

3.1.1 Obres originals

- *General Information*: http://commons.wikimedia.org/wiki/Image:Star_of_life.svg
- *Personal Data*: http://commons.wikimedia.org/wiki/Image:Gnome-preferences-desktop-personal.svg
- *Confirm*: http://commons.wikimedia.org/wiki/Image:Artículo_bueno.svg

3.2 Icones del protocol START

Pulse

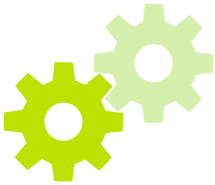
Breath

Walking

Capillar refill

Retry breath

BPM



Mental status

3.3 Icones dels controls

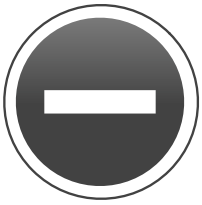
Right

Left

Down

Up

Add



Substract

Firmat: Xavier Jurado Cristóbal.

Bellaterra, juny de 2008

Resum

Els sistemes d'atenció mèdica en situacions de pèrdues humanes massives necessiten classificar un número molt elevat de víctimes en un temps limitat i amb pocs recursos disponibles. Els mètodes clàssics es basen en una etiqueta de triatge de paper on el personal d'emergències anota l'estat del pacient juntament amb una quantitat reduïda d'informació. En aquest projecte descrivim l'anàlisi, disseny i implementació d'interfícies gràfiques per dispositius mòbils pel protocol de triatge START, per ser utilitzades pels equips d'emergència. Els components del projecte inclouen un dispositiu portàtil Nokia N810 amb connexió GPS, i IDBlue, un lector d'etiquetes RFID per Bluetooth.

Resumen

Los sistemas de atención médica en situaciones de pérdidas humanas masivas necesitan clasificar un número muy elevado de víctimas en un tiempo limitado y con pocos recursos disponibles. Los métodos clásicos se basan en una etiqueta de papel donde el personal de emergencias anota el estado del paciente juntamente con una cantidad limitada de información. En este proyecto describimos el análisis, diseño y implementación de varias interfaces gráficas para dispositivos móviles del protocolo de triage START, para ser utilizadas por los equipos de emergencias. Los componentes del proyecto incluyen un dispositivo portátil Nokia N810 con conexión GPS, y IDBlue, un lector de etiquetas RFID por Bluetooth.

Abstract

Medical care at mass casualty incidents requires rapid triage of an overwhelming numbers of victims, in a limited time and with little resources available. Traditional systems rely on a paper triage tag on which rescuers and medical providers mark the patient's triage status and record limited information. In this project, we describe the analysis, design and development of handheld device oriented graphical user interfaces based on START triage method for use by rescuers responding to disasters. The components of this project includes a Nokia N810 mobile device with GPS capabilities and a IDBlue, a Bluetooth based RFID reader.