



FACULTADE DE INFORMÁTICA
DEPARTAMENTO DE ENXEÑERÍA DE COMPUTADORES

PROXECTO FIN DE CARREIRA
ENXEÑERÍA INFORMÁTICA

*Deseño e Implementación dunha Gaita MIDI
Sen Fíos en Tempo Real Empregando
Software/Hardware Libre*

Autor: Alejo Pacín Jul
Director: Santiago José Barro Torres
Director: Tiago Manuel Fernández Caramés

A Coruña, a 5 de outubro do 2018.

OKI

Colaborador Oficial en
Soluciones de Impresión

Esta obra está licenciada coa licenza Creative Commons Atribución-Compartir igual 3.0 España.

Para ver unha copia desta licenza, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> ou envíe unha carta a Creative Commons, CP 1866, Mountain View, CA 94042, EE.UU.



A aqueles que manteñen viva a nosa cultura e, en especial, ós que promoven que dita cultura sexa libre.

Agradecementos

É de ben nacido ser agradecido e son moitos os agradecementos que debería plasmar nesta memoria; pero para ilo faríame falta un anexo completo. Sendo o espazo do que dispoño máis ben reducido, gustaríame aproveitalo para dar as grazas a aquelas persoas más especiais.

- A *Tiago M. Fernández*, por titorizar e dirixir este proxecto, así como pola súa inestimable e nunca ben pagada axuda cando as cousas se torceron.
- A *Santiago J. Barro*, por dirixir este proxecto e animarme a plasmar nel a idea na que levaba máis dunha década cavilando.
- A *Lis Latas*, pola súa axuda á hora de construir o corpo do punteiro que acompaña estas páxinas.
- A *Pepe Vaamonde*, por prestarse coma músico de estudo para gravar as mostras de son que emprega esta gaita.
- A *Óscar Rodríguez Fernández*, por ensinarme praticamente todo aquilo que sei sobre o mundo da gaita galega.
- A *Jacobo Aragunde e Javier Morán*, pola axuda prestada con *LibrePlan*.
- A *Laura M. Castro*, por proporcionar desinteresadamente o seu modelo para a memoria do proxecto feito en *L^AT_EX*.
- A todas aquelas persoas anónimas que se molestaron en perder parte do seu tempo para cubrir a enquisa do estudo de viabilidade que acompaña esta memoria.

A eles e a todos aqueles que se me quedan no tinteiro, os meus máis sinceiros agradecementos.

Resumo

O obxectivo último deste proxecto foi a creación dunha gaita MIDI baseada en software e hardware libre, que tivo coma obxecto a simulación o máis fielmente posible dunha gaita galega, así como a recolleita e realización das esixencias do usuario dado que, a día de hoxe, non existe ningunha gaita MIDI no mercado que cumpra coas mesmas na súa totalidade.

O proxecto comezou analizando as gaitas MIDI xa existentes no mercado, recollendo as opinións tanto de expertos coma dos usuarios xerais, para logo plasmalos nun deseño que, finalmente, nos permitiu implementar o controlador MIDI correspondente.

Para iso fíxose uso da plataforma Arduino (hardware libre modular), empregando tanto placas base (Uno), coma módulos externos (sensores de presión e capacitivos) ou superpostos que se comunican entre si empregando protocolos moi variados segundo as súas necesidades (I2C, UART, USB, MIDI, etc.). Todo este hardware está programado en C/C++, Processing/Wiring e Arduino.

Para o control de dito hardware e do sintetizador faise uso dunha aplicación de configuración gráfica multiplataforma escrita en Java e Swing, que explota toda a potencialidade dos mesmos, facendo que a simulación sexa o máis completa posible.

Palabras clave: Gaita galega; Controlador MIDI; Microprogramación; Redes sen fíos; Tempo real; Hardware libre; Software libre; Arduino; C/C++; Zigbee; Java; JSON.

Índice xeral

	Páxina
1. Introdución	1
1.1. Obxectivo principal	1
1.2. Outros obxectivos	2
1.3. Motivación	2
1.4. Visión xeral do sistema	5
1.5. Métodos e fases do proxecto	7
1.6. Estructura da memoria	8
2. Contextualización	9
2.1. A gaita galega	9
2.1.1. Breve percorrido histórico	10
2.1.2. Constitución	11
2.1.3. O punteiro	12
2.1.4. Os bordóns	13
2.1.5. Dixitación	15
2.1.6. Tonalidade	16
2.2. Estado da arte	17
2.2.1. Protocolos de comunicación de instrumentos musicais	17
2.2.1.1. MIDI	17
2.2.1.2. OSC	26
2.2.1.3. MIDI vs OSC	28
2.2.1.4. Alternativas actuais	29
2.2.2. Gaitas MIDI	30
2.2.2.1. Master Gaita	30
2.2.2.2. Technopipes	32

2.2.2.3. ePipe	33
2.2.2.4. Hevia Electronic Bagpipe	35
2.2.2.5. OpenPipe	36
3. Planificación	39
3.1. Ciclo de vida	39
3.2. Descripción das tarefas	43
3.3. Xestión do proxecto	48
3.4. Planificación inicial	49
4. Análise de viabilidade	53
4.1. Planificación inicial	54
4.2. Determinación	54
4.2.1. Obxectivos	54
4.2.2. Alternativas	55
4.2.3. Restriccóns	57
4.3. Avaliación de alternativas e resolución de riscos	57
4.3.1. Análise de riscos	57
4.3.2. Prototipo 1	61
4.4. Desenvolvemento e validación do seguinte nivel do producto	63
4.4.1. Concepto de operación	63
4.5. Planificación da seguinte fase ou ciclo	64
4.5.1. Planificación de requisitos	64
4.5.2. Planificación do ciclo de vida	64
4.6. Estudio de viabilidade	64
5. Análise de requisitos	75
5.1. Determinación	76
5.1.1. Obxectivos	76
5.1.2. Alternativas	76
5.1.3. Restriccóns	76
5.2. Avaliación de alternativas e resolución de riscos	77
5.2.1. Análise de riscos	77
5.2.2. Prototipo 2	78
5.3. Desenvolvemento e validación do seguinte nivel do producto	78
5.3.1. Simulacións, modelos e programas de proba	78
5.3.2. Requisitos hardware, software e económicos	81

5.3.2.1. Extracción de requisitos por parte do proxectando	81
5.3.2.2. Extracción de requisitos por parte de expertos	86
5.3.2.3. Extracción de requisitos por parte dos usuarios	90
5.3.3. Validación de requisitos	99
5.4. Planificación da próxima fase ou ciclo	104
5.4.1. Planificación do deseño	104
6. Deseño do sistema	105
6.1. Determinación	106
6.1.1. Obxectivos	106
6.1.2. Alternativas	106
6.1.3. Restriccóns	106
6.2. Avaliación de alternativas e resolución de riscos	107
6.2.1. Análise de riscos	107
6.2.2. Prototipo 3	107
6.2.2.1. Prototipo hardware	108
6.2.2.2. Prototipo software	127
6.3. Desenvolvemento e validación do seguinte nivel do producto	130
6.3.1. Simulacións, modelos e programas de proba	130
6.3.2. Deseño do producto hardware e software	130
6.3.2.1. Deseño do producto hardware	130
6.3.2.2. Deseño do producto software	130
6.3.3. Verificación e validación do deseño	131
6.4. Planificación da próxima fase ou ciclo	133
6.4.1. Planificación do desenvolvemento	133
7. Desenvolvemento do sistema	135
7.1. Determinación	136
7.1.1. Obxectivos	136
7.1.2. Alternativas	136
7.1.3. Restriccóns	136
7.2. Avaliación de alternativas e resolución de riscos	137
7.2.1. Análise de riscos	137
7.2.2. Prototipo operacional	137
7.2.2.1. Prototipo hardware	138
7.2.2.2. Prototipo software	151
7.3. Desenvolvemento e validación do seguinte nivel do producto	168

7.3.1.	Simulacións, modelos e programas de proba	168
7.3.2.	Deseño detallado	169
7.3.2.1.	Deseño hardware	169
7.3.2.2.	Deseño software	169
7.3.3.	Ensamblado e codificación	174
7.3.3.1.	Ensamblado	174
7.3.3.2.	Codificación	174
7.3.4.	Probas de unidade	193
7.3.4.1.	Hardware	193
7.3.4.2.	Software	195
7.3.5.	Integración e probas	202
7.3.6.	Probas de aceptación	205
7.3.7.	Documentación	210
8. Conclusións		215
8.1.	Incidencias durante o desenvolvemento do proxecto	215
8.1.1.	Planificación	215
8.1.1.1.	Xestión do proxecto	215
8.1.2.	Deseño do sistema	218
8.1.2.1.	Prototipo 3	218
8.1.3.	Implementación	221
8.1.3.1.	Ensamblado e codificación	221
8.1.3.2.	Integración e probas	224
8.1.3.3.	Implantación	226
8.2.	Conclusións finais	227
8.3.	Traballo futuro	229
A. Planificación completa		233
A.1.	Planificación inicial	233
A.2.	Planificación final	233
B. Enquisa		245
B.1.	Resultados	245
C. Arduino		255
C.1.	Historia	256
C.2.	Instalación	256
C.2.1.	Windows	256

C.2.2. GNU/Linux	258
C.3. Hardware	259
C.3.1. Placas Arduino	259
C.3.2. Linguaxe de programación Arduino	260
C.3.2.1. Funcións básicas e operadores	262
C.4. Bibliotecas en Arduino	271
C.4.1. Creación de bibliotecas	273
D. XBee	277
D.1. Formatos, antenas e modos de datos	279
D.2. XBee ZB	279
D.3. ZigBee	279
D.3.1. Visión xeral	280
D.3.2. Historia	281
D.3.2.1. ZigBee 2004	282
D.3.2.2. ZigBee 2006	282
D.3.2.3. ZigBee PRO	282
D.3.3. Usos	283
D.4. Estándar e perfís	284
D.4.1. Licencia	284
D.4.2. Perfís de aplicación	284
D.5. Hardware de comunicacíóns RF	285
D.6. Tipos de dispositivos e modos de operación	286
D.7. Software	287
D.7.1. Capa de rede	287
D.7.2. Capa de aplicación	289
D.7.3. Compoñentes principais	289
D.7.4. Modelos de comunicación	289
D.7.5. Comunicación e detección de dispositivos	290
D.8. Servizos de seguridade	292
D.8.1. Modelo de seguridade básico	292
D.8.2. Arquitectura de seguridade	293
E. Fritzing	295
E.1. Obxectivos	295
E.2. Interface de usuario	296
E.2.1. Seccións	296

E.2.2. Vistas	296
F. Glosario de acrónimos	299
G. Glosario de termos	301
Bibliografía	303

Índice de figuras

Figura	Páxina
1.1. Deseño conceptual do sistema.	6
2.1. Clarinetes/óboes duplos nas Cantigas de Santa María	10
2.2. Gaita galega nas Cantigas de Santa María	11
2.3. Partes dunha gaita galega	11
2.4. Punteiro	12
2.5. Palleta	13
2.6. Ronco	13
2.7. Pallón	14
2.8. Ronqueta	14
2.9. Chillón	15
2.10. Dixitación aberta	15
2.11. Dixitación pechada	16
2.12. Correspondencia entre tonalidades	17
2.13. Conector MIDI	20
2.14. Portos e cable MIDI	21
2.15. Master Gaita	30
2.16. Technopipes	32
2.17. ePipe	34
2.18. Hevia MBS 300	35
2.19. OpenPipe	36
3.1. Ciclo de vida en espiral	40
3.2. Planificación inicial (1/2)	50
3.3. Planificación inicial (2/2)	51

4.1. Planificación inicial do ciclo de análisis de viabilidade.	54
4.2. Prototipo 1.	62
4.3. Concepto de operación.	63
4.4. Planificación inicial do ciclo de análisis de requisitos.	65
4.5. Resultados do estudio de viabilidade (p. 1).	70
4.6. Resultados do estudio de viabilidade (p. 2).	71
4.7. Resultados do estudio de viabilidade (p. 3).	72
4.8. Resultados do estudio de viabilidade (p. 8).	72
5.1. Prototipo 2: pantalla de inicio.	79
5.2. Prototipo 2: pantalla de selección.	79
5.3. Prototipo 2: pantalla de afinación.	80
5.4. Prototipo 2: pantalla de sensibilidad.	80
5.5. Prototipo 2: pantalla de dicitación.	81
5.6. Lis Latas.	86
5.7. Pepe Vaamonde.	89
5.8. Resultados da extracción de requisitos por parte dos usuarios (p. 4).	92
5.9. Resultados da extracción de requisitos por parte dos usuarios (p. 5).	93
5.10. Resultados da extracción de requisitos por parte dos usuarios (p. 6).	94
5.11. Resultados da extracción de requisitos por parte dos usuarios (p. 7).	95
5.12. Resultados da extracción de requisitos por parte dos usuarios (p. 8).	96
5.13. Planificación inicial do ciclo de diseño.	104
6.1. Arduino.	110
6.2. BeagleBone.	110
6.3. PandaBoard.	111
6.4. Atmel QTouch.	111
6.5. Raspberry Pi.	112
6.6. BMP085.	114
6.7. MPR121.	115
6.8. Arduino Fio.	116
6.9. XBee ZB.	117
6.10. MIDI Breakout.	118
6.11. XBee Explorer USB.	119
6.12. Arduino.	119
6.13. Moco.	119
6.14. Arduino Wireless SD Shield.	120

6.15. Arduino Wireless Proto Shield.	120
6.16. MicroDrive G1.	122
6.17. Batería LiPo 850 mAh con conector JST.	123
6.18. Prototipo 3 hardware, emisor: vista real.	125
6.19. Prototipo 3 hardware, receptor: vista real.	125
6.20. Prototipo 3 hardware: vista esquemática.	126
6.21. Prototipo 3: pantalla de inicio.	127
6.22. Prototipo 3: pantalla de selección.	128
6.23. Prototipo 3: pantalla de afinación.	128
6.24. Prototipo 3: pantalla de sensibilidad.	129
6.25. Prototipo 3: pantalla de dixitación.	129
6.26. Deseño de alto nivel.	132
6.27. Planificación inicial do ciclo de desenvolvemento.	133
 7.1. Router.	139
7.2. Receptor.	140
7.3. Sensor de presión.	142
7.4. Interface do sensor de presión.	143
7.5. Ficheiro de proba do sensor de presión.	144
7.6. Sensores capacitivos.	145
7.7. Interface dos sensores capacitivos.	146
7.8. Ficheiro de proba dos sensores capacitivos.	147
7.9. Lector de tarxetas.	148
7.10. Interface do lector de tarxetas.	149
7.11. Ficheiro de proba do lector de tarxetas.	150
7.12. Deseño de nivel intermedio.	152
7.13. Servizo de dispositivos.	154
7.14. Servizo de dispositivos.	155
7.15. Servizo de dispositivos.	156
7.16. Servizo de dispositivos.	157
7.17. Servizo de configuración.	158
7.18. Servizo de configuración.	159
7.19. Servizo de configuración.	160
7.20. Servizo de configuración.	161
7.21. Servizo de configuración.	162
7.22. Servizo de configuración.	163
7.23. Servizo de configuración.	164

7.24. Dispositivo	165
7.25. Configuración.	165
7.26. Configuración de selección.	166
7.27. Servidor MIDI.	167
7.28. Configuración do servidor MIDI.	168
7.29. Deseño detallado.	170
7.30. Servizo de internacionalización.	171
7.31. Servizo de notificacións.	171
7.32. Servizo de navegación web.	172
7.33. Servizo MIDI.	173
7.34. Pseudo-código do sensor de presión.	176
7.35. Medición dos sensores capacitivos.	177
7.36. Configuración da sensibilidade do punteiro.	178
7.37. Matriz de dixitación aberta.	182
7.38. Matriz de dixitación pechada.	183
7.39. Ficheiro de configuración en formato JSON.	184
7.40. Detección de dispositivos.	187
7.41. Recuperación da configuración dun dispositivo.	187
7.42. Envío dunha nova configuración a un dispositivo.	187
7.43. Mapeo da configuración do servidor MIDI.	189
7.44. Notificacións.	189
7.45. Servizo de navegación web.	190
7.46. Servizo MIDI: sistemas UNIX.	191
7.47. Relacións afinación xusta.	193
7.48. Probas de unidade do sensor de presión.	194
7.49. Resultado das probas de unidade do sensor de presión.	196
7.50. Probas de unidade dos sensores capacitivos.	197
7.51. Resultado das probas de unidade dos sensores capacitivos.	198
7.52. Ficheiro de proba do lector de tarxetas.	199
7.53. Ficheiro de proba do lector de tarxetas.	200
7.54. Tests unitarios.	201
7.55. Exemplo de test unitario con Mockito.	201
7.56. Resultados tests unitarios.	202
7.57. Tests de integración.	203
7.58. Exemplo de test de integración.	203
7.59. Resultados tests integracion.	204

7.60. Tests de integración da conectividade	205
7.61. Tests de integración da conectividade	206
7.62. Resultados dos tests de integración da conectividade.	207
7.63. Tests de aceptación.	208
7.64. Exemplo de test de aceptación.	209
7.65. Resultados tests aceptación.	209
7.66. Botón de busca.	211
7.67. Documentación da API da aplicación de configuración.	213
7.68. Menú de axuda da aplicación.	214
8.1. XBee Explorer USB.	218
8.2. Moco.	220
8.3. Consumo de memoria.	222
8.4. Consumo de memoria da versión simplificada.	225
A.1. Planificación inicial (p. 1).	234
A.2. Planificación inicial (p. 2).	235
A.3. Planificación inicial (p. 3).	236
A.4. Planificación inicial (p. 4).	237
A.5. Planificación inicial (p. 5).	238
A.6. Planificación inicial (p. 6).	239
A.7. Planificación inicial (p. 7).	240
A.8. Planificación inicial (p. 8).	241
A.9. Planificación final (p. 1).	243
A.10. Planificación final (p. 2).	244
B.1. Resultados da enquisa (p. 1).	246
B.2. Resultados da enquisa (p. 2).	247
B.3. Resultados da enquisa (p. 3).	248
B.4. Resultados da enquisa (p. 4).	249
B.5. Resultados da enquisa (p. 5).	250
B.6. Resultados da enquisa (p. 6).	251
B.7. Resultados da enquisa (p. 7).	252
B.8. Resultados da enquisa (p. 8).	253
B.9. Resultados da enquisa (p. 9).	254
C.1. IDE Arduino.	257

D.1. Familia XBee [1].	278
D.2. XBee ZB.	280
D.3. ZigBee: pila do protocolo	288
E.1. Fritzing: vista de prototipado.	296
E.2. Fritzing: vista esquemática.	297
E.3. Fritzing: vista de PCB.	297

Índice de cadros

Táboa	Páxina
2.1. Mensaxes MIDI.	24
2.2. Modos MIDI.	26
8.1. Resumo do proxecto.	228

Capítulo 1

Introducción

Índice xeral

1.1. Obxectivo principal	1
1.2. Outros obxectivos	2
1.3. Motivación	2
1.4. Visión xeral do sistema	5
1.5. Métodos e fases do proxecto	7
1.6. Estructura da memoria	8

1.1. Obxectivo principal

○ Proxecto que se expón nesta memoria leva por título “**Deseño e implementación dunha gaita MIDI sen fíos en tempo real empregando software/hardware libre**”.

O propio título reflexa bastante ben o que se buscaba con este proxecto. Avaliar, analizar, deseñar e implementar de maneira teórico-práctica un controlador MIDI que simule o máis fielmente posible unha gaita galega cumplindo cunha serie de requisitos e/ou restriccións:

- Que non empregue fíos.
- Que reproduza son en tempo real.
- E que empregue exclusivamente hardware e software libre.

1.2. Outros obxectivos

Outros obxectivos secundarios pero non menos importantes que se plantexaron durante a realización deste proxecto foron os seguintes:

- Realizar un estudo de viabilidade que amosara o que realmente demanda o mercado.
- Estudar a fondo as tecnoloxías implicadas no proxecto e determinar se son aplicables ó mesmo.
- Demostrar que o uso de hardware/software libre é viable para este tipo de proxectos.

1.3. Motivación

Son moitos e moi variados os motivos que levaron á realización do presente proxecto, a maioría dos cales están presentes no título do mesmo.

O primeiro dos motivos que levou á súa realización foi a idea do mesmo como tal, madurada polo proxentando durante máis dunha década, froito da mistura de dúas das súas paixóns: a música e a informática. Pero dita idea non sería levada a cabo coma Proxecto Fin de Carreira sen o pulo e a motivación por parte dos directores do mesmo, que alentaron ó proxectando a levar dita idea adiante como tal.

Outro dos motivos que impulsaron o proxecto foi a continua constatación ó longo dos anos de que os diferentes productos comerciais que ían saíndo ó mercado non cumplían coas expectativas e requisitos que agardaban os profesionais da materia. Boa parte desas expectativas e requisitos moitas veces solicitados, a maioría dos cales eran obvios, non acabaron de chegar nunca, ben por falta de tecnoloxía, ben por falta de coñecementos dos propios creadores dos productos.

O continuo fracaso dos mesmo a nivel práctico (que non comercial, onde ti-
veron unha acollida relativamente alta), xunto coa recente aparición de novas

tecnoloxías que a priori se poderían aplicar ó producto final, máis os coñecementos adquiridos polo proxectando ó longo dos últimos anos, motivaron tamén a realización deste proxecto.

Parte deses requisitos e expectativas foron reflexados directamente no título deste proxecto: a ausencia de fíos e a xeración de son en tempo real.

Actualmente non existe ningún producto comercial deste tipo que non empriegue fíos. Probablemente por falta de tecnoloxía no seu momento que garantizase a autonomía necesaria e a ausencia de retardos. A principal vantaxe de non empregar fíos nunha gaita MIDI é a liberdade de movementos, moi útil por exemplo, enriba dun escenario, onde prima o dinamismo.

O seguinte dos motivos xorde da necesidade de cubrir certas áreas do día a día no que as gaitas tradicionais frouxean máis ou menos e que as gaitas MIDI poden chegar a cubrir aportando certas vantaxes. Entre elas podemos destacar:

- Ensaios no domicilio particular.
 - A día de hoxe, ensaiar no domicilio particular en lugares como as cidades é pouco menos ca imposible. Se a iso sumamos a elevada sonoridade da gaita, podemos dar por seguras as queixas veciñais e, de insistir, incluso algunha denuncia por superar os decibelios permitidos.
 - Isto cunha gaita MIDI non pasa, dado que se pode regular o volume ou empregar uns cascos para a saída de audio.
- Transcripción de melodías.
 - Para facer a transcripción dunha melodía para gaita, se só contamos cunha gaita tradicional e un ordenador, a transcripción hai que facela praticamente a man.
 - Tendo a man un controlador MIDI pódese conectar ó ordenador e xerar a partitura mentres tocamos. Se ese mesmo controlador é unha gaita MIDI, aforramos ter que aprender a tocar outro instrumento (coma pode ser un teclado) tan só para pasar as partituras. Cunha gaita MIDI, a transcripción é totalmente natural.

- Afinación en condicións desfavorables.
 - Para profesionais que adoitan tocar enriba dun escenario en calquera época do ano, conseguir afinar correctamente o instrumento pode chegar a ser imposible dependendo das condicións de temperatura e humidade principalmente.
 - Neste suposto, a gaita MIDI tamén sae gañando, dado que ditos parámetros non lle afectan.
- Interpretación en varias tonalidades.
 - Hai determinadas pezas nas que o autor recomenda tocar nunha tonalidade determinada. Un gaiteiro cun repertorio amplio, coñecerá probablemente varias pezas con este tipo de restriccións en distinta tonalidade. Se ten que interpretar áinda que só sexan un par delas en distinta tonalidade durante unha actuación, implicará que ou ben, ten máis dunha gaita (maior custe) ou ben precisará andar cambiando de punteiro e de bordóns entre peza e polo tanto volver a afinar (ademas do tempo que se perde -inasumible-).
 - Cunha gaita MIDI o cambio de tonalidade é instantáneo.
- Interacción automatizada con outros dispositivos MIDI.
 - O protocolo MIDI permite enviar mensaxes de sincronización e de disparo de eventos, o que á súa vez permite interactuar de maneira automatizada con otros dispositivos MIDI que, por exemplo, permitirían que un só músico tocando un único controlador fixese soar á vez varios instrumentos pre-programados e parecer un grupo e non un solista.

E por último, unha motivación autoimposta: a de facelo empregando exclusivamente (sempre na medida do posible e do razonable) hardware e software libre.

A día de hoxe tanto o mundo do hardware coma do software libre é un mundo maduro que pode competir ó mesmo nivel que o privativo e levar a cabo un proxecto coma este, que toca varias ramas da Enxeñería Informática, era unha boa ocasión para intentar demostrarlo.

Ademais, fomentar o coñecemento e a cultura libre debería ser o habitual no ámbito da educación, no que se engloba o ensino universitario. Compartir o coñecemento fomenta a colaboración e o pensamento crítico e, no tocante a este proxecto, pode xerar unha comunidade arredor do mesmo.

E xerar unha comunidade de colaboradores debería redundar na mellora do proxecto a longo prazo, do que se poderían beneficiar todos os interesados.

Se o levamos ó terreo económico, redunda na mellora dun producto comercial, que pode incrementar as vendas e ó mesmo tempo mellorar a satisfacción dos clientes.

Se o levamos ó terreo cultural, contribúe á conservación da cultura e tradición galegas. E iso é beneficio para todos.

Segundo estes principios, o contido deste proxecto atópase dispoñible baixo a licencia GPLv3 (e CC-BY-SA para contidos non licenciables baixo GPLv3) no enderezo <http://proxecto-puding.org>. Así mesmo, co fin de xerar comunidade arredor do proxecto, existen unha serie de contas de promoción en redes sociais e unha rolda de correo accesibles desde o mesmo enderezo.

1.4. Visión xeral do sistema

Aínda que máis adiante se explicará en detalle, convén dar unha visión xeral do sistema para comprender os seguintes capítulos.

Esquematicamente falando, o deseño conceptual do sistema ó que se pretendía chegar era o da figura 1.1. Nela podemos observar un sistema composto por catro elementos ou actores, sendo un deles o propio usuario.

Dos tres restantes, o situado á esquerda corresponde a la parte hardware e os situados á dereita, a la parte software.

O usuario será o encargado de tocar a gaita (ou controlador) MIDI, que en-

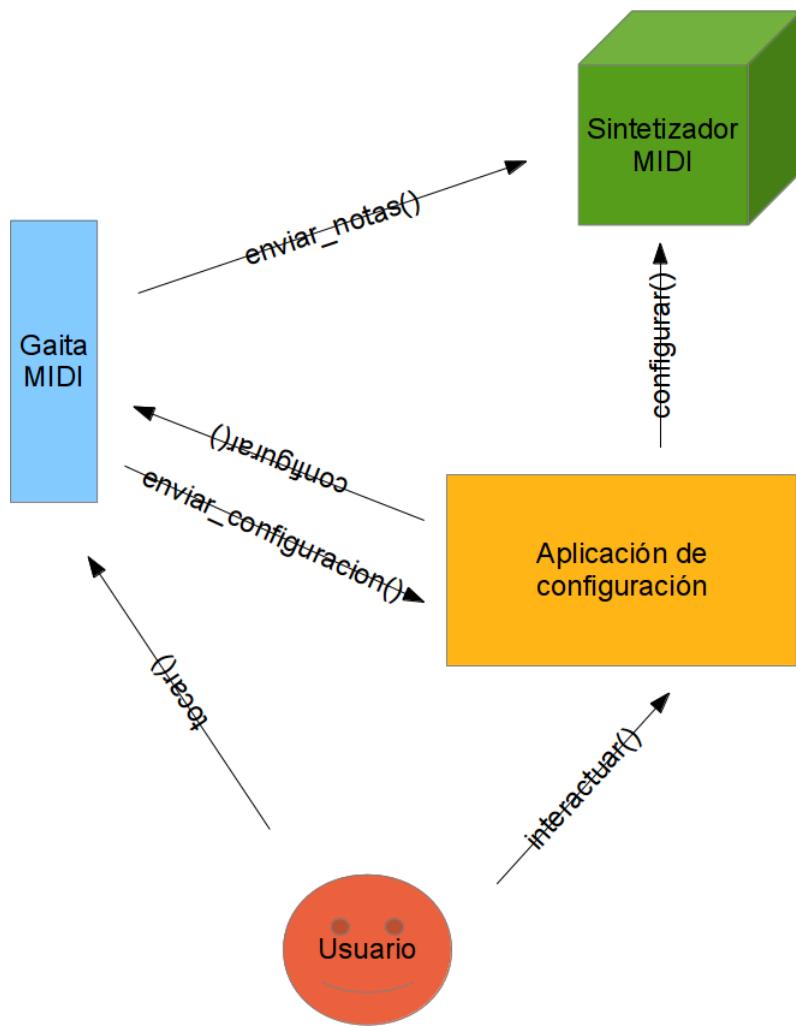


Figura 1.1: Deseño conceptual do sistema.

viará a melodía ou conxunto de notas ó sintetizador MIDI, que se encargará de reproducilas.

Así mesmo, poderá modificar a configuración de calquera dos dous elementos (dependendo dos parámetros modificados) a través dunha interface gráfica, a aplicación de configuración.

Cando a aplicación de configuración detecte algúin controlador MIDI conectado encargarase de rexistralo e amosar a configuración persoal que este lle envíe, información que será vista polo usuario e que logo poderá modificar a vontade.

1.5. Métodos e fases do proxecto

Tendo claro hacia onde ía encamiñado o proxecto, procedeuse a acometer o mesmo en diferentes fases.

Nunha primeira fase analizouse a viabilidade do proxecto, incluíndo un estudo da viabilidade do mesmo, que abranguiu dende o estudo das solucións xa existentes ata a realización dunha enquisa entre os posibles usuarios, pasando por un primeiro prototipo.

Nunha segunda fase analizarónse os requisitos, facendo uso dun estudo das solucións xa existentes, dun segundo prototipo aparentemente funcional e da enquisa previa para a extracción de requisitos por parte do proxectando, por parte de expertos e por parte de usuarios anónimos a través de simulacións, modelos e programas de proba.

Nunha terceira fase acometeuse o deseño. Previo estudo das tecnoloxías a empregar a priori (Arduino, MIDI, etc.), fíxose un terceiro prototipo (hardware e software) a partir do que se obtivo un primeiro deseño.

Nunha cuarta e última fase acometeuse o desenvolvemento. Unha vez obtido o deseño inicial, obtívose un prototipo funcional sobre o que realizar máis probas e que deu lugar a un deseño más detallado, que se empregou para a construción

e implementación do producto final, que foi probado e refinado mediante as pertinentes probas de unidade e integración.

En todas as fases houbo tamén un apartado de determinación de obxectivos, avaliación de alternativas e resolución de riscos, coa fin de asentar as bases de cada unha e de prever posibles problemas.

Finalizado todo este proceso obtívose o producto final, así coma unha serie de conclusións que levarán a obter diferentes melloras futuras do mesmo.

1.6. Estructura da memoria

A división e transcurso do proxecto seguiu as liñas mestras que se expoñen na presente memoria, articulada en dous grandes bloques:

1. *Contextualización e traballos previos.* Neste apartado inclúese unha breve pincelada histórica da gaita galega, a súa composición, os principais protocolos que se empregaron no proxecto e o estado da arte actual.
2. *Desenvolvemento do caso de estudo.*
 - a) Análise da viabilidade.
 - b) Análise dos requisitos.
 - c) Deseño do sistema.
 - d) Desenvolvemento do sistema.

Inclúense tamén varios anexos con información complementaria relativa ó proxecto que, por extensión, son difícilmente intercalables nos apartados previos.

Capítulo 2

Contextualización

Índice xeral

2.1. A gaita galega	9
2.1.1. Breve percorrido histórico	10
2.1.2. Constitución	11
2.1.3. O punteiro	12
2.1.4. Os bordóns	13
2.1.5. Dixitación	15
2.1.6. Tonalidade	16
2.2. Estado da arte	17
2.2.1. Protocolos de comunicación de instrumentos musicais	17
2.2.2. Gaitas MIDI	30

ANTES de entrar de cheo no corpo do proxecto, é preciso contextualizar o mesmo desde varios puntos de vista (histórico, técnico, estado da arte), coa finalidade de sentar as bases para a súa correcta comprensión a posteriori.

2.1. A gaita galega

A gaita galega é un instrumento de vento madeira propio de Galicia, norte de Portugal, occidente de Asturias e o Bierzo. É o símbolo por excelencia da música tradicional galega [2].

2.1.1. Breve percorrido histórico

A gaita de fol xorde hai aproximadamente 2000 anos cando alguén engadiu un fol, é dicir, un depósito flexible para o aire apertado debaixo do brazo, aos dous tubos sonoros que antes se tocaban directamente coa boca (clarinetes e óboes duplos), como se pode apreciar na figura -2.1-; así xurdiron as gaitas de fol.



Figura 2.1: Representación de clarinetes/óboes duplos nas Cantigas de Santa María (s. XIII) [3].

Antes do século XII carecemos praticamente de información iconográfica ou escrita que nos permita saber que aconteceu coa gaita de fol desde a invención do fol. Non entanto, a partir deste momento comeza a ser abundante a información, sobre todo a iconográfica, o que indica que a gaita de fol foi un importante instrumento musical, empregado tanto polas clases sociais superiores como polo pobo [4].

As primeiras representacións gráficas da gaita galega son de mediados do século XIII. Contaba entón de fol, punteiro e soprete, sen roncón (véxase o seguinte apartado). Coincidindo co desenvolvemento da polifonía, engadíronse os bordóns e na segunda metade do XIII e no XIV aparece xa cun roncón, con dous, ou mesmo sen ningún (figura 2.2). Nun principio, este tiña forma abucinada e posteriormente tomou a forma de copa.

Na segunda metade do século XIX introduciuse un chillón como apoio ó punteiro, colocado a carón deste. Ben entrado o século XX apareceu unha ronqueta, máis pequena que o segundo bordón dos séculos anteriores [2].



Figura 2.2: Representación dunha gaita galega nas Cantigas de Santa María (s. XIII) [2].

2.1.2. Constitución

A gaita galega actual (como a amosada na figura 2.3) compонse ou pode constar de [2]:

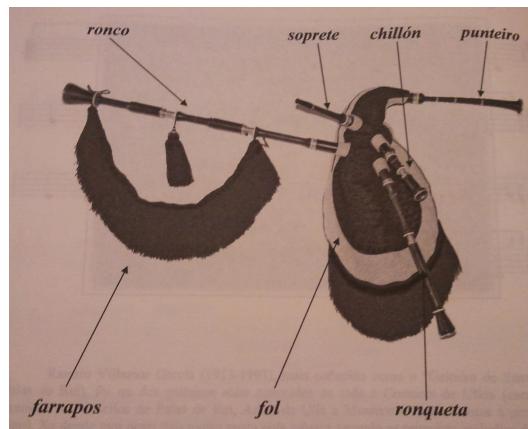


Figura 2.3: Partes dunha gaita galega [5].

- Un **soprete**, co que se introduce o aire no fol.
- Un **fol**, onde se almacena o aire que fai que soe o instrumento.
- Un **punteiro**, o tubo cónico que sae polo frontal do fol e que é o que produce a melodía principal en base á colocación dos dedos.

- Un **ronco** ou **roncón**, o tubo que se apoia sobre o ombro do gaiteiro, que soa na mesma tonalidade que o punteiro, pero dúas oitavas más grave.
- Unha **ronqueta**, o tubo que se apoia sobre o antebrazo derecho do gaiteiro, que soa na mesma tonalidade que o punteiro, pero unha oitava más grave.
- Un **chillón** ou **ronquillo**, que sae do fol paralelo á ronqueta, que soa na dominante da tonalidade do punteiro e na mesma oitava.

2.1.3. O punteiro

O punteiro é a parte más importante e complexa da gaita galega 2.4.

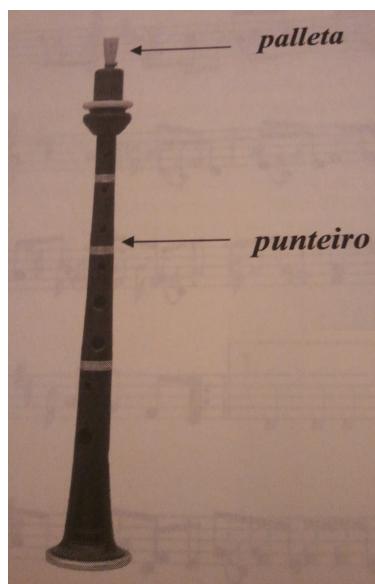


Figura 2.4: Punteiro [5].

É o tubo sonoro no que facemos a melodía. É un tubo cónico de madeira onde se fan once (ou doce) buratos, dos cales tres permanecen sempre abertos (óídos do punteiro que serven para mellorar a afinación) e os restantes poden ficar abertos ou pechados polas xemas dos dedos segundo sexa necesario para acadar os diferentes sons e así poder crear unha melodía.

A **palleta** (figura 2.5) é o elemento que produce o son. Consiste en dúas laminiñas moi finas de cana de bambú que se colocan sobre dun tudel metálico

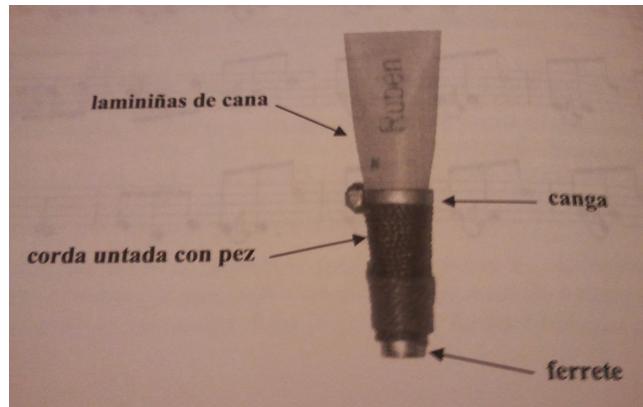


Figura 2.5: Palleta [5].

que recibe o nome de ferrete (normalmente de latón ou cobre) que se amarran ó mesmo empregando corda untada con pez e un ariño metálico (normalmente arame de latón) coñecido co nome de canga. A calidade e singularidade do son do punteiro vese condicionada en gran medida pola calidade da palleta [5].

2.1.4. Os bordóns

Coñécense coma **bordóns** o conxunto de tubos sonoros que producen notas pedal¹. No caso da gaita galega, o ronco, a ronqueta e o chillón.

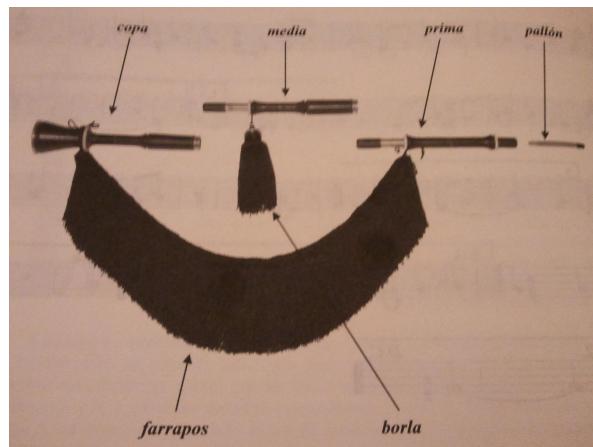


Figura 2.6: Ronco [5].

¹Na música tonal, unha nota pedal é unha nota sostida ou continua que se mantén mentres o resto da melodía segue avanzando.

O **ronco** (figura 2.6) é un tubo sonoro construído en madeira que se fai en tres partes (prima, media e copa) e que produce unha nota pedal. Está feito en tres partes para poder xogar coa súa lonxitude e así poder afinar co punteiro, sempre canto máis alonguemos o ronco máis grave será o seu son e canto máis o acurtemos máis agudo. O ronco afínase sobre a tónica do punteiro, e o seu son é dúas oitavas máis grave que o son deste.

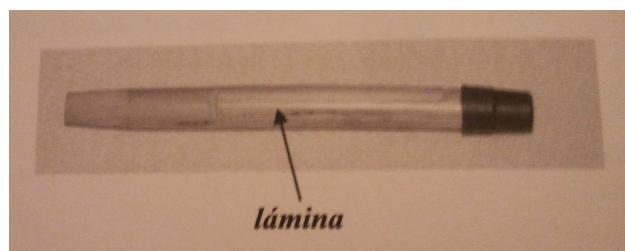


Figura 2.7: Pallón [5].

O son é producido polo **pallón** (2.7), que é un trociño de cana de bambú pechado por un dos seus extremos sobre o que se fai un corte oblicuo de non moita extensión.

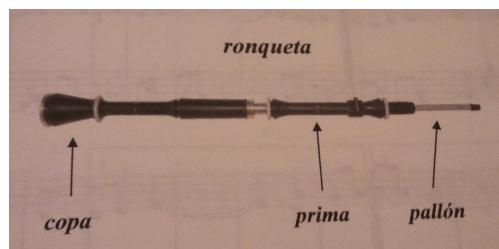


Figura 2.8: Ronqueta [5].

A **ronqueta** (figura 2.8) é un tubo sonoro de madeira construído en dúas pezas (prima e copa). O son prodúceo un pallón de dimensíons más reducidas cas do pallón que utiliza o ronco. Afínase na tónica do punteiro e o seu son é unha oitava máis grave que o son do punteiro. Ó igual que o ronco mantén unha nota pedal.

O **chillón** (figura 2.9) é un tubo sonoro de madeira construído en dúas pezas só que de tamaño máis pequeno que as da ronqueta. O son pode estar producido



Figura 2.9: Chillón [5].

por un pallón pequenío ou por unha pallete (igual á que se emprega no punteiro), estes últimos son os verdadeiros chillóns aínda que na actualidade están quedando en desuso xa que a súa forte sonoridade tapa a melodía producida polo punteiro. Os chillóns que empregan pallón afínanse na dominante do punteiro e os que empregan pallete afínanse na tónica, estando ambos na oitava deste [5].

2.1.5. Dixitación

Defíñese como **dixitación** da gaita galega a maneira concreta de colocación dos dedos sobre o punteiro. Tradicionalmente falando, existen dous modos de dixitación: *aberto* e *pechado*.

No modo *aberto* (figura 2.10) vanse erguendo os dedos ata a nota que se deseja facer soar. Este é o modo máis extendido a día de hoxe.



Figura 2.10: Dixitación aberta [6].

No modo *pechado* (figura 2.11) só se ergue o dedo que se corresponde coa nota que se quere facer soar. É de facer notar que existen diferentes modos de dixitación en pechado, pero a máis extendida no noso país é a que se amosa a continuación.



Figura 2.11: Dixitación pechada [6].

2.1.6. Tonalidade

Tradicionalmente podemos atopar por toda a xeografía galega gaitas afinadas en diferentes tonalidades, as cales reciben os seguintes nomes:

- Gaita grileira (afinada en Re).
- Gaita redonda (afinada en Do).
- Gaita tumbal (afinada en Si b).

A altura de son dun tubo sonoro varía en función da súa lonxitude e do seu calibre interno, de forma que canto maior sexan estes, más grave será a afinación e canto menor sexan, más aguda. Polo tanto, a gaita que maiores dimensíóns ten é a gaita tumbal e a que menores, a gaita grileira.

Para unha maior facilidade de lectura represéntase exactamente igual na partitura unha melodía tocada por unha gaita grileira que por unha gaita tumbal ou por unha gaita redonda, indicando, iso si, ó comezo da peza, que gaita debemos empregar. Á hora de tocar con outros instrumentos debemos ser conscientes da diferenza que hai da representación escrita ó son real que producimos con cada gaita. A continuación amosáse a relación que existe entre ambas [7] (figura 2.12):

	Representación escrita	Son real
Gaita grileira (Re)		
Gaita redonda (Do)		
Gaita tumbal (Si b)		

Figura 2.12: Correspondencia entre tonalidades [7].

2.2. Estado da arte

2.2.1. Protocolos de comunicación de instrumentos musicais

2.2.1.1. MIDI

MIDI é o acrónimo de *Musical Instrument Digital Interface*. Trátase dun estándar técnico que describe un protocolo, unha interface e uns conectores dixitais e permite que unha ampla variedade de instrumentos musicais electrónicos, ordenadores e outros dispositivos relacionados se conecten entre si [8].

Unha soa conexión MIDI pode transportar ata 16 canles de información, cada unha das cales pode ser enrutada a un dispositivo distinto.

O protocolo MIDI transporta mensaxes de eventos que especifican notación musical, altura e velocidade, sinais de control para parámetros coma o volume, o vibrato, o *audio panning*, o *cues* (disparadores) e os sinais de reloxo que establecen e sincronizan o tempo entre varios dispositivos. Estas mensaxes envíanse a outros dispositivos onde se controlan a xeración de son e outras características. Estos datos poden tamén poden gravarse nun dispositivo hardware ou software chamado secuenciador, que pode ser empregado para editar os datos e reproducilos outra vez máis tarde.

A tecnoloxía MIDI foi estandarizada en 1983 por un grupo de representantes da industria musical e é mantida pola *MIDI Manufacturers Association* (MMA). Todos os estándar MIDI son desenvolvidos e publicados conjuntamente pola MMA (USA) e polo *MIDI Committee* da *Association of Musical Electronics Industry* (AMEI) (Xapón).

2.2.1.1. Historia

O repentino inicio dos sintetizadores analóxicos na música popular dos anos 70 levou ós músicos a esixir máis prestacións dos seus instrumentos. Interconectar sintetizadores analóxicos é relativamente sinxelo xa que estes poden controlarse a través de osciladores de voltaxe variable.

A aparición do sintetizador dixital a finais da mesma década trouxo consigo o problema da incompatibilidade dos sistemas que usaba cada compañía fabricante. Deste modo facíase necesario crear unha linguaxe común por enriba dos parámetros que cada marca ía xerando ó longo do desenvolvemento dos distintos instrumentos electrónicos postos a disposición dos profesionais do sector.

O estándar MIDI foi inicialmente proposto nun documento dirixido á *Audio Engineering Society* por *Dave Smith*, presidente da compañía *Sequential Circuits* en 1981. A primeira especificación MIDI publicouse en agosto de 1983.

Cómpre aclarar que o protocolo MIDI non transmite sinais de audio, senón datos de eventos e mensaxes de controladores que se poden interpretar de maneira arbitraria, de acordo coa programación do dispositivo que os recibe. É dicir, o protocolo MIDI é unha especie de “partitura” que contén as instruccións en valores numéricos cando xerar unha nota de son e as características que debe ter; o aparato ó que se envíe dita partitura transformaraa en música completamente audible.

Na actualidade, a gran maioría dos creadores musicais emprega o protocolo MIDI a fin de levar a cabo a edición de partituras e a instrumentación previa á gravación con instrumentos reais. Sen embargo, a perfección adquirida polos

sintetizadores na actualidade leva á utilización de forma directa nas gravacións os sons resultantes do envío da partitura electrónica a ditos sintetizadores de última xeración.

2.2.1.1.2. Hardware

2.2.1.1.2.1. Dispositivos

Os dispositivos MIDI poden clasificarse en tres grandes categorías:

- **Controladores.** Xeran as mensaxes MIDI (activación ou desactivación dunha nota, variacións de ton, etc.). O controlador máis familiar para os músicos ten forma de teclado de piano, ó ser este instrumento o más empregado á hora de compor e integrar as obras orquestais; sen embargo, hoxe en día contrúense todo tipo de instrumentos con capacidade de transmisión vía interface MIDI: órganos de tubos, guitarras, parches de percusión, clarinetes electrónicos e incluso gaitas MIDI.
- **Sintetizadores.** Tamén coñecidos coma unidades xeradoras de son ou módulos de son, reciben as mensaxes MIDI e transfórmansas en sinais sonoros (lembremos que o protocolo MIDI non transmite audio, senón paquetes de ordes en formato numérico).
- **Secuenciadores.** Non son más que aparatos destinados a gravar, reproducir ou editar mensaxes MIDI. Poden desenvolverse ben en formato hardware, ben coma software de ordenador, ou ben incorporados nun sintetizador.

Estes son os tres grandes tipos de dispositivos MIDI. Aínda así, podemos atopar no mercado dispositivos que reúnen dúas ou tres das funcións descritas. Por exemplo, os órganos electrónicos dispoñen dun controlador (o propio teclado) e dunha unidade xeradora de son; algúns modelos tamén inclúen un secuenciador.

2.2.1.1.2.2. Cables e conectores

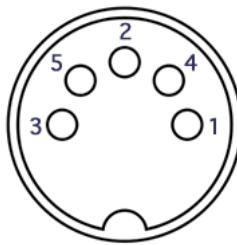


Figura 2.13: Conector MIDI [8].

Un cable MIDI emprega un conector de tipo DIN de 5 pins ou contactos (figura 2.13).

A transmisión de datos só emprega un destes, o número 5. Os números 1 e 3 reserváronse para engadir funcións nun futuro. Os restantes (2 e 4) empréganse -respectivamente- coma blindaxe e para transmitir unha tensión de +5 voltios, para asegurarse de que a electricidade flúa na dirección desexada.

A finalidade do cable MIDI é a de permitir a transmisión dos datos entre dous dispositivos ou instrumentos electrónicos. Na actualidade, os fabricantes dos equipos económicos e, por tanto, moi populares, de empresas tales como Yamaha, Casio, Korg e Roland previron a substitución dos cables e conectores MIDI estándar polos de tipo USB, que son más sinxelos de atopar no comercio e que permiten unha conexión sinxela ós ordenadores persoais.

2.2.1.1.2.3. Conexiós

O sistema de funcionamento do protocolo MIDI é de tipo *simplex*, é dicir, só pode transmitir sinais nun sentido. A dirección que toman os sinais é sempre desde un dispositivo 'mestre' hacia un dispositivo 'escravo'. O primeiro xera a información e o segundo recíbea.

Para entender ben o sistema de conexión, debemos saber que nun dispositivo MIDI pode haber ata tres conectores (figura 2.14):

- **MIDI OUT:** conector do que saen as mensaxes xeradas polo dispositivo mestre.
- **MIDI IN:** sirve para recibir mensaxes desde un dispositivo escravo.

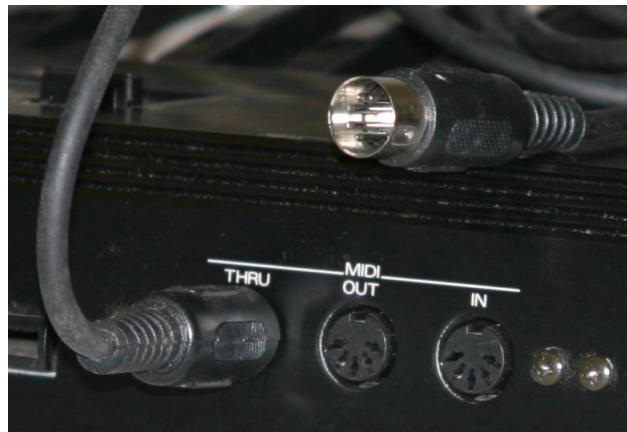


Figura 2.14: Portos e cable MIDI [8].

- **MIDI THRU** tamén é un conector de saída, pero que neste caso envía unha copia exacta das mensaxes que entran por *MIDI IN*.

O formato máis simple de conexión é o formado por un dispositivo mestre (por exemplo, un controlador) e un escravo (coma un sintetizador). Neste caso, o mestre disporá dun conector *MIDI OUT* (de onde sairán as mensaxes MIDI xeradas), o cal deberemos de unir ó conector *MIDI IN* no escravo.

O protocolo MIDI admite a conexión dun solo mestre a varios dispositivos escravos en cascada. Para eses casos empregarase o conector *MIDI THRU*, unindo o mestre cunha das unidades do modo descrito anteriormente. No conector *MIDI THRU* desa unidad obtense unha copia das mensaxes MIDI que entran por *MIDI IN*, polo que se conectamos sucesivamente o *MIDI THRU* dun dispositivo co *MIDI IN* do seguinte, obtemos o que se coñece coma unha conexión **Daisy Chain**.

Aínda que existe a posibilidade de conectar en cascada varios dispositivos MIDI, é certo que existe unha limitación. As características eléctricas dos conectores MIDI fan a señal proclive á degradación, polo que son poucos os dispositivos que se poden conectar en cascada antes de notar perdas apreciables de información.

2.2.1.1.3. Software

A especificación MIDI inclúe un aspecto software que parte desde a mesma organización dos bytes.

2.2.1.3.1. Mensaxes MIDI

A transmisión dos datos efectúase en serie, é dicir, un bit tras outro, de maneira asíncrona, o que obriga a agregar un bit de inicio e outro de parada.

Para clarificar o dito, pódese dicir sinxelamente que unha transmisión asíncrona de datos dase cando un receptor non sabe cando virá o seguinte dato, polo que se atopa en estado de constante espera, xa sexa en nivel alto ou baixo, ata que se produza un cambio de estado que indique o inicio dunha nova mensaxe.

Este primeiro bit debe ser sempre o mesmo, para que sexa sempre diferente ó estado “por defecto”, así que este bit non pode formar parte do byte recibido. A este bit que serve para indicar a chegada dun dato e permite ó dispositivo receptor prepararse para a cadea de bits que vén despois, coñéceselle como “bit de inicio”. Na especificación MIDI, a entrada atópase nun estado alto por defecto, así que o bit de inicio é un 0.

O bit de parada serve para dar tempo ó dispositivo receptor para decidir que facer coa información unha vez recibida. No caso do protocolo MIDI, este bit é sempre o 1.

A velocidade de transmisión do protocolo MIDI definiuse en 31.250 baudios, polo que só deben transcorrer 32 microsegundos entre un bit e o seguinte; nin más, nin menos. Tamén se esixe que a orde de envío dos bits de cada byte sexa LSB, é dicir, o de menos peso primeiro.

Na especificación MIDI existen dous tipos de bytes: de estado -status byte- e de información -data byte-.

Estes diferéncianse polo primeiro bit: se é un 1, será un byte de estado e se é un 0, é un byte de datos.

Ó xerar unha mensaxe MIDI, por norma xeral, sempre enviamos un byte de estado, que pode estar seguido de certa cantidade de bytes de datos. Por exemplo, podemos enviar unha primeira mensaxe de estado “activar nota”, seguida dun byte de datos informando que nota é a que se activa. Nalgúnsas ocasións e segundo o dispositivo MIDI do que trate, pode ocorrer que se omita o byte de estado se é idéntico ó anterior.

Por exemplo, se tocamos a tecla Do central dun teclado MIDI mandaría:

- 1001xxxx (Note on)
- 00111100 (Valor 60 que corresponde á nota Do central ”C3”)
- 0xxxxxxxx (A velocidade ou intensidade co que se foi apretada a tecla)

Pero ó soltala, pode omitir o byte de estado e apagala “por volume” no canto de apagala cunha mensaxe de “Note off” (1000xxxx). É dicir, transmitiría só os dous seguintes bytes:

- 00111100 (Valor 60 que corresponde á nota Do central ”C3”)
- 00000000 (A velocidade cero, que indica que ten que deixar de soar esa nota)

Omitindo así o byte de estado. É máis, se novamente pulsamos a mesma tecla outra vez, volvería omitir o byte de estado.

Á súa vez, as mensaxes de estado divídense en dous grupos: mensaxes de canle e mensaxes de sistema. As mensaxes de canle envíanse a un dispositivo específico, mentres que as mensaxes de sistema son recibidas por todos os dispositivos.

Na seguinte táboa (táboa 2.1) amósanse todas as mensaxes dispoñibles.

Os primeiros bytes, cuxos últimos catro bits están marcados como “cccc”, refírense a mensaxes de canle; o resto de bytes son mensaxes de sistema.

Byte estado	Descripción
1000cccc	Desactivación de nota
1001cccc	Activación de nota
1010cccc	Postpulsación polifónica
1011cccc	Cambio de control
1100cccc	Cambio de programa
1101cccc	Postpulsación monofónica de canle
1110cccc	Pitch
11110000	Mensaxe exclusiva do fabricante
11110001	Mensaxe de trama temporal
11110010	Punteiro posición de canción
11110011	Selección de canción
11110100	Indefinido
11110101	Indefinido
11110110	Requerimento de entoación
11110111	Fin de mensaxe exclusiva
11111000	Reloxo de temporización
11111001	Indefinido
11111010	Inicio
11111011	Continuación
11111100	Parada
11111101	Indefinido
11111110	Espera activa
11111111	Reseteo do sistema

Cadro 2.1: Mensaxes MIDI.

2.2.1.1.3.2. Canles MIDI

Como se comentou con anterioridade, o protocolo MIDI está pensado para comunicar un único controlador con varios sintetizadores (cada un dos cales pode ter un ou varios instrumentos sintetizados que se desexan utilizar), todo por un mesmo medio de transmisión. É dicir, tódolos dispositivos á cadea MIDI reciben tódalas mensaxes xeradas desde o controlador. Isto fai necesario un método para diferenciar cada un dos instrumentos. Este método é o que se denomina **canle**.

O protocolo MIDI pode direccionar ata 16 canles (tamén chamadas voces ou instrumentos); por ilo, ó instalar o sistema MIDI será necesario asignar un número de canle para cada dispositivo.

2.2.1.3.3. General MIDI

O protocolo MIDI permite seleccionar o son dun instrumento a través dunha mensaxe de cambio de programa, pero non hai garantía de que dous dispositivos teñan o mesmo son para un programa determinado. O programa 0 pode corresponderse cun piano nun dispositivo e cunha frauta noutro.

O estándar **General MIDI** (GM) aprobouse en 1991 e estableceu e estandardizou un banco de sons que permite que un ficheiro MIDI estándar creado por un dispositivo soe de maneira similar cando sexa reproducido noutro.

GM especifica un banco de 128 sons agrupados en 16 familias de 8 instrumentos relacionados e asigna un número de programa específico a cada instrumento. Os dispositivos compatibles con GM deben ofrecer polifonías de ata 24 notas. Calquera cambio de programa que se faga seleccionará o mesmo son de instrumento en calquera dispositivo compatible con GM.

O estándar GM elimina a variación no mapeo de notas. GM especifica que a nota número 69 se corresponde coa nota A440 (La a 440 Hz, que é a nota que se toma como referencia para afinar), o que fai que o Do central se fixe como nota número 60.

No estándar GM, a gaita forma parte da familia de instrumentos étnicos (105-112) e corresponde ó número 109.

2.2.1.3.4. Modos MIDI

Dentro do sistema MIDI decidiuse crear unha serie de diferentes **modos** de funcionamento, cada un con certas características. Antes de velos, debemos diferenciar os seguintes conceptos:

- *Monofónico.* Un instrumento monofónico só pode reproducir unha nota simultaneamente. É dicir, para reproducir unha nova nota debe deixar de soar primeiro a nota anterior. Por exemplo, os instrumentos de vento son monofónicos, xa que só reproducen un único son cada vez.
-

- *Polifónico.* Un instrumento polifónico pode reproducir varias notas simultaneamente. Un exemplo é o piano, que pode formar acordes por medio de facer soar dúas ou más notas á vez.

Aclarados estes conceptos, podemos resumir os modos MIDI na seguinte táboa (táboa 2.2).

Número	Nome	Descripción
1	Omni on / poly	Funcionamento polifónico sen información de canle
2	Omni on / mono	Funcionamento monofónico sen información de canle
3	Omni off / poly	Funcionamento polifónico con múltiples canles
4	Omni off / mono	Funcionamento monofónico con múltiples canles

Cadro 2.2: Modos MIDI.

O modo *Omni On* significa que un dispositivo responderá en tódalas canles. Estas configuracións resérvanse para configuracións onde só empreguemos un instrumento.

No modo *Onmi Off* polifónico (modo 3), un dispositivo só responde a unha única canle MIDI predeterminada.

No modo *Onmi Off* monofónico (modo 4), cada voz pode ser asignada á súa propia canle MIDI.

2.2.1.2. OSC

Open Sound Control (OSC) [9] é un formato de contido para a comunicación entre ordenadores, sintetizadores de son e outros dispositivos multimedia optimizados para as tecnoloxías de rede modernas.

Ademais de aportar os beneficios da tecnoloxía moderna para o mundo dos instrumentos musicais electrónicos, as vantaxes de OSC inclúen a interoperabilidade, a precisión, a flexibilidade e unha maior organización e documentación.

2.2.1.2.1. Motivación

OSC é un formato de contido desenvolvido no *Center for New Music and Audio Technologies* (CNMAT) da *University of California*, Berkeley, por *Adrian Freed* e *Matt Wright*, comparable a formatos coma XML, WDDX ou JSON.

Orixinalmente foi pensado para compartir información musical (xestos, parámetros e secuencias de notas) entre instrumentos musicais (especialmente instrumentos musicais electrónicos coma os sintetizadores), ordenadores e outros dispositivos multimedia.

OSC é usado normalmente como alternativa ó estándar MIDI de 1983 naqueles casos nos que se precisa maior resolución e parámetros musicais más elaborados.

As mensaxes OSC envíanse normalmente a través de Internet ou de redes persoais ou dun estudo usando UDP/IP e Ethernet.

OSC ofrece a músicos e desenvolvedores unha maior flexibilidade nos tipos de datos que se poden enviar a través de cable, o que permite que novos tipos de aplicacións se poidan comunicar a alto nivel entre elas.

2.2.1.2.2. Características

- Esquema de nomeado simbólico, dinámico e ampliable.
- Datos numéricos simbólicos e de alta resolución.
- Linguaxe de tipo *pattern matching* para especificar múltiples receptores dunha única mensaxe.
- Etiquetas temporais de alta resolución.
- Conxuntos de mensaxes cuxos efectos poden simultanearse.

Hai ducias de implementacións de OSC, incluíndo son en tempo real e entornos de procesamento de medios, ferramentas web interactivas, sintetizadores software e unha longa lista de linguaxes de programación e dispositivos hardware.

OSC emprégase amplamente en controladores musicais experimentais e incorporouse en múltiples productos comerciais e open source.

2.2.1.2.3. Deseño

As mensaxes OSC constan de:

- Un patrón de dirección: os patróns de direccións forman un espacio de nomes xerárquico, herdado dos sistemas de ficheiros GNU/Linux e das URL.
- Unha cadea que contén etiquetas de tipos: son cadeas que conteñen unha representación dos tipos de datos dos argumentos en forma de etiquetas.
- Argumentos: represéntanse en formato binario con aliñamento de 4 bytes.
Os tipos soportados son:
 - Enteiros sen signo de 32 bits en complemento a dous.
 - Números en punto flotante de 32 bits en formato IEEE.
 - Arrays de datos codificados de 8 bits rematados nun punteiro a nulo.
 - Blobs de tamaño arbitrario (audio ou vídeo).
- Unha etiqueta temporal opcional.

2.2.1.3. MIDI vs OSC

As vantaxes de OSC sobre MIDI son principalmente a conectividade a través de Internet, a resolución dos tipos de datos e a facilidade de especificar unha ruta simbólica.

OSC pode ser enviado a través de conexións Ethernet a velocidades de banda larga, pero é pouco adecuado como solución única para un estudo de grabación, dado que a día de hoxe non conta co apoio da maioría do hardware e software más empregados.

O aumento de tamaño das mensaxes OSC en comparación coas mensaxes MIDI fano unha solución impracticable para moitos dispositivos móveis e as súas

alabadas vantaxes de velocidade sobre MIDI desaparecen cando ambos transmiten polo mesmo medio.

OSC non é propietario, pero non é mantido por ningunha organización estandarizadora.

Por todos estes motivos, decidiuse usar o protocolo MIDI para o proxecto, dado que se adapta moito mellor ás esixencias do mesmo.

2.2.1.4. Alternativas actuais

Pasou xa moito tempo dende que no 1983 se anunciara MIDI coma novo protocolo de comunicación entre instrumentos musicais dixitais heteroxéneos. E se botamos a vista atrás e comparamos co estado actual, a verdade é que non cambiou moito.

Certamente resulta increíble que un sistema con tantos anos ás costas e sen apenas modificacións siga sendo empregado tan profusamente a día de hoxe.

Aínda que houbo un intento de actualizar o protocolo alá polo 2013 de nome *HD-MIDI* [10] e que prometía un maior rango de canles, maior rango e resolución de datos, transporte sobre redes con e sen fíos e incluso soporte de controladores de guitarra, a realidade é que fóra duns poucos prototipos presentados por fabricantes, non se volveu saber máis nada del.

Houbo tamén unha corrente *revival* alternativa en favor de OSC, o conto non pasou de soporte esporádico para algúns software recoñecido de sistemas MacOs, pero sen repercusión real.

Polo que parece, tal e como comenta *Nik Reiman* [11], MIDI non vai ir a ningures, dado que é tan simple e universal, que calquera aplicación ou hardware que realices empregando MIDI non ten un alto risco de obsolescencia por dito motivo.

2.2.2. Gaitas MIDI

Fóra do que poida parecer a priori, o estado da arte en canto a gaitas MIDI se refire é moi amplio e variado, polo que estudialo na súa totalidade describindoo en profundidade escapa do ámbito deste proxecto, aínda que se inclúen as referencias principais necesarias para que o lector interesado poida profundizar no tema.

Curiosamente, os mellores exemplos son os que teñen entre os seus destinos o mercado galego, polo que nos centraremos case exclusivamente nestes modelos, ordenados por cantidad de funcionalidades e, por tanto, tamén por precio de venda.

Así mesmo, comentaranse as carencias detectadas en cada exemplo, as cales se tratarán de subsanar no producto obtido ó final do presente proxecto.

2.2.2.1. Master Gaita



Figura 2.15: Master Gaita [12].

É un controlador MIDI montado nun tubo de PVC con nove dispositivos táctiles. Un cable semiríxido e unha caixa (onde se aloxa a circuitería) serven de punto de apoio a modo de fol para suxeitar o punteiro.

Características:

- Cinco dixitacións cromáticas seleccionables (galega, asturiana, escocesa, francesa e estendida de 2,5 oitavas).
- Compatible con outras gaitas como a de Boto, Sac de Gemecs e Xeremía.

- Cambio a calquera tonalidade e oitava.
- Control independente do roncón e ronqueta, permitindo afinar esta última na tónica ou na dominante.
- Catro programas sonoros seleccionables con dous instrumentos cada un.
- Acompañamento automático a dúo por terceiras superiores ou inferiores.
- Conexión directa a un ordenador empregando un *Game port*.
- Conexión a unha cadea MIDI estándar co cable MIDI incorporado.
- Posibilidade de conexión mediante cable conversor USB-MIDI (non incluído).
- Auto-apagado tras 2 ou 10 minutos (a escoller) de inactividade.
- Suxección similar á dunha gaita.
- Alimentación con pila de 9V ou fonte externa.
- Manual de usuario e conxunto de samples.

Carencias detectadas:

- As dixitacións son limitadas.
 - Non ten conexión directa cun ordenador moderno. Precisa dun conversor, o que implica maiores retardos.
 - Non dispón de conexión sen fíos.
 - Precisa de alimentación separada da conexión de datos.
 - A suxección aínda que similar á dunha gaita, é de todo menos cómoda.
 - Non se pode acoplar a unha gaita.
 - A forma do punteiro é cilíndrica, non cónica como a dun punteiro real, o que xera rechazo.
-

- Os sensores sobresaen do punteiro, o que fai perder a sensación de burato dun punteiro real.

Prezos:

- Master Gaita: 275 euros (IVE n.i.).

2.2.2.2. Technopipes

Figura 2.16: Technopipes [13].

Controlador MIDI montado nun tubo de PVC con posibilidade de acoplarlle un accesorio de apoio (non incluído).

Características:

- Dixitacións galega, asturiana e escocesa.
- Dispón de bordóns.
- Soporta 4 tonalidades: La, Sib, Do e Re.
- Portátil.
- Conta con saída MIDI (cable incluído).
- Conta con saída de audio para auriculares (non incluídos).
- Funcionamento a pilas. Ata 10 horas de autonomía.
- Sensibilidade axustable.

- Conta con metrónomo.
- Pode gravar e reproducir o que se toca.
- Conta cun conxunto de samples.
- Reconfigurable desde o propio controlador.

Carencias detectadas:

- As dixitacións son limitadas.
- Non ten conexión directa cun ordenador moderno. Precisa dun conversor, o que implica maiores retardos.
- Non dispón de conexión sen fíos.
- Non se pode acoplar a unha gaita.
- A forma do punteiro é cilíndrica, non cónica como a dun punteiro real, o que xera rechazo.
- Os sensores sobresaen do punteiro, o que fai perder a sensación de burato dun punteiro real.
- Autonomía limitada.

Prezos:

- Technopipes: 365 euros (IVE inc.).
- Soporte: 26 euros (IVE inc.).

2.2.2.3. ePipe

Consta dun punteiro MIDI e dun conversor de analóxico-dixital de grandes proporcións. Existe a opción de disimular o conversor dentro dun amplificador.

Características:

- Dixitacións galega e asturiana.
-



Figura 2.17: ePipe [14].

- Catro conxuntos de samples.
- Tonalidades: La, Sib, Si, Do, Do♯ e Re.
- Controis independentes de volume.
- Conta con sensor de presión regulable.
- Sensibilidade regulable.
- Conectores MIDI completos.
- Saída para auriculares.

Carencias detectadas:

- As dixitacións son limitadas.
- As tonalidades son limitadas.
- Non ten conexión directa cun ordenador moderno. Precisa dun conversor, o que implica maiores retardos.
- Non dispón de conexión sen fíos.
- Os sensores son pouco convencionais e propensos a erros.
- As dimensíons do conversor analóxico-dixital son desproporcionadas.

Prezos:

- ePipe 15: 645 euros (IVE n.i.).
- ePipe 25 (amplificador): 745 euros (IVE n.i.).

2.2.2.4. Hevia Electronic Bagpipe



Figura 2.18: Hevia MBS 300 [15].

Controlador MIDI que simula unha gaita case completa (sen bordóns reais). O modelo que se analiza aquí é o MBS-300, existindo actualmente un modelo máis recente, o MBS-250, que é máis barato, pero tamén vén con funcionalidades recortadas, o que non nos interesa para a avaliación de características.

Características:

- Sensión de presión regulable.
- Dispón de bordóns.
- Dixitacións totalmente configurables.
- Oitavas totalmente configurables.
- Transporte de tonalidade.
- Soporta glissando e vibrato.
- Reconfigurable desde o propio controlador.
- Módulo bluetooth (opcional). Alcance 100 m.

Carencias detectadas:

- As dixitacións son limitadas.
 - As tonalidades son limitadas.
-

- Non se pode acoplar a unha gaita convencional.
- A forma do punteiro é cilíndrica, non cónica como a dun punteiro real, o que xera rechazo.
- Os sensores exactamente ó mesmo nivel do punteiro, o que dificulta a localización dos mesmos.
- A conexión bluetooth con esa potencia ten un alto consumo, o que merma a súa autonomía ostensiblemente.

Prezos:

- MBS-300: 1.999 euros (IVE n.i.).
- Módulo bluetooth: 127 euros (IVE n.i.).
- Módulo externo de son: 170 euros (IVE n.i.).

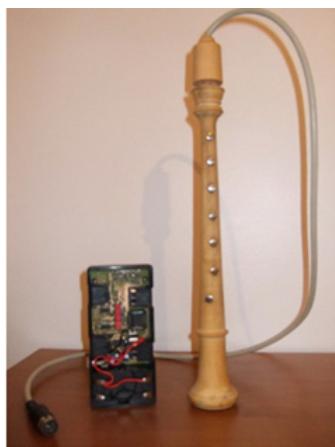
2.2.2.5. OpenPipe

Figura 2.19: OpenPipe [16].

Punteiro MIDI non comercial con formato de punteiro convencional e conexión vía porto MIDI desenvolvido baixo os principios do software libre.

Características:

- Sensibilidade regulable.
- Dispón de bordóns.
- Dixitacións totalmente configurables.
- Transporte de tonalidade.
- Conexión mediante porto MIDI e conversor MIDI-USB.

Carencias detectadas:

- Non se pode acoplar a unha gaita convencional, aínda que sería facilmente modificable para ilo.
- Os sensores sobresaen do punteiro, o que fai perder a sensación de burato dun punteiro convencional.
- Un pouco curto en canto a funcionalidade, pero máis que suficiente para un PFC.

Capítulo 3

Planificación

Índice xeral

3.1. Ciclo de vida	39
3.2. Descripción das tarefas	43
3.3. Xestión do proxecto	48
3.4. Planificación inicial	49

NESTE capítulo exporase a planificación do proxecto: como se levou a cabo, que ciclo de vida se escolleu e por que, como quedou a planificación inicial, etc.

O motivo de poñer a planificación nun capítulo a parte previo ó resto dos capítulos do proxecto, é dar unha visión global dos mesmos coa finalidade de que se entenda o porqué desa distribución.

3.1. Ciclo de vida

O ciclo de vida escollido foi o **ciclo de vida en espiral** [17] (figura 3.1) por ser o que mellor se adapta ás características deste proxecto.

Dito ciclo de vida conta coas seguintes características:

- Permite combinar a natureza interactiva de construción de prototipos cos aspectos controlados e sistemáticos do modelo en cascada.
- Pon especial énfase na análise dos riscos.

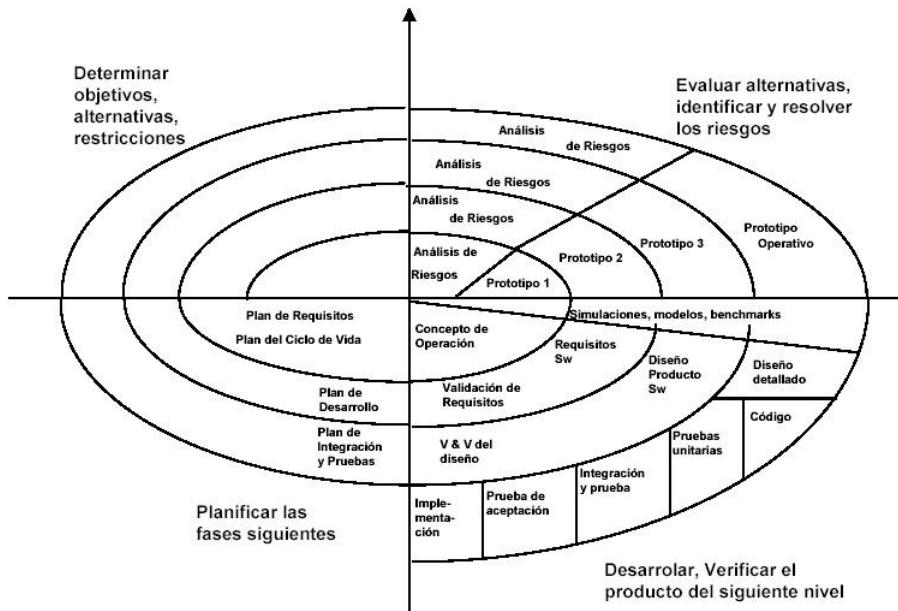


Figura 3.1: Ciclo de vida en espiral [17].

- O software desenvólvese nunha serie de versións incrementais:
 - Durante as primeiras iteracións, a versión incremental será un modelo en papel ou un prototípo.
 - Durante as últimas iteracións, prodúcense versións cada vez más completas do sistema.
 - A idea básica é que cada fase (volta na espiral) establece os seguintes pasos:
 - Determinación de obxectivos, alternativas e restriccions.
 - Avaliación de alternativas (considerando análise de riscos).
 - Desenvolvemento do seguinte nivel de producto.
 - Planificación da seguinte fase (ciclo en espiral).
 - Cada ciclo (volta) na espiral representa unha fase do proceso de desenvolvemento.
 - Por ilo, o ciclo máis interno correspón dese coa viabilidade do sistema.
 - O seguinte, coa análise de requisitos e así sucesivamente.

- Non hai fases fixas no modelo. O modelo adáptase ás necesidades que xordan en cada proxecto.
- Os bloques de deseño, codificación e probas execútanse de forma se-
cuencial para a entrega dunha características iniciais operativas.
- Despois dilo, vólvese revisar o producto e cómo mellorar/ampliar as súas características operativas. O producto actualízase, obtendo un prototipo operativo “posto ó día” que serve de demostración e validación.
- O sistema pasa por un proceso actualizado de desenvolvemento en cascada que finalmente obtén unha nova versión do producto.
- Esquema básico (espiral de 4 pasos):
 1. 1º ciclo:
 - Determinación de:
 - ◊ Obxectivos.
 - ◊ Alternativas.
 - ◊ Restriccions.
 - Avaliación de alternativas e resolución de riscos:
 - ◊ Análise de riscos.
 - ◊ Prototipo 1.
 - Desenvolvemento e validación do seguinte nivel de producto:
 - ◊ Concepto de operación.
 - Planificación da próxima fase (ciclo):
 - ◊ Planificación de requisitos.
 - ◊ Planificación do ciclo de vida.
 2. 2º ciclo:
 - Determinación de:
 - ◊ Obxectivos.
 - ◊ Alternativas.
 - ◊ Restriccions.
 - Avaliación de alternativas e resolución de riscos:
 - ◊ Análise de riscos.

- ◊ Prototipo 2.
 - Desenvolvemento e validación do seguinte nivel de producto:
 - ◊ Simulacións, modelos e programas de proba.
 - ◊ Requisitos do software.
 - ◊ Validación de requisitos.
 - Planificación da próxima fase (ciclo):
 - ◊ Planificación do deseño.
3. 3º ciclo:
- Determinación de:
 - ◊ Obxectivos.
 - ◊ Alternativas.
 - ◊ Restriccións.
 - Avaliación de alternativas e resolución de riscos:
 - ◊ Análise de riscos.
 - ◊ Prototipo 3.
 - Desenvolvemento e validación do seguinte nivel de producto:
 - ◊ Simulacións, modelos e programas de proba.
 - ◊ Deseño do producto software.
 - ◊ Verificación e validación do deseño.
 - Planificación da próxima fase (ciclo):
 - ◊ Planificación do desenvolvemento.
4. 4º ciclo:
- Determinación de:
 - ◊ Obxectivos.
 - ◊ Alternativas.
 - ◊ Restriccións.
 - Avaliación de alternativas e resolución de riscos:
 - ◊ Análise de riscos.
 - ◊ Prototipado operacional.
 - Desenvolvemento e validación do seguinte nivel de producto:
 - ◊ Simulacións, modelos e programas de proba.

- ◊ Deseño detallado.
- ◊ Codificación.
- ◊ Probas de unidade.
- ◊ Integración e probas.
- ◊ Probas de aceptación.
- ◊ Implantación.

As vantaxes que presenta este ciclo de vida son as seguintes:

- Permite adaptar o proceso de desenvolvemento ás necesidades cambiantes do proxecto e ó coñecemento que se vai adquirindo.
- Permite o manexo de prototipos, ligándooos coa análise de riscos.
- Xestioná explicitamente os riscos.

Pero tamén ten os seus inconvintes:

- Requiere dunha considerable habilidade para a consideración do risco.
- O modelo é relativamente novo e non se manexou tanto coma os anteriores.

Como se pode deducir claramente da sección de vantaxes, este ciclo de vida adáptase moi ben ó tipo de proxecto do que estamos a falar.

3.2. Descripción das tarefas

Aplicando o exposto no apartado anterior sobre o ciclo de vida, o seguinte foi adaptalo ás necesidades do proxecto e definir cada unha das tarefas.

Neste apartado describense tódalas tarefas folla listadas na planificación, así coma o traballo realizado en cada unha das mesmas.

1. Análise de viabilidade:

- Determinación de:
 - Obxectivos: estableceranse os obxectivos da fase de análise de viabilidade do proxecto.
-

- Alternativas: estableceranse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se poidan cumplir.
 - Restriccóns: estableceranse restriccóns aplicables a ditos obxectivos.
- Avaliación de alternativas e resolución de riscos:
- Análise de riscos: determinaranse os riscos que comportan as distintas alternativas e as súas posibles solucións.
 - Prototipo 1: realizarase un primeiro prototipo que nos permita intuir como será o producto final.
- Desenvolvemento e validación do seguinte nivel de producto:
- Concepto de operación: trata de definir de maneira conceptual cómo se operará co producto, aclarando que é o que se pretende conseguir máis exactamente, de xeito que sexa máis sinxelo obter os requisitos nunha fase posterior.
- Planificación da próxima fase (ciclo):
- Planificación de requisitos: estableceranse as tarefas da seguinte fase da planificación do proxecto, a análise de requisitos.
 - Planificación do ciclo de vida: establecerase o ciclo de vida a empregar, xustificando o seu uso.

2. Análise de requisitos:

- Determinación de:
- Obxectivos: estableceranse os obxectivos da fase de análise de requisitos do proxecto.
 - Alternativas: estableceranse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se poidan cumplir.
 - Restriccóns: estableceranse restriccóns aplicables a ditos obxectivos.
- Avaliación de alternativas e resolución de riscos:
- Análise de riscos: determinaranse os riscos que comportan as distintas alternativas e as súas posibles solucións.

- Prototipo 2: realizarase un segundo prototipo que nos permita obter de maneira sinxela os requisitos do producto.
- Desenvolvemento e validación do seguinte nivel de producto:
 - Simulacións, modelos e programas de proba: estableceranse unha serie de probas a realizar sobre o prototipo anterior unha vez obtidos os requisitos.
 - Requisitos hardware, software e económicos:
 - Extracción de requisitos propia: estableceranse os requisitos do producto por parte do proxectando.
 - Extracción de requisitos por parte de expertos: estableceranse os requisitos do producto por parte de expertos preseleccionados.
 - Extracción de requisitos a través dos usuarios: Estableceranse os requisitos do producto por parte de usuarios anónimos.
 - Validación de requisitos: cruzaranse tódolos datos obtidos durante a obtención de requisitos e validarase por parte do proxectando e dos expertos que o producto é correcto (fai o que se lle pide), empregando as probas definidas anteriormente.
- Planificación da próxima fase (ciclo):
 - Planificación do deseño: estableceranse as tarefas da seguinte fase da planificación do proxecto, o deseño.

3. Deseño:

- Determinación de:
 - Obxectivos: estableceranse os obxectivos da fase de deseño do proxecto.
 - Alternativas: estableceranse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se poidan cumplir.
 - Restriccións: estableceranse restriccións aplicables a ditos obxectivos.
 - Avaliación de alternativas e resolución de riscos:
-

- Análise de riscos: determinaranse os riscos que comportan as distintas alternativas e as súas posibles solucións.
 - Prototipo 3:
 - Prototipo hardware: realizarase un prototipo hardware mediante ferramentas software.
 - Prototipo software: realizarase un terceiro prototipo, baseado no prototipo da fase anterior integrando tódolos cambios precisos para pasar a validación de requisitos.
 - Desenvolvemento e validación do seguinte nivel de producto:
 - Simulacións, modelos e programas de proba: estableceranse unha serie de probas a realizar sobre o prototipo anterior, co fin de determinar se o deseño posterior é correcto.
 - Deseño do producto hardware e software.
 - Deseño do producto hardware: baseándose no prototipo hardware anterior, realizarase un deseño formal do producto hardware.
 - Deseño do producto software: baseándose no prototipo software anterior, realizarase un deseño formal do producto software.
 - Verificación e validación do deseño: verificarase que o deseño se fixo correctamente (que se axusta ás especificacións) e que é correcto (fai o que se lle pide).
 - Planificación da próxima fase (ciclo):
 - Planificación do desenvolvemento: estableceranse as tarefas da seguinte fase da planificación do proxecto, o desenvolvemento.
4. Desenvolvemento:
- Determinación de:
 - Obxectivos: estableceranse os obxectivos da fase de desenvolvemento do proxecto.
 - Alternativas: estableceranse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se poidan cumplir.

- Restriccóns: estableceranse restriccóns aplicables a ditos obxectivos.
- Avaliación de alternativas e resolución de riscos:
 - Análise de riscos: determinaranse os riscos que comportan as distintas alternativas e as súas posibles solucións.
 - Prototipado operacional.
 - Prototipo hardware:
 - ◊ Integración do hardware: seguindo o prototipo hardware realizado por software da fase anterior e o posterior deseño hardware, realizarase unha versión física do mesmo.
 - ◊ Encapsulamento do hardware: unha vez integrado o hardware será preciso encapsulalo de forma que sexa sinxelo interactuar con el.
 - Prototipo software:
 - ◊ Desenvolvemento do prototipo: elaboración dun prototipo software completo (incluindo a parte do controlador hardware).
 - ◊ Gravación dos samples: elaboración, partindo de cero, dunha fonte de sons de calidade para empregar co producto.
- Desenvolvemento e validación do seguinte nivel de producto:
 - Simulacións, modelos e programas de proba: estableceranse unha serie de probas a realizar sobre o prototipo anterior, co fin de determinar se o deseño posterior é correcto.
 - Deseño detallado:
 - Deseño hardware: ampliación do deseño hardware da fase anterior.
 - Deseño software: ampliación do deseño software da fase anterior.
 - Ensamblado e codificación:
 - Ensamblado: integración do hardware e do encapsulamento.
 - Codificación: desenvolvemento completo software.

- Probas de unidade: testeо das distintas partes do producto facendo uso das probas definidas previamente.
- Integración e probas: fusión de tódalas partes do producto nunha única e posterior testeо facendo uso das probas definidas previamente.
- Probas de aceptación: verificación e validación do producto por parte de expertos.
- Implantación: simulación da implantación do producto nun sistema limpo e preparado para o mesmo. Por exemplo, unha máquina virtual creada expresamente para a defensa do proxecto.

3.3. Xestión do proxecto

Para poder manexar as tarefas anteriores era necesario facer uso dunha ferramenta de xestión de proxectos, comunmente coñecida como **xestor de proxectos** ou planificador.

Tendo en conta a filosofía do proxecto, o correcto era empregar un xestor de proxectos libre. Facendo un pouco de investigación e preguntando a profesionais con experiencia no uso dos mesmos, preseleccionáronse varios de entre os moitos existentes, para o seu posterior estudio:

- OpenProj [18]
- Planner [19]
- GanttProject [20]
- LibrePlan [21]

Nunha primeira avaliación superficial, *Planner* e *GanttProject* arroxaron unha imaxe de demasiado básicos, mentres que *LibrePlan* permite xestionar proxectos complexos, co inconviante de ter unha interface máis lenta que as alternativas xa mencionadas.

Foi nesta primeira avaliación onde *OpenProj* destacou sobre o resto por quedarxe xusto en medio: lixeiro, completo e moi similar (por non dicir un clon) ó *MS Project* [22], planificador xa manexado polo proxectando e, polo tanto, reducindo a curva de aprendizaxe e o tempo invertido en dominalo.

Escolleuse entón *OpenProj* coma primeira opción, pero logo dunha serie de incidencias que se comentan no capítulo 8, a opción final empregada foi **LibrePlan**.

Para rematar, facer notar que por motivos de dispoñibilidade de tempo do proxectando (estudos, traballo, etc.) decidiuse empregar un calendario de media xornada para o proxecto.

3.4. Planificación inicial

Pouco a pouco fóreronse introducindo todos os datos ata dar coa **planificación inicial**, cun total de **688 horas** de traballo planificadas.

A continuación amósase a planificación inicial (figuras 3.2 e 3.3) recortada coas tarefas organizadas de forma arbórea. Dada a súa extensión, iranse desglosando por ciclos nos seguintes capítulos para facilitar a súa comprensión.

Pode consultarse a planificación completa (incluíndo as dependencias entre tarefas) no apéndice A.

A estimación da duración de cada tarefa contempla tanto o erro na propia estimación coma os posibles desvíos derivados da análise de riscos e outros imprevistos.

A modo de conclusión, comentar que ás veces merece a pena perder algo de tempo probando unha ferramenta que se sabe que será de calidade, aínda que resulte más complexa, ca empregar outra máis sinxela pero con alta probabilidade de fallo, como así aconteceu.

LIBREPLAN
OpenWebPlanning

INICIO > Planificación > Planificación de prox.

Nome	Inicio	Fin
☐ PFC	9/20/11	5/26/12
☐ Análise de viabilidade	9/20/11	10/1/11
☐ Determinación 1	9/20/11	9/21/11
☐ Obxectivos 1	9/20/11	9/20/11
☐ Alternativas 1	9/20/11	9/20/11
☐ Restriccións 1	9/20/11	9/21/11
☐ Avaliación de alternativas e resolución de riscos 1	9/21/11	9/23/11
☐ Análise de riscos 1	9/21/11	9/22/11
☐ Prototipo 1	9/22/11	9/23/11
☐ Desenvolvemento e validación do seguinte nivel do producto 1	9/23/11	9/27/11
☐ Concepto de operación	9/23/11	9/27/11
☐ Planificación da seguinte fase ou ciclo 1	9/27/11	10/1/11
☐ Planificación de requisitos	9/27/11	9/29/11
☐ Planificación do ciclo de vida	9/29/11	10/1/11
☐ Análise de requisitos	10/1/11	11/26/11
☐ Determinación 2	10/1/11	10/4/11
☐ Obxectivos 2	10/1/11	10/3/11
☐ Alternativas 2	10/3/11	10/3/11
☐ Restriccións 2	10/3/11	10/4/11
☐ Avaliación de alternativas e resolución de riscos 2	10/4/11	10/12/11
☐ Análise de riscos 2	10/4/11	10/5/11
☐ Prototipo 2	10/5/11	10/12/11
☐ Desenvolvemento e validación do seguinte nivel do producto 2	10/12/11	11/18/11
☐ Simulacóns, modelos e programas de proba 2	10/12/11	10/19/11
☐ Requisitos hardware, software e económicos	10/19/11	11/15/11
☐ Extracción de requisitos propia	10/19/11	10/21/11
☐ Extracción de requisitos por parte de expertos	10/21/11	10/25/11
☐ Extracción de requisitos a través dos usuarios	10/25/11	11/15/11
☐ Validación de requisitos	11/15/11	11/18/11
☐ Planificación da próxima fase ou ciclo 2	11/18/11	11/26/11
☐ Planificación do deseño	11/18/11	11/26/11
☐ Deseño	11/26/11	1/21/12
☐ Determinación 3	11/26/11	11/29/11
☐ Obxectivos 3	11/26/11	11/28/11
☐ Alternativas 3	11/28/11	11/28/11
☐ Restriccións 3	11/28/11	11/29/11
☐ Avaliación de alternativas e resolución de riscos 3	11/29/11	12/14/11
☐ Análise de riscos 3	11/29/11	11/30/11
☐ Prototipo 3	11/30/11	12/14/11
☐ Prototipo hardware 3	11/30/11	12/6/11
☐ Prototipo software 3	11/30/11	12/14/11
☐ Desenvolvemento e validación do seguinte nivel do producto 3	12/14/11	1/7/12
☐ Simulacóns, modelos e programas de proba 3	12/14/11	12/21/11
☐ Deseño do producto hardware e software	12/21/11	1/4/12
☐ Deseño do producto hardware	12/21/11	12/28/11
☐ Deseño do producto software	12/28/11	1/4/12
☐ Verificación e validación do deseño	1/4/12	1/7/12
☐ Planificación da próxima fase ou ciclo 3	1/7/12	1/21/12
☐ Planificación do desenvolvemento	1/7/12	1/21/12
☐ Desenvolvemento	1/21/12	5/26/12

Figura 3.2: Planificación inicial (1/2) [21].

<input checked="" type="checkbox"/>	Desenvolvemento	1/21/12	5/26/12
<input checked="" type="checkbox"/>	Determinación 4	1/21/12	1/24/12
	Obxectivos 4	1/21/12	1/23/12
	Alternativas 4	1/23/12	1/23/12
	Restriccions 4	1/23/12	1/24/12
<input checked="" type="checkbox"/>	Avaliación de alternativas e resolución de riscos 4	1/24/12	3/3/12
	Análise de riscos 4	1/24/12	1/25/12
<input checked="" type="checkbox"/>	Prototipado operacional	1/25/12	3/3/12
<input checked="" type="checkbox"/>	Prototipo hardware 4	1/25/12	2/10/12
	Integración do hardware	1/25/12	2/4/12
	Encapsulamento do hardware	2/4/12	2/10/12
<input checked="" type="checkbox"/>	Prototipo software 4	2/10/12	3/3/12
	Desenvolvemento do prototipo	2/10/12	3/3/12
	Gravación dos samples	2/10/12	2/16/12
<input checked="" type="checkbox"/>	Desenvolvemento e validación do seguinte nivel do producto 4	3/3/12	5/26/12
	Simulacions, modelos e programas de proba 4	3/3/12	3/10/12
<input checked="" type="checkbox"/>	Deseño detallado	3/10/12	3/24/12
	Deseño hardware	3/10/12	3/17/12
	Deseño software	3/17/12	3/24/12
<input checked="" type="checkbox"/>	Ensamblado e codificación	3/24/12	5/5/12
	Ensamblado	3/24/12	4/7/12
	Codificación	4/7/12	5/5/12
	Probas de unidade	5/5/12	5/12/12
	Integración e probas	5/12/12	5/19/12
	Probas de aceptación	5/19/12	5/24/12
	Implantación	5/24/12	5/26/12

Figura 3.3: Planificación inicial (2/2) [21].

Capítulo 4

Análise de viabilidade

Índice xeral

4.1. Planificación inicial	54
4.2. Determinación	54
4.2.1. Obxectivos	54
4.2.2. Alternativas	55
4.2.3. Restriccións	57
4.3. Avaliación de alternativas e resolución de riscos	57
4.3.1. Análise de riscos	57
4.3.2. Prototipo 1	61
4.4. Desenvolvemento e validación do seguinte nivel do producto	63
4.4.1. Concepto de operación	63
4.5. Planificación da seguinte fase ou ciclo	64
4.5.1. Planificación de requisitos	64
4.5.2. Planificación do ciclo de vida	64
4.6. Estudio de viabilidade	64

NESTE capítulo exporase a análise de viabilidade do proxecto baseándose no esquema proporcionado pola planificación inicial: desde o estudo das solucións xa existentes ata a realización dunha enquisa entre os posibles usuarios, pasando por un primeiro prototípo.

4.1. Planificación inicial

A continuación (figura 4.1) exponse a planificación inicial do ciclo de análise da viabilidade.

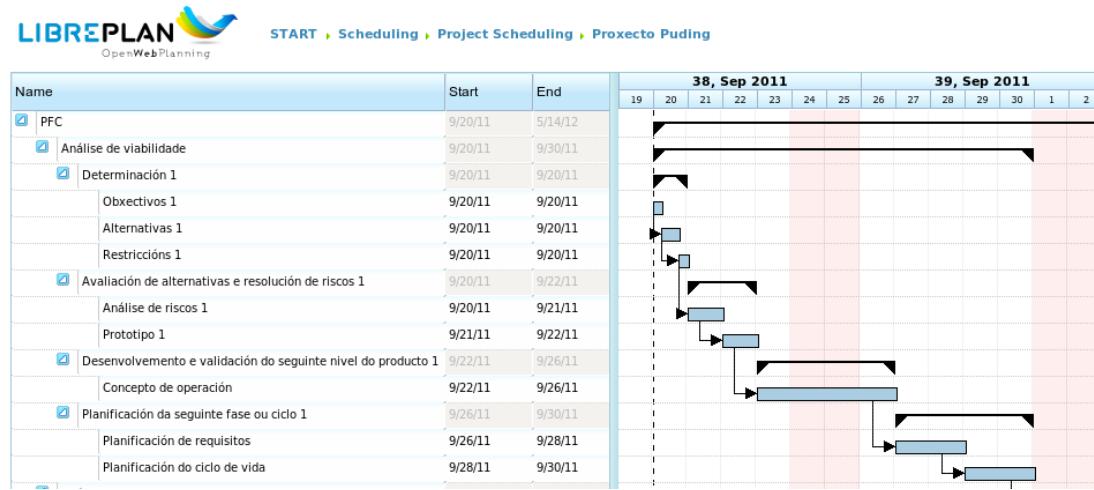


Figura 4.1: Planificación inicial do ciclo de análise de viabilidade.

Total: 36 horas.

4.2. Determinación

4.2.1. Obxectivos

Estableceranse os obxectivos da fase de análise de viabilidade do proxecto.

Obxectivos:

1. Determinar a viabilidade (1) de deseñar (2) e implementar (3) unha gaita MIDI (4) sen fíos (5) en tempo real (6) empregando software (7)/ hardware (8) libre.¹

¹Os números entre parénteses indican a partición do obxectivo principal en subobxectivos. Preséntanse así para non distorsionar o mesmo. Serán referenciados coma 1.X nas seguintes seccións.

4.2.2. Alternativas

Estableceronse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se puidesen cumprir.

Alternativas:

1. Se non é posible determinar a viabilidade do proxecto, pode optarse por:
 - a) Analizar soamente aquelas partes que sexa posible.
 - b) Saltarse completamente a análise de viabilidade e avanzar a cegas.
 - c) Cancelar e mudar de proxecto
2. Se non é posible determinar a viabilidade do deseño, pode optarse por:
 - a) Analizar soamente aquelas partes do deseño que sexa posible.
 - b) Saltarse completamente a análise de viabilidade do deseño e avanzar a cegas.
 - c) Empregar o deseño dalgunha das solucións xa existentes.
 - d) Cancelar e mudar de proxecto.
3. Se non é posible determinar a viabilidade da implementación, pode optarse por:
 - a) Analizar soamente aquellas partes da implementación que sexa posible.
 - b) Saltarse completamente a análise de viabilidade da implementación e avanzar a cegas.
 - c) Empregar a implementación dalgunha das solucións xa existentes.
 - d) Cancelar e mudar de proxecto.
4. Se non é posible determinar a viabilidade do emprego do protocolo MIDI, pode optarse por:
 - a) Analizar soamente aquellas partes do protocolo que sexa posible.
 - b) Saltarse completamente a análise de viabilidade do protocolo e avanzar a cegas.

- c) Empregar outro protocolo similar alternativo.
 - d) Cancelar e mudar de proxecto.
5. Se non é posible determinar a viabilidade do emprego de tecnoloxía sen fíos, pode optarse por:
- a) Analizar soamente aquelas partes da tecnoloxía que sexa posible.
 - b) Saltarse completamente a análise de viabilidade da tecnoloxía e avanzar a cegas.
 - c) Empregar conexión por cable.
 - d) Cancelar e mudar de proxecto.
6. Se non é posible determinar a viabilidade do emprego de tempo real, pode optarse por:
- a) Analizar soamente aquelas partes nas que sexa posible determinar a viabilidade do emprego de tempo real.
 - b) Saltarse completamente a análise de viabilidade do emprego de tempo real e avanzar a cegas.
 - c) Cancelar e mudar de proxecto.
7. Se non é posible determinar a viabilidade do emprego de software libre, pode optarse por:
- a) Analizar soamente aquelas partes nas que sexa posible determinar a viabilidade do emprego de software libre.
 - b) Saltarse completamente a análise de viabilidade do emprego de software libre e avanzar a cegas.
 - c) Empregar software privativo nas partes onde non é posible determinar a viabilidade do emprego de software libre.
 - d) Cancelar e mudar de proxecto.
8. Se non é posible determinar a viabilidade do emprego de hardware libre, pode optarse por:

- a) Analizar soamente aquelas partes nas que sexa posible determinar a viabilidade do emprego de hardware libre.
- b) Saltarse completamente a análise de viabilidade do emprego de hardware libre e avanzar a cegas.
- c) Empregar hardware privativo nas partes onde non é posible determinar a viabilidade do emprego de hardware libre.
- d) Cancelar e mudar de proxecto.

4.2.3. Restriccóns

Estableceronse restriccóns aplicables a ditos obxectivos.

- As propias restriccóns veñen dadas polo propio título do proxecto. A saber:
 - 1. Empregar o protocolo MIDI (1.4).
 - 2. Empregar tecnoloxía sen fíos (1.5).
 - 3. Empregar tempo real (1.6).
 - 4. Empregar software libre (1.7).
 - 5. Empregar hardware libre (1.8).
 - 6. E/ou as derivadas de calquera das súas alternativas.

4.3. Avaliación de alternativas e resolución de riscos

4.3.1. Análise de riscos

Determináronse os riscos que comportaban as distintas alternativas e as súas posibles solucións.

- 1. Alternativas 1.1.
 - a) Riscos:
 - 1) Que as partes que queden sen analizar sexan inviables.
-

- 2) Que parte ou a totalidade do proxecto sexa inviable.
- 3) Que o tempo restante para a execución do proxecto non sexa suficiente.

b) Solucións:

- 1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 3) Agardar a presentalo na seguinte convocatoria.

2. Alternativas 1.2.

a) Riscos:

- 1) Que as partes que queden sen analizar sexan inviables.
- 2) Que parte ou a totalidade do deseño sexa inviable.
- 3) Enxeñería inversa. Alto custe e alta probabilidade de erros no deseño.
- 4) Que o tempo restante para a execución do proxecto non sexa suficiente.

b) Solucións:

- 1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 3) Solicitar o empréstimo dos deseños ou cancelar e mudar de proxecto.
- 4) Agardar a presentalo na seguinte convocatoria.

3. Alternativas 1.3.

a) Riscos:

- 1) Que as partes que queden sen analizar sexan inviables.
- 2) Que parte ou a totalidade da implementación sexa inviable.

3) Enxeñería inversa. Alto custe e alta probabilidade de erros na implementación.

4) Que o tempo restante para a execución do proxecto non sexa suficiente.

b) Solucións:

1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.

2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.

3) Solicitar o empréstito da implementación ou cancelar e mudar de proxecto.

4) Agardar a presentalo na seguinte convocatoria.

4. Alternativas 1.4.

a) Riscos:

1) Que as partes que queden sen analizar sexan inviables.

2) Que parte ou a totalidade do protocolo sexa inviable.

3) Que non exista protocolo alternativo ou non sexa viable.

4) Que o tempo restante para a execución do proxecto non sexa suficiente.

b) Solucións:

1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.

2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.

3) Cancelar e mudar de proxecto.

4) Agardar a presentalo na seguinte convocatoria.

5. Alternativas 1.5.

a) Riscos:

1) Que as partes que queden sen analizar sexan inviables.

- 2) Que parte ou a totalidade da conexión sen fíos sexa inviable.
- 3) Que a conexión por cable tampouco sexa viable.
- 4) Que o tempo restante para a execución do proxecto non sexa suficiente.

b) Solucións:

- 1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 3) Cancelar e mudar de proxecto.
- 4) Agardar a presentalo na seguinte convocatoria.

6. Alternativas 1.6.

a) Riscos:

- 1) Que as partes que queden sen analizar sexan inviables.
- 2) Que parte ou a totalidade do emprego de tempo real sexa inviable.
- 3) Cancelar e mudar de proxecto.

b) Solucións:

- 1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 3) Agardar a presentalo na seguinte convocatoria.

7. Alternativas 1.7.

a) Riscos:

- 1) Que as partes que queden sen analizar sexan inviables.
- 2) Que parte ou a totalidade do software libre a emplegar sexa inviable.
- 3) Que non exista software privativo e/ou que o que haxa non sexa viable.

4) Cancelar e mudar de proxecto.

b) Solucións:

- 1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 3) Cancelar e mudar de proxecto.
- 4) Agardar a presentalo na seguinte convocatoria.

8. Alternativas 1.8.

a) Riscos:

- 1) Que as partes que queden sen analizar sexan inviables.
- 2) Que parte ou a totalidade do hardware libre a empregar sexa inviable.
- 3) Que non exista hardware privativo e/ou que o que haxa non sexa viable.
- 4) Cancelar e mudar de proxecto.

b) Solucións:

- 1) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 2) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
- 3) Cancelar e mudar de proxecto.
- 4) Agardar a presentalo na seguinte convocatoria.

4.3.2. Prototipo 1

Realizouse un primeiro prototipo (figura 4.2) que nos permite intuir como será o producto final.



Figura 4.2: Prototipo 1.

4.4. Desenvolvemento e validación do seguinte nivel do producto

4.4.1. Concepto de operación

Tratouse de definir de maneira conceptual como se operará co producto, aclarando que é o que se pretendía conseguir máis exactamente, de xeito que fose máis sinxelo obter os requisitos nunha fase posterior.

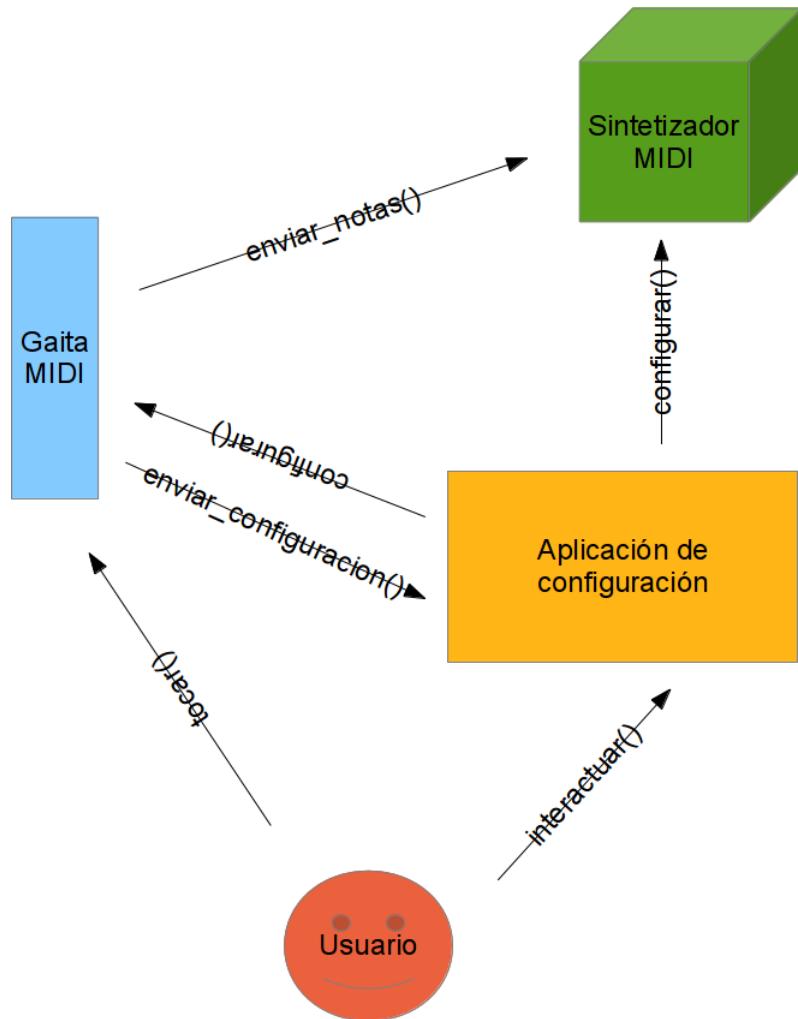


Figura 4.3: Concepto de operación.

Se observamos á gráfica anterior (figura 4.3) vemos que contamos con catro

actores:

- *Usuario*: será o encargado de *tocar()* a Gaita MIDI e de *interactuar()* coa *Aplicación de configuración*.
- *Aplicación de configuración*: será a encargada de *configurar()* tanto a *Gaita MIDI* coma o *Sintetizador MIDI*.
- *Gaita MIDI*: será a encargada de *enviar_notas()* ó *Sintetizador MIDI* e de *enviar_configuracion()* á *Aplicación de configuración*.
- *Sintetizador MIDI*: será o encargado de *reproducir()* as notas ou mensaxes MIDI que lle chegan da *Gaita MIDI*.

4.5. Planificación da seguinte fase ou ciclo

4.5.1. Planificación de requisitos

A continuación (figura 4.4) exponse a planificación inicial do ciclo de análise de requisitos.

Total: 160 horas.

4.5.2. Planificación do ciclo de vida

A planificación do ciclo de vida expúxose xa no capítulo 3 coa finalidade de dar unha visión global dos capítulos posteriores.

4.6. Estudio de viabilidade

A presente sección sitúase ó final do capítulo para non distorsionar a orde de exposición das tarefas reflexadas na planificación.

Para estudio de viabilidade e a extracción de requisitos a través dos usuarios creouse unha enquisa anónima dispoñible en liña [23]. No presente apartado centrámonos exclusivamente na parte relacionada coa viabilidade do proxecto. As

Name	Start	End
Análise de requisitos	9/30/11	11/25/11
Determinación 2	9/30/11	10/3/11
Obxectivos 2	9/30/11	10/3/11
Alternativas 2	10/3/11	10/3/11
Restriccóns 2	10/3/11	10/3/11
Avaliación de alternativas e resolución de riscos 2	10/3/11	10/11/11
Análise de riscos 2	10/3/11	10/4/11
Prototipo 2	10/4/11	10/11/11
Desenvolvemento e validación do seguinte nivel do producto 2	10/11/11	11/17/11
Simulacións, modelos e programas de proba 2	10/11/11	10/18/11
Requisitos hardware, software e económicos	10/18/11	11/14/11
Extracción de requisitos propia	10/18/11	10/20/11
Extracción de requisitos por parte de expertos	10/20/11	10/24/11
Extracción de requisitos a través dos usuarios	10/24/11	11/14/11
Validación de requisitos	11/14/11	11/17/11
Planificación da próxima fase ou ciclo 2	11/17/11	11/25/11
Planificación do deseño	11/17/11	11/25/11
Deseño	11/25/11	11/25/11

Figura 4.4: Planificación inicial do ciclo de análise de requisitos.

relacionadas coa análise de requisitos comentaranse no capítulo seguinte.

Na enquisa preguntábase sobre datos persoais non identificativos (idade, procedencia), o seu perfil coma gaiteiro, coñecementos sobre as gaitas MIDI, caracterísiticas (fixadas de antemán en extraccións previas) preferibles dunha gaita MIDI e a súa prioridade, características propias a engadir e a súa prioridade e un pequeno estudio económico.

A redacción da enquisa foi a seguinte:

O Proxecto Puding trata de crear unha gaita MIDI integralmente libre e galega.

Esta enquisa está orientada a recoller os requisitos desexables dunha gaita MIDI e a estudiar o perfil do tipo de gaiteiro interesado nela.

A enquisa é totalmente anónima (a excepción dos datos xeográficos e de idade) precisos para a elaboración do perfil. Tódolos datos

recabados serán empregados exclusivamente para a fin anteriormente exposta.

Os únicos requisitos desexables para cubrir esta enquisa son:

- *Ser gaiteiro.*
- *Responder o máis completa e correctamente a mesma. Os datos incorrectos, incongruentes ou incompletos son contraproducentes para o estudio, polo que serán excluidos dos resultados.*

Gracias de antemán.

Enquisa:

1. *¿Que anos tes?*
2. *¿En que localidade resides?*
3. *¿A que provincia pertence dita localidade?*
 - a) *A Coruña.*
 - b) *Lugo.*
 - c) *Ourense.*
 - d) *Pontevedra.*
 - e) *Outra. ¿Cal?*
4. *Como gaiteiro, ¿en que nivel te incluirías?*
 - a) *Principiante.*
 - b) *Intermedio.*
 - c) *Profesional.*
5. *¿Contas con gaita de teu?*
 - a) *Si.*
 - b) *Non, só cun punteiro de iniciación.*
 - c) *Non, pero téñoa encargada xa.*
 - d) *Non.*
6. *¿Que pensas da aplicación da tecnoloxía actual ó mundo da gaita?*

- a) Creo que non aportaría vantaxes sobre o que hai.
- b) A tecnoloxía podería traer vantaxes, pero por tradición preferiría seguir empregando a gaita de toda a vida.
- c) Gustaríame preservar a maior parte da gaita tradicional, pero melloraría con tecnoloxía certos aspectos.
- d) A tecnoloxía está totalmente infrautilizada no mundo da gaita. Vai sendo hora de crear unha gaita 2.0.

7. ¿Sabes o que é unha gaita ou punteiro MIDI?

- a) Si.
- b) Non, pero teño escoitado falar delas.
- c) Non.

8. En caso afirmativo, ¿conñeces algunha das seguintes?

- a) Master Gaita
- b) Technopipe
- c) E-pipe
- d) Hevia Electronic Bagpipe
- e) Outra. ¿Cal?

9. ¿Estarías interesado nunha gaita ou punteiro MIDI?

- a) Si, teño pensado mercar unha.
- b) Si, pero só se cumpre coas miñas expectativas.
- c) Non, pero podería chegar a interesarme se xurde algo xeitoso.
- d) Non, de ningunha maneira.

10. En caso afirmativo, ¿que uso lle darías? [Resposta múltiple]

- a) Para ensaiar na casa.
- b) Para componer ou pasar pezas a un ordenador.
- c) Para empregar nas clases.
- d) Para empregar en concertos.
- e) Outro. ¿Cal?

11. ¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade (1 a 5 -a maior número, maior prioridade-)?

- a) Ausencia de retardos.
 - b) Salvagarda da configuración entre usos.
 - c) Presión regulable.
 - d) Sensores dos dedos regulables independentemente.
 - e) Afinación base regulable.
 - f) Afinación regulable por nota.
 - g) Dixitación personalizable.
 - h) Sensor da presión do fol deshabilitable.
 - i) Vibrato.
 - l) Glisandos.
 - m) Bordóns.
 - n) Samples de calidad.
 - ñ) Punteiro con forma tradicional.
 - o) Punteiro acoplable a un fol calquera.
 - p) Posibilidade de conexión a un ordenador.
 - q) Conexión sen fíos.
12. Indica outras características que che gustaría que tivese, ordeadas por prioridade.
13. ¿Estarías disposto a mercar unha gaita MIDI coas características anteriores (incluídas as que ti mesmo engadiches)?
- a) Si.
 - b) Non.
14. De ser así, ¿canto estarías disposto a pagar por unha?
15. En caso contrario, ¿por que non?
16. Sabendo que as gaitas MIDI presentes hoxe en día no mercado oscilan entre os 275 euros e os 2800 euros e que a maior parte delas non satisfacen as características anteriores ¿canto estarías disposto a pagar por unha que as inclúise todas ou case todas?

É de facer notar que as respostas a dita enquisa non son públicas, polo que ningún usuario previo pode influir coas súas respostas sobre usuarios posteriores. Evidentemente, esta medida está enfocada a evitar un posible problema de

falsa correlación forte nas respostas, o que podería invalidar totalmente a enquisa.

Os resultados completos poden consultarse no apéndice B.

Os resultados relativos á viabilidade foron os das seguintes figuras (4.5 a 4.7).

Tendo en conta estes resultados, podemos afirmar que o perfil xeral do gaiteiro que respondeu a enquisa é o seguinte:

- Menor de 30 anos.
- Galego e maiormente da provincia de Lugo.
- Con residencia fóra das capitais de provincia.
- Cun nivel intermedio coma gaiteiro.
- Con gaita de seu.
- Que lle gustaría preservar a maior parte da gaita tradicional, pero melloraría con tecnoloxía certos aspectos. Aínda que hai unha parte importante que aínda pensando que a tecnoloxía podería traer vantaxes, preferiría seguir empregando a gaita de toda a vida.
- Que coñece o concepto de gaita MIDI.
- Que coñece un ou varios modelos de gaitas MIDI existentes no mercado. Pero que curiosamente o máis vendido actualmente (e con moita diferencia) é o menos coñecido de todos.
- Que está interesado en mercar unha gaita MIDI se cumpre coas súas expectativas ou que podería chegar a estalo se xorde algo xeitoso.

Así mesmo, logo de respostar á parte referida ós requisitos (que como xa se comentou, se exporá no seguinte capítulo), volvéuselles preguntar se estarían dispostos a mercar unha gaita MIDI coas características expostas, incluíndo as que eles mesmos reclamaban.

Como se pode observar na figura 4.8, a resposta foi totalmente afirmativa, incluso por parte de aqueles que albergaban dúbidas ó respecto, o que fai pensar

10 responses

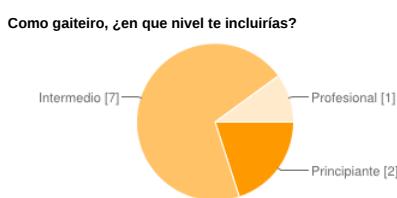
Resumen [Ver las respuestas completas](#)

¿Que anos tés?
14 18 17 41 22 28 23 35 19 24

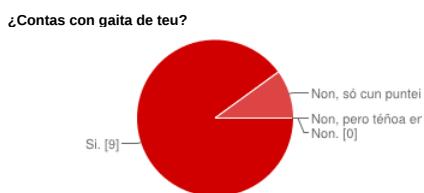
¿En que localidade resides?
Rábade Lugo Meira Neda Oviedo Rodeiro Antas de Ulla Rodeiro Alcalá de Henares Madrid



A Coruña	1	10%
Lugo	4	40%
Ourense	0	0%
Pontevedra	2	20%
Other	3	30%



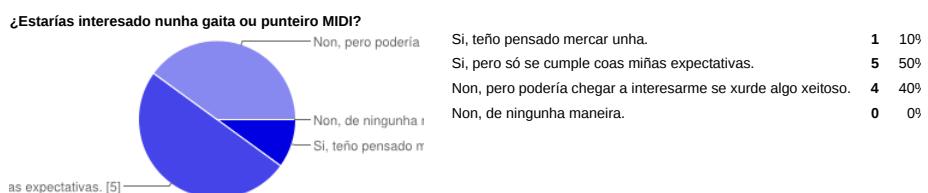
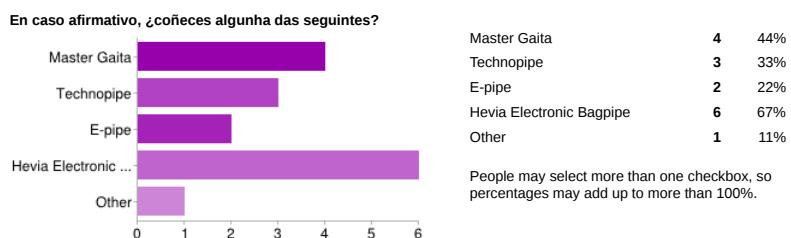
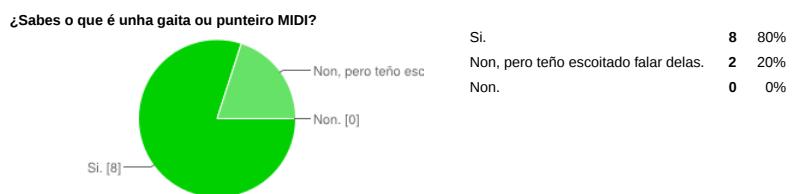
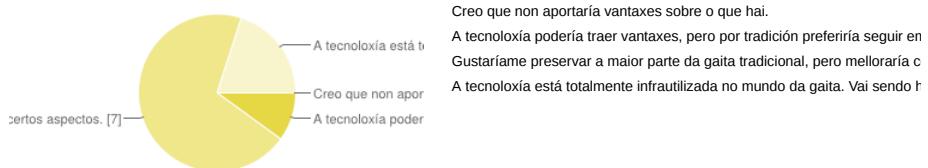
Principiante	2	20%
Intermedio	7	70%
Profesional	1	10%



Si.	9	90%
Non, só cun punteiro de iniciación.	1	10%
Non, pero téñoa encargada xa.	0	0%
Non.	0	0%

¿Que pensas da aplicación da tecnoloxía actual ó mundo da gaita?

Figura 4.5: Resultados do estudio de viabilidade (p. 1).



En caso afirmativo, ¿que uso lle darías?

Figura 4.6: Resultados do estudio de viabilidade (p. 2).

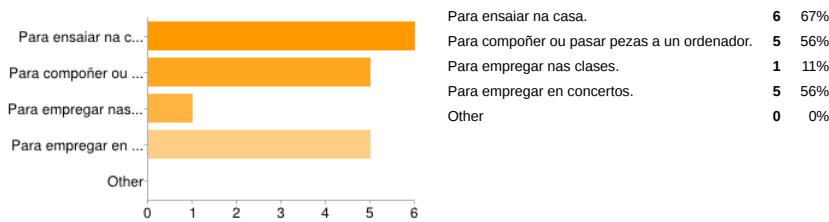


Figura 4.7: Resultados do estudio de viabilidade (p. 3).

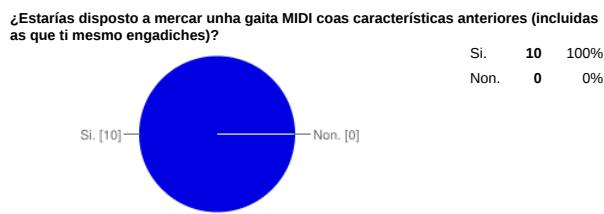


Figura 4.8: Resultados do estudio de viabilidade (p. 8).

que se vai por bo camiño. Por este motivo, a pregunta sobre as razóns para non mercala quedou sen resposta.

A modo de avaliación da enquisa, comentar que é evidente que o número de persoas que respondaron a enquisa é claramente non significativo. Se se aplica o **problema de Fermi** [24] a este caso cos seguintes datos:

- Adoita haber 1 grupo de gaitas en cada concello.
- Cada grupo adoita estar composto por 10 músicos, dos cales a metade adoitan ser gaiteiros.
- Galicia conta con 315 concellos.

Pódese estimar entón que só en Galicia hai xa preto de 1600 gaiteiros.

Se se repite dita estimación co total de grupos rexistrados (273) na *Asociación de Gaiteiros Galegos* [25] en lugar de concellos, arroxa un total aproximado de 1400 gaiteiros.

Tomando como dato final a media de ditas estimacións, saen **1500 gaiteiros**.

Polo que a mostra que se obtivo sería inferior ó 1% do total estimado. Para que a enquisa resultase significativa a mostra debería de ser bastante maior.

Supoñendo unha distribución normal e aplicando a fórmula para o cálculo do tamaño mostral coñecido o tamaño da poboación [26]:

$$n = \frac{N}{1 + \frac{e^2(N-1)}{(z^2)pq}} \quad (4.1)$$

Onde:

- n , é o tamaño da mostra.
- N , o tamaño da poboación.
- e , o erro mostral (marxe de erro admitida).
- z , valor de z correspondente ó valor de confianza.
- pq , a varianza da poboación; onde se $p = q = 0,5$ (a metade dos suxeitos responde afirmativamente e a outra negativamente), entón $pq = 0,25$ (cte.).

Para valores:

- Nivel de confianza do 95%.
- $N = 1500$.
- $e = 0,1$.
- $z = 1,96$ ($\alpha = 0,05$).
- $pq = 0,25$.

Entón:

$$n = \frac{1500}{1 + \frac{0,1^2(1500-1)}{(1,96^2)0,25}} \approx 90 \quad (4.2)$$

É dicir, precisaríamos que uns 90 gaiteiros (un 6 % da poboación) respondesen á enquisa.

Aínda tendo en conta o anterior, tendo evitado o condicionamento entre usuarios e vista a dispersión mostral, os resultados da enquisa parecen amosar unha forte correlación verdadeira, motivo polo cal tendemos a crer que o proxecto é comercialmente viable desde o punto de vista de interés do mercado no producto.

Capítulo 5

Análise de requisitos

Índice xeral

5.1. Determinación	76
5.1.1. Obxectivos	76
5.1.2. Alternativas	76
5.1.3. Restriccóns	76
5.2. Avaliación de alternativas e resolución de riscos	77
5.2.1. Análise de riscos	77
5.2.2. Prototipo 2	78
5.3. Desenvolvemento e validación do seguinte nivel do producto	78
5.3.1. Simulacións, modelos e programas de proba	78
5.3.2. Requisitos hardware, software e económicos	81
5.3.3. Validación de requisitos	99
5.4. Planificación da próxima fase ou ciclo	104
5.4.1. Planificación do deseño	104

NESTE capítulo exporase a análise de requisitos do proxecto baseándose no esquema proporcionado pola planificación inicial: desde a extracción e validación de requisitos ata a realización dunha enquisa entre os posibles usuarios, pasando por un segundo prototipo.

5.1. Determinación

5.1.1. Obxectivos

Establecerónse os obxectivos da fase de análise de requisitos do proxecto.

Obxectivos:

- Determinar os requisitos necesarios do deseño e implementación dunha gaita MIDI sen fíos en tempo real empregando software/hardware libre.

5.1.2. Alternativas

Establecerónse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se puidesen cumplir.

Alternativas:

- Se non é posible determinar os requisitos do proxecto, pode optarse por:
 1. Determinar soamente os requisitos daquelas partes que sexa posible.
 2. Cancelar e mudar de proxecto.

5.1.3. Restriccóns

Establecerónse restriccóns aplicables a ditos obxectivos.

- As propias restriccóns veñen dadas polo propio título do proxecto. A saber:
 1. Empregar o protocolo MIDI.
 2. Empregar tecnoloxía sen fíos.
 3. Empregar tempo real.
 4. Empregar software libre.
 5. Empregar hardware libre.
 6. E/ou as derivadas de calquera das súas alternativas.

5.2. Avaliación de alternativas e resolución de riscos

5.2.1. Análise de riscos

Determináronse os riscos que comportaban as distintas alternativas e as súas posibles solucións.

- Alternativas 1.

1. Riscos:

- a) Que nin a posteriori se poidan obter os requisitos das partes restantes.
- b) Que os requisitos obtidos a posteriori:
 - 1) Eleven moito os custes temporais, económicos, de esforzo e/ou de recursos.
 - 2) Sexan incompatibles cos xa obtidos.
- c) Que o tempo restante para a execución do proxecto non sexa suficiente.

2. Solucións:

- a) Aplicar algunha das subsolucións alternativas ou cancelar e mudar de proxecto.
 - b) Solucións propostas:
 - 1) Aplicar medidas de mitigación para que a planificación non se vexa afectada en extremo, ou cancelar e mudar de proxecto se fan inviable o mesmo.
 - 2) Reanalarizar os requisitos e refactorizar os mesmos ata obter un sistema compatible, ou cancelar e mudar de proxecto se non é posible.
 - c) Agardar a presentalo na seguinte convocatoria.
-

5.2.2. Prototipo 2

Como segundo prototipo optouse por realizar unha interface gráfica semi-funcional de desenvolvemento rápido que plasmara os requisitos básicos típicos de sistemas similares xa existentes coa fin de avaliar os mesmos e obter novos requisitos ou desbotar parte dos xa obtidos, así como ir obtendo parte do deseño e distribución dos elementos da interface.

A idea principal era ter algo tanxible co que tanto expertos coma outros usuarios puideran interactuar debido a que obter os requisitos en abstracto dun sistema tan complexo non é sinxelo. Ademais, a maioría das veces, tanto expertos coma usuarios en xeral, “non saben o que queren pero si o que non queren”, polo que resulta moito máis sinxelo darles algo que poder criticar que dé pé tanto a desbotar requisitos xa presentes como á obtención de novos requisitos.

Así mesmo, presentóuselles o primeiro prototipo (desenvolvido na primeira fase) para poder facer exactamente o mesmo cos requisitos hardware.

A continuación (figuras 5.1 a 5.5) preséntanse tódalas pantallas da interface á que se chegou mediante proceso de prototipado rápido e que foi posteriormente presentada ós expertos para a súa análise conxunta co proxectando.

5.3. Desenvolvemento e validación do seguinte nivel do producto

5.3.1. Simulacóns, modelos e programas de proba

As probas a realizar neste nivel do producto consistirán en verificar sobre o *Prototipo 2* todos aqueles requisitos software aplicables ó mesmo recollidos na especificación de requisitos, sendo ditas probas realizadas a tres bandas, é dicir, por parte do proxectando, dos expertos e de usuarios aleatorios.

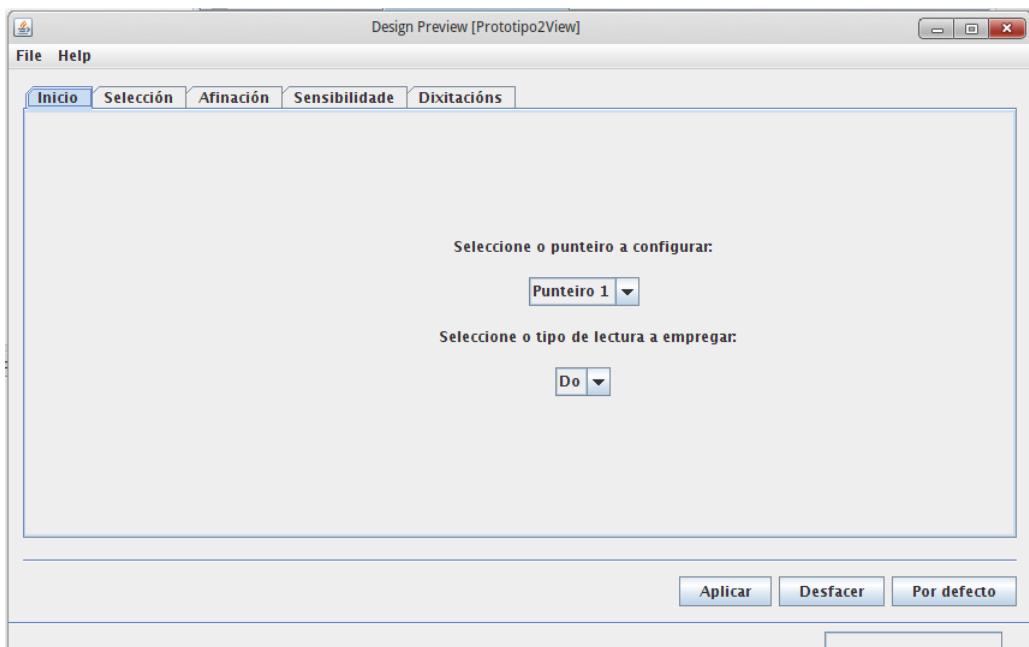


Figura 5.1: Prototipo 2: pantalla de inicio.

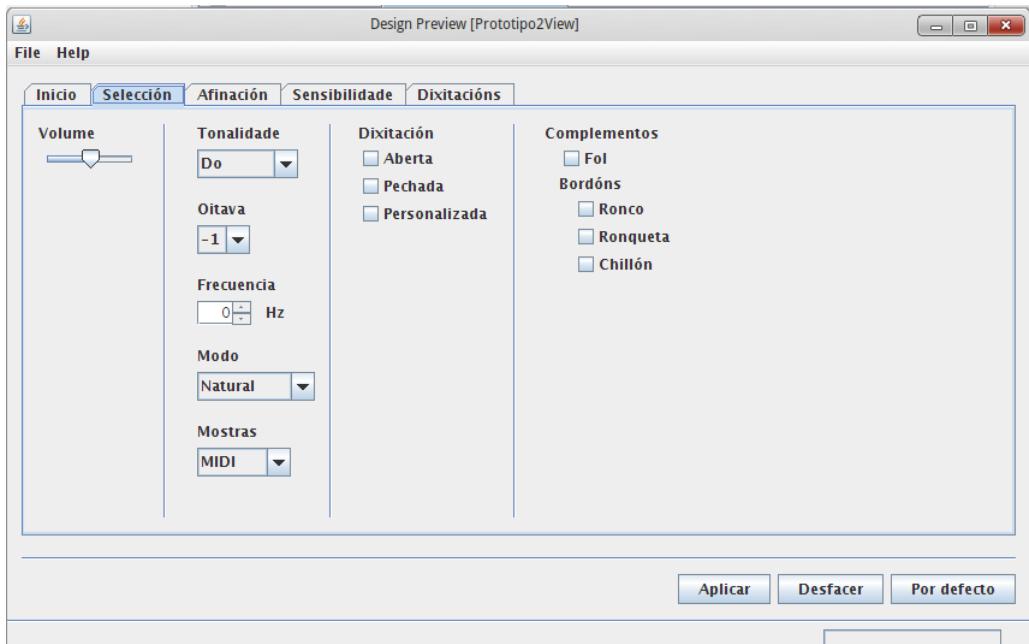


Figura 5.2: Prototipo 2: pantalla de selección.

80 5.3. Desenvolvemento e validación do seguinte nivel do producto

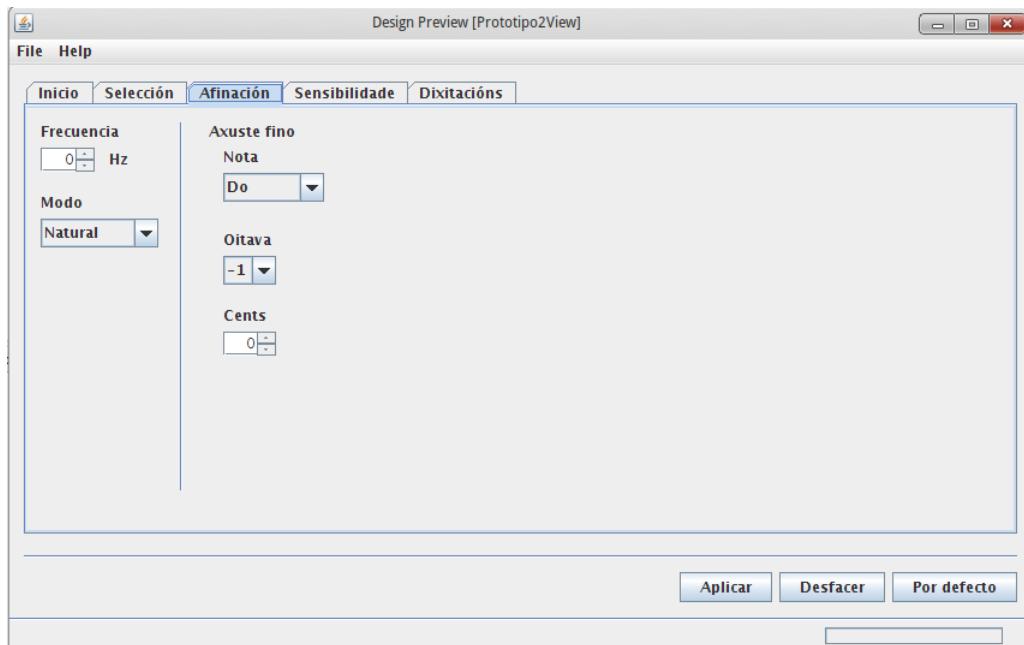


Figura 5.3: Prototipo 2: pantalla de afinación.

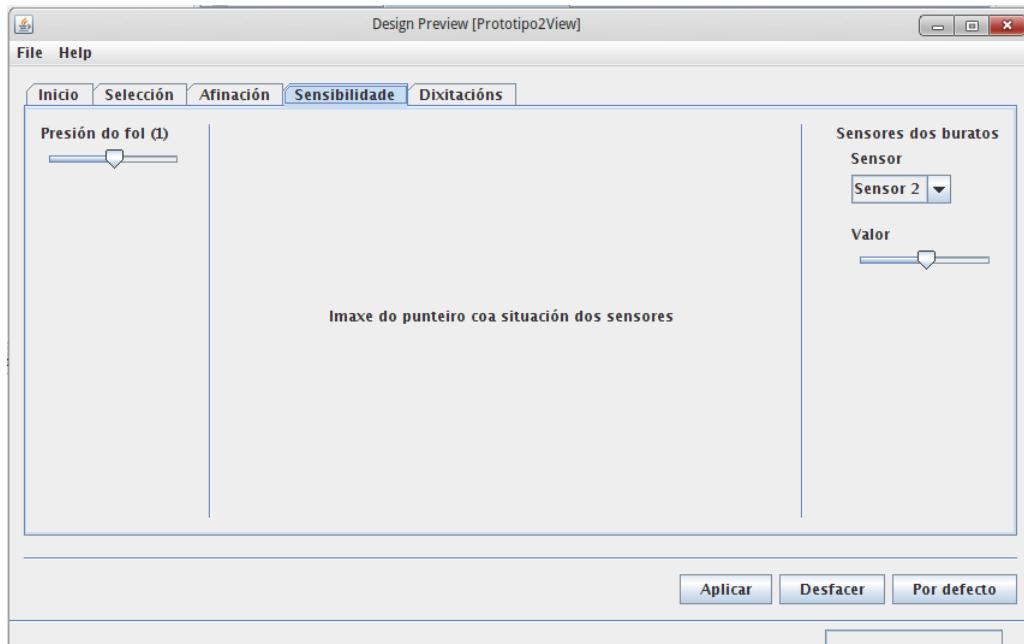


Figura 5.4: Prototipo 2: pantalla de sensibilidade.

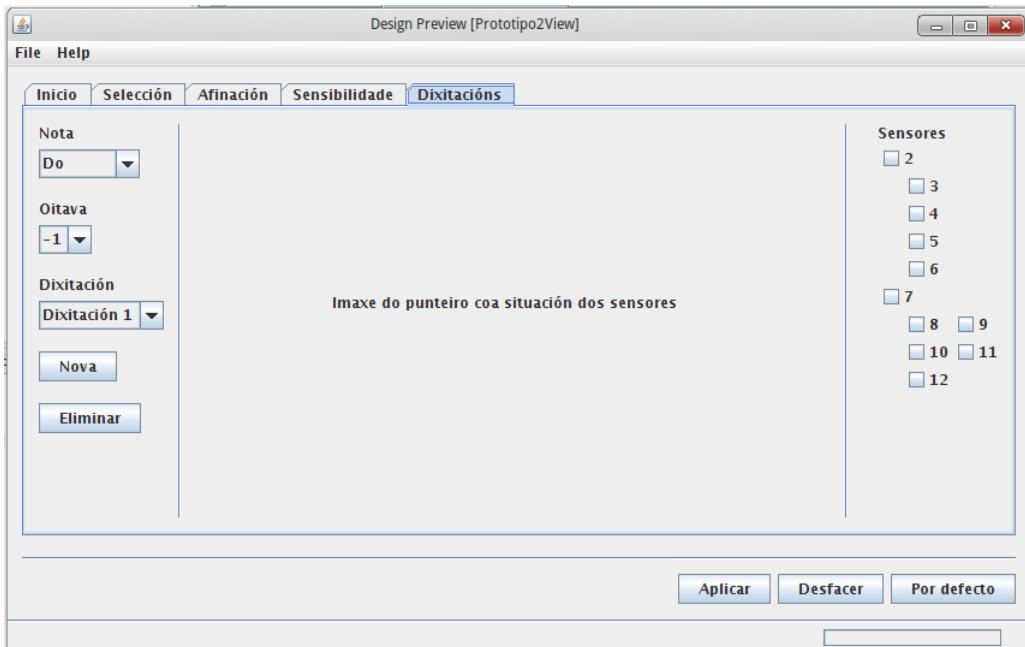


Figura 5.5: Prototipo 2: pantalla de dixitación.

5.3.2. Requisitos hardware, software e económicos

5.3.2.1. Extracción de requisitos por parte do proxectando

Para a extracción de requisitos por parte do proxectando tivéronse en conta múltiples factores, xa que o obter só unha boa imitación non abonda para que o producto final acabe tendo éxito. Así, entre os factores que se manexaron destacan:

- A análise exhaustiva do equivalente “analóxico” e da interacción co usuario.
- A análise exhaustiva dos productos comerciais xa existentes no mercado.
- O establecemento de fortes restriccións coa fin de lograr un producto de calidade.
- A aplicación da filosofía do Software Libre.
- A non discriminación de usuarios.
- O enfoque mercantil, é dicir, o establecemento dunha estratexia comercial.

Tendo en conta estes factores e outros non menos importantes e botando man do *Prototipo 1* e do *Prototipo 2*, nos que se foron plasmando todas as ideas que ían surxindo, chegouse á seguinte especificación de requisitos software, hardware e económicos.

■ Especificación de requisitos software:

- Principais:
 - Software libre.
 - ◊ Todo o código desenvolvido durante o proxecto será liberado baixo unha licencia libre.
 - ◊ Todo software que se empregue para a realización do proxecto será libre (sempre que sexa posible).
 - Aplicación multiplataforma.
 - ◊ A aplicación debe de ser portable a distintos sistemas operativos.
 - ◊ Non debe de ser necesario recompilala manualmente para o seu correcto funcionamento. Non debe de ter dependencia de tecnoloxías ou ferramentas que limiten a súa portabilidade.
 - ◊ Garantiza a uniformidade da aplicación e, polo tanto, reduce a curva de aprendizaxe entre plataformas.
 - ◊ Reduce a complexidade e os custes do proxecto.
 - ◊ Reduce a aparición de errores e facilita a súa localización e corrección.
 - Retardo mínimo.
 - ◊ O retardo existente entre a execución dunha nota e a súa audición debe de ser imperceptible ou aceptable polo usuario.
 - ◊ Isto implica o uso de tempo real.
 - ◊ Máis concretamente e, relacionándoo directamente co software, esixe o uso dun sistema operativo con soporte para tempo real.
 - Emprego da mesma configuración en entornos distintos.

- ◊ Non é de rigor ter que reconfigurar o punteiro de cada vez que o queremos empregar, nin ter que facelo se o empregamos nun equipo distinto ó habitual.
- ◊ Implica que a configuración debe ser gardada no punteiro.
- Uso de varios punteiros sobre a mesma aplicación e receptor.
 - ◊ Uso de varios punteiros sobre a mesma aplicación e receptor.
 - ◊ Non é de rigor ter que empregar varias instancias da aplicación e/ou varios equipos para poder emplegar varios punteiros.
- Regulación da presión do fol.
 - ◊ Debe de ser posible a regulación da presión necesaria para facer soar o punteiro.
 - ◊ Non tódolos usuarios apretan o mesmo.
- Regulación da sensibilidade dos sensores, tanto conxunta coma independentemente.
 - ◊ Non hai dúas persoas iguais e, incluso dentro da mesma persoa, non tódolos dedos son iguais, polo que é preciso poder regular os sensores para o correcto funcionamento do punteiro.
- Regulación de volume.
 - ◊ Debe ser posible regular o volume de saída do son, por motivos evidentes.
- Regulación da tonalidade ou nota base.
 - ◊ Debe ser posible tocar en calquera tonalidade sen limitación algunha de nota ou altura.
- Regulación da frecuencia base de afinación.
 - ◊ Debe ser posible regular a frecuencia base de afinación.
 - ◊ Dependendo da situación, é preciso afinar nunha frecuencia ou noutra.
- Regulación individual da frecuencia de afinación de cada nota.
 - ◊ Debe ser posible retocar individualmente a afinación de cada nota co fin de poder afinar correctamente cos bordóns “analóxicos” da gaita e/ou con outros instrumentos.
- Configuración da dixitación.

- Empregarase a dixitación da gaita galega (aberta, pechada).
- Darase a posibilidade de incluír outras dixitacións ou persoalizar as xa existentes.
- Tesitura ilimitada.
 - A maioría das opcións comerciais existentes limitan moito a tesitura, o que limita gravemente ó usuario á hora de interpretar unha peza.
 - Debe ser o propio usuario o que decida ónde están os seus límites, non o sistema.
- Opcionais:
 - Menús navegables dende o punteiro.
 - Facilita a tarefa de variar a configuración do punteiro sen ter que achegarse á computadora.
 - Deshabilitación do sensor de presión.
 - O sensor de presión emprégase para evaluar cándo debe de soar a gaita.
 - Sería interesante poder facelo tamén empregando unha “dixitación especial” que nos permitise poder tocar sen ter que facer uso do fol.
 - Existen moitos casos nos que isto sería unha opción moi cómoda.
 - “Vibrato continuo”.
 - Posibilidade aplicar a técnica de “vibrato continuo” presente noutros punteiros comerciais.
 - Inclusión de bordóns.
 - Ronco, ronqueta e chillón.
 - Sobre afinación natural.
 - Con posibilidade de aplicar cortes.
 - Uso de *samples* reais.
 - Inclusión da posibilidade do uso de *samples* pregravados no canto de MIDI.
 - Posibilidade de distintas afinacións e/ou toques.

■ Especificación de requisitos hardware:

- Principais:
 - Uso de hardware libre.
 - ◊ Todo o hardware empregado na construción do punteiro ha de ser libre.
 - Punteiro ó máis parecido a un punteiro “analóxico”.
 - ◊ Mínima intrusión, mínimo rechazo.
 - Uso de sensores capacitivos.
 - ◊ Son os más adecuados.
 - Mesmas dimensíons para tódolos buratos.
 - ◊ Facilita o uso e axuste dos sensores.
 - Conexión sen fíos.
 - ◊ Facilita a mobilidade e a comodiade á hora de empregar o punteiro.
 - ◊ Mínima intrusión, mínimo rechazo.
 - ◊ Pode verse condicionada á falta de espacio no punteiro, debido á necesidade de incorporar unha batería recargable.
- Opcionais:
 - Inclusión de botóns de selección.
 - ◊ Facilita a tarefa de variar a configuración do punteiro sen ter que achegarse á computadora.
 - ◊ Esta opción dependerá do espacio físico sobrante no interior do punteiro.
 - Entrega do producto en estuche de calidade.
 - ◊ Dar boa impresión incrementa a confianza no producto.
 - ◊ Dependerá dos requisitos económicos.
 - Entrega dun netbook/tablet ou similar debidamente preparado e configurado para empregar coa gaita.
 - ◊ Concepto “plug & play”.
 - ◊ A día de hoxe segue existindo unha ampla cota de que non está familiarizada coa informática básica, polo que sería interesante ofrecer dita opción.

- ◊ Dependerá dos requisitos económicos.

■ **Especificación de requisitos económicos:**

- Principais:
 - Saída ó mercado do modelo básico por debaixo dos 600 euros.
 - ◊ En ningún caso debería de sobrepasar os 600 euros, salvo inclusión de hardware adicional opcional.

5.3.2.2. Extracción de requisitos por parte de expertos

Para a extracción de requisitos por parte de expertos, preseleccionáronse dous recoñecidos expertos na materia entre os moitos existentes e coñecidos por parte do proxectando.

Xosé Luis “Lis” Latas Vilanova



Figura 5.6: Lis Latas.

Xosé Luís Latas Vilanova (Lugo, 1971) [27], coñecido nos ambientes gaiteirís como Lis Latas, pertence a esa nova e brillante xeración de artesáns en que conflúen todos os saberes e técnicas tradicionais de construcción da gaita a carón das más avanzadas pesquisas e investigacións destinadas a melloraren o instrumento nacional galego por excelencia. Así, apóis un estudo minucioso da construcción, tímbrica e afinación das gaitas galegas antigas, pasando por unha análise rigorosa das súas características sonoras na época actual, as gaitas que levan o selo Lis posúen un timbre de seu, propio, que combina o mellor da estética sonora de hai tempo coa minuciosidade na afinación, capaces de singularizaren positiva e acusticamente os instrumentos construídos no Obradoiro de Gaitas Lis Latas. E esa perfección na

afinación da escala da gaita foi conseguida combinando variábeis fundamentais que teñen incidencia directa nesa cualidade sonora: por un lado, mediante o estudo pormenorizado da localización exacta (en décimas de milímetro) en que debe ir un burato do punteiro; en segundo lugar, a través da comparación do diámetro entre os diferentes buratos entre si, atendendo á columna de ar; como terceiro factor a tener en conta, mediante unha análise milimétrica das dimensións da lonxitude do punteiro; aliás, a través dunha investigación rigorosa da súa precisa conicidade; e, finalmente, grazas a unha minuciosa análise dos ouvidos que se achán na base do punteiro e da relación que presentan entre si. Son, pois, uns instrumentos capaces de afinaren a 440 en calquera das súas notas, máxime a termos en conta o punteiro afinábel, verdadeira carta de presentación do Obradoiro e do cal falaremos más abaixo.

Con 4 anos inicia as súas primeiras clases de baile tradicional en *Candai*, na casa de *Tiopedro*, continuando no grupo *As Feiticeiras de Paradai* no que ingresa ós 7 anos de idade. Dous anos despois pasa a formar parte das escolas de *Cántigas e Frores de Lugo*, onde toma os primeiros contactos coa gaita galega. Ós 14 anos comeza a impartir clases de baile en diferentes parbularios e, xa con 17, entra a formar parte da primeira asociación de mestres de baile de Lugo. Un ano máis tarde nace unha nova asociación cultural na capital lucense, *Reviravolta*, da que é socio fundador. A partir deste momento comeza a dar clases de baile e gaita en diferentes asociacións da provincia de Lugo e o occidente de Asturias: *Antaruxas e Sorteiros da Fonsagrada*, *Ai-riños do Neira de Baralla*, *Xistra de Pedrafita do Cebreiro*, asociación de *Triacastela*, *Os Castros en Taramundi*, *Pola Vila de San Antolín de Ibias*, asociación de *San Martín de Oscos*, asociación do *Cádabo* e tamén no colexió da mesma localidade, asociación de veciños de *Conturiz*, ...

Realiza estudos universitarios na *Escola de Maxisterio Infantil* do campus lucense. Na súa ampla traxectoria profesional forma parte de

proxectos de diferente índole:

- Na actualidade forma parte do grupo folk *Reviravolta* de Lugo como director musical.
- Traballa durante 4 anos (1996-2000) na tenda musical *Arco Iris* de Lugo como técnico da sección de folk e reparador de instrumentos musicais.
- No ano 1997 produce o disco *O Miño non pasa por Escocia* do grupo *Reviravolta*.
- Colabora no disco *Razas* de Arturo Vaquero (1998). Participa na banda sonora da serie televisiva *Camiño de Santiago*, emitida en Telecinco no ano 1999.
- Comeza no mundo da construición de instrumentos no ano 1999 da man do ilustre gaiteiro, Pepe Vaamonde, pilar fundamental na súa evolución como artesán.
- Presenta o traballo discográfico *166.000 Xentes Libres* no *Lincoln Center* de Nova York (novembro do 2000).
- Colabora con diferentes grupos musicais: *Seare* de Sarria e *Os moricegos de Taramundi*.
- Realiza un curso de ebanistería impartido polo mestre *Delfín Blanco* no ano 2001.
- No ano 2002 patenta o primeiro punteiro afinable do mundo incorporando unha mellora á patente no 2005.
- Ademais do seu traballo como artesán, na actualidade segue impartindo clases de baile nalgunhas asociacións da provincia lucense e dirixindo o grupo folk *Reviravolta*.

Pepe Vaamonde Manteiga

Leva máis de 25 anos vinculado á gaita e á música folc. Con 15 anos empeza a compor e na década dos noventa gaña a maioría de concursos de Galicia como gaiteiro solista (*I Certame Pelegrín, Soutelo de Montes*, tres edicións do *Constantino Bellón* -o concurso máis



Figura 5.7: Pepe Vaamonde.

importante a nivel galego-).

É sobre todo un gaiteiro cun son propio, que é capaz de interpretar as pasaxes más virtuosas e os más emotivos cunha mestría inaudita.

Actualmente compaxina o seu traballo en *Pepe Vaamonde Grupo (PVG)* [28] co seu labor docente en varias asociacións folclóricas e no Centro de Música Fingoi de Lugo.

Ambos expertos traballan xuntos no obradoiro de construción de gaitas *Lis Latas*, feito polo cal están acostumados a colaborar entre eles e polo que existe certa simbiose que axudará á hora da extracción de requisitos. Isto, xunto cos coñecementos previo de ambos por parte do proxectando, axudará enormemente na comprensión entre tódalas partes e na chegada a bo porto do procedemento.

O procedemento seguido para a extracción de requisitos por parte dos expertos consistiu na exposición por parte do proxectando dos requisitos previamente

extraídos, apoiándose no *Prototipo 1* e *Prototipo 2*, para posteriormente discutilos e avalialos un por un cos expertos, dándoos finalmente todos por bons. Así mesmo, instouse ós expertos a avaliar se faltaba ou sobraba algúns, dando lugar unicamente á inclusión do seguinte requisito hardware:

- Superficie dos sensores colocada por debaixo da zona de contacto.
 - Dá sensación de burato.
 - Facilita a adaptación e o uso do punteiro.
 - Mínima intrusión, mínimo rechazo.

5.3.2.3. Extracción de requisitos por parte dos usuarios

Como se comentou no capítulo 4, para a extracción de requisitos a través dos usuarios creouse unha enquisa anónima dispoñible en liña [23] na que se preguntaba sobre datos persoais non identificativos (idade, procedencia), o seu perfil como gaiteiro, coñecementos sobre as gaitas MIDI, caracterísiticas (fixadas de antemán en extraccións previas) preferibles dunha gaita MIDI e a súa prioridade, características propias a engadir e a súa prioridade e un pequeno estudio económico. A redacción da enquisa pode consultarse tamén no capítulo 4.

Como se pode observar, na enquisa non se indicaron tódolos requisitos extraídos anteriormente, senón que se tratou de reflexar os más importantes (para dar unha idea básica das funcionalidades do producto) mesturados con outros non tan obvios (para poder confirmalos, aprazalos ou desbotalos). Ademais, non se explicou o motivo de ningún dos mesmos, para que o enquiskado tivera que reflexionar por si mesmo o motivo da súa inclusión. E ó mesturar os requisitos básicos cos opcionais podíanse descartar máis facilmente respuestas incoherentes (por exemplo, se un enquiskado incluía os opcionais e non incluía os básicos).

Así mesmo, é de facer notar que as respuestas a dita enquisa non son públicas, polo que ningún usuario previo pode influír coas súas respuestas sobre usuarios posteriores. Evidentemente, esta medida está enfocada a evitar un posible problema de falsa correlación forte nas respuestas, o que podería invalidar totalmente a enquisa.

Os resultados completos poden consultarse no apéndice B.

Os resultados relativos ós requisitos foron os seguintes (figuras 5.8 a 5.12).

Tendo en conta estes resultados, podemos afirmar que o perfil xeral do gaiteiro que respondeu a enquisa:

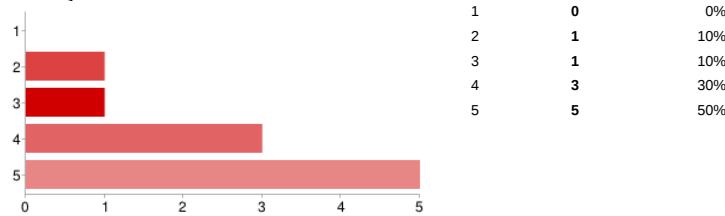
- Comprende perfectamente a importancia da ausencia de retardos.
- Non ten de todo claro o porqué da salvagarda da configuración entre usos.
- Considera importante que a presión do fol sexa regulable.
- Considera importante a regulación independente dos sensores dos dedos áinda que non o ten de todo claro.
- Considera indispensable que a afinación base sexa regulable.
- Tamén considera indispensable que dita afinación se poida regular por nota.
- A dixitación personalizable parece darlle igual, áinda que a prefire.
- Non ten moi claro que a posibilidade de deshabilitar o sensor do fol sexa un requisito esixible.
- Sabe que o vibrato é un requisito importante.
- O mesmo pasa cos glisandos, áinda que non tan claramente.
- Élle indiferente que teña bordóns ou non.
- Pide claramente uns *samples* de calidade.
- Quere un punteiro con forma tradicional.
- E que dito punteiro sexa acoplable a un fol calquera.
- Cre indispensable que se poida conectar a un ordenador.
- E que esa conexión se poida realizar sen fíos de por medio.

92 5.3. Desenvolvemento e validación do seguinte nivel do producto

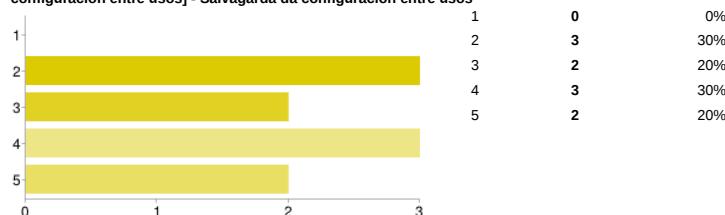
Editar formulario - [Enquisa] - Google Docs

<https://docs.google.com/spreadsheet/gform?key=0AhN...>

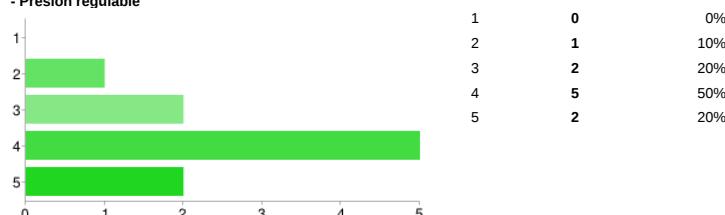
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Ausencia de retardos] - Ausencia de retardos



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Salvagarda da configuración entre usos] - Salvaguarda da configuración entre usos

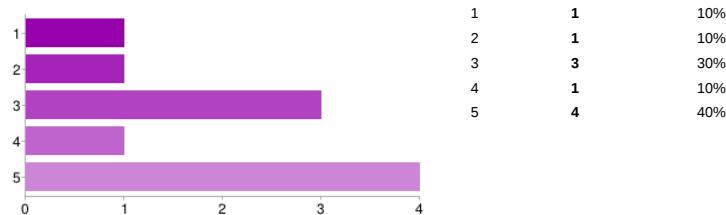


¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Presión regulable] - Presión regulable

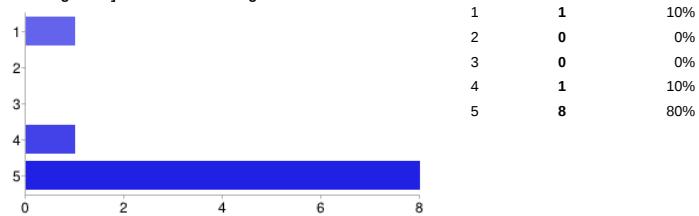


¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Sensores dos dedos regulables independientemente] - Sensores dos dedos regulables independientemente

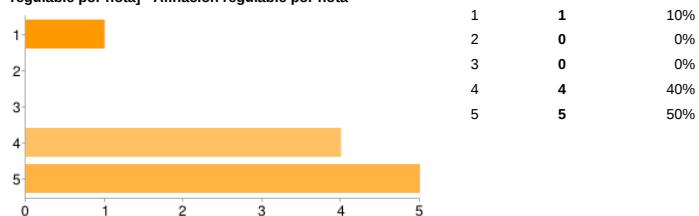
Editar formulario - [Enquisa] - Google Docs

<https://docs.google.com/spreadsheet/gform?key=0AhN...>

¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Afinación base regulable]



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Afinación regulable por nota]



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Dixitación personalizable]

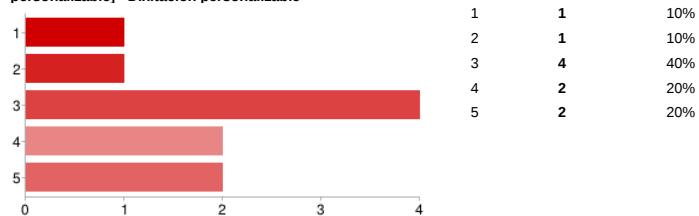


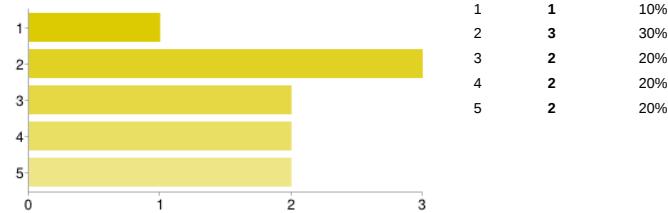
Figura 5.9: Resultados da extracción de requisitos por parte dos usuarios (p. 5).

94 5.3. Desenvolvemento e validación do seguinte nivel do producto

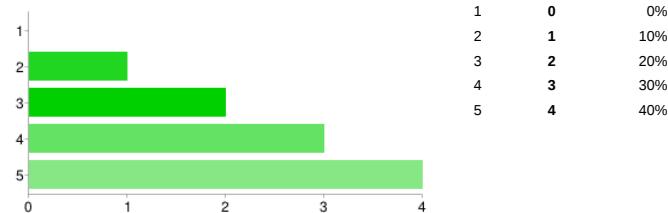
Editar formulario - [Enquisa] - Google Docs

<https://docs.google.com/spreadsheet/gform?key=0AhN...>

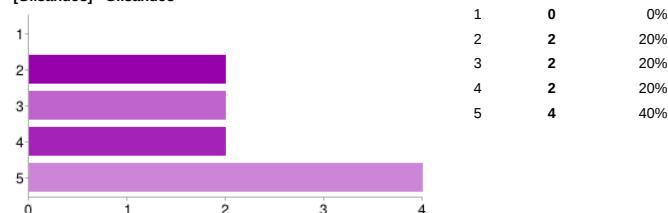
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Sensor da presión do fol deshabilitable] - Sensor da presión do fol deshabilitable



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Vibrato] - Vibrato



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Glisandos] - Glisandos



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Bordóns] - Bordóns

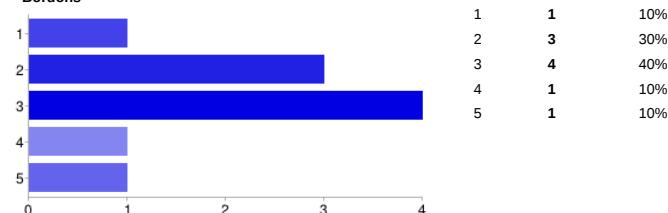
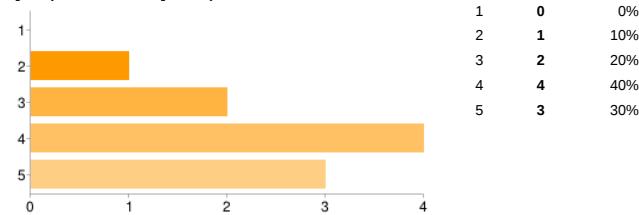


Figura 5.10: Resultados da extracción de requisitos por parte dos usuarios (p. 6).

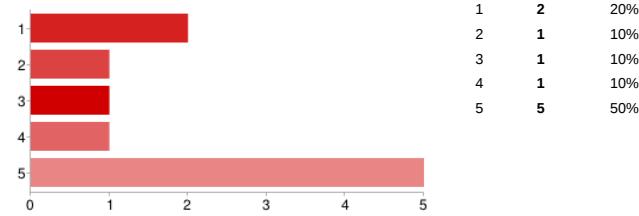
Editar formulario - [Enquisa] - Google Docs

<https://docs.google.com/spreadsheet/gform?key=0AhN...>

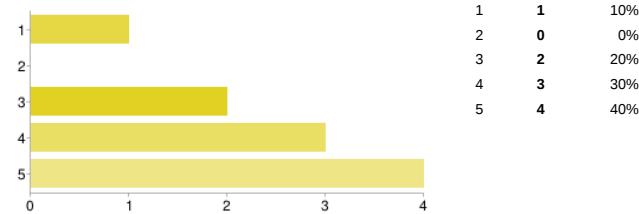
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
 [Samples de calidade] - Samples de calidade



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
 [Punteiro con forma tradicional] - Punteiro con forma tradicional



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
 [Punteiro acopiable a un fol calquera] - Punteiro acopiable a un fol calquera



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
 [Posibilidade de conexión a un ordenador] - Posibilidade de conexión a un ordenador



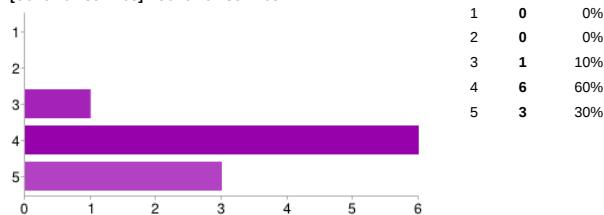
Figura 5.11: Resultados da extracción de requisitos por parte dos usuarios (p. 7).

96 **5.3. Desenvolvemento e validación do seguinte nivel do producto**

Editar formulario - [Enquisa] - Google Docs

<https://docs.google.com/spreadsheet/gform?key=0AhN...>

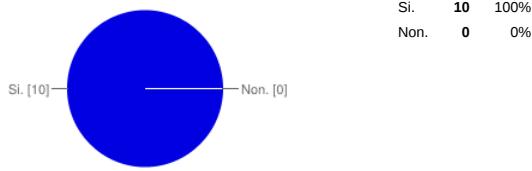
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
[Conexión sen fíos] - Conexión sen fíos



Indica outras características que che gustaría que tivese, ordeadas por prioridade.

-Cascos ou auriculares de escutar música que se conecten a gaita para oír ti so, sin que os demais as oian ou que non oian case nada. Tesitura do instrumento de 2 octavas cromáticas completas e a sensible inferior (Si4-Dó7) idéntico tamaño e forma que un punteiro normal que sexa cómoda e non moi grande o fol

¿Estarías disposto a mercar unha gaita MIDI coas características anteriores (incluidas as que tí mesmo engadiches)?



De ser así, ¿canto estarías disposto a pagar por unha?

1750 € 500 1500 500 1000 1500 500 150 300 500

En caso contrario, ¿por que non?

Sabendo que as gaitas MIDI presentes hoxe en día no mercado oscilan entres os 275 € e os 2800 € e que a maior parte delas non satisfacen as características anteriores ¿canto estarías disposto a pagar por unha que as inclúise todas ou case todas?

3400 € 500 1500 500 1000 1500 1000 150 350 750

Number of daily responses

Figura 5.12: Resultados da extracción de requisitos por parte dos usuarios (p. 8).

Como extracción de requisitos propia dos usuarios, atopámonos con tres peticions interesantes:

- Inclusión dunha toma para auriculares.
- Tesitura de dúas oitavas completas máis a sensible inferior.
- Idéntico tamaño e forma que un punteiro normal.

Todas estaban contempladas xa nos requisitos obtidos anteriormente,

- a primeira iría incluída no ordenador ou equivalente ó que se conectase
- a segunda está xa contemplada no *Prototipo 2*, como se pode comprobar,
- e a terceira tamén,

pero pasaron a formar parte más específicamente dos requisitos, para non pasalos por alto en futuras consultas da documentación.

Así mesmo, resulta interesante comprobar que os posibles usuarios se tomaron a enquisa en serio e incluso se preocuparon por achegar posibles requisitos moi específicos e que non son nada fáciles de detectar a priori.

E centrándose xa na parte final da enquisa, o estudio económico, conta con datos moi interesantes.

Á hora de respestar á pregunta sobre se mercarían unha gaita MIDI coas caracterísiticas anteriores (incluidas as propostas polos propios usuarios) todos respstaron afirmativamente (incluso aqueles que albergaban dúbidas ou non estaban totalmente interesados), o que nos fai pensar que imos por bo camiño.

Por este motivo, a pregunta sobre as razóns para non mercala quedou sen resposta.

A seguinte pregunta, en connexión coa última, estaban formuladas e ordeñadas de maneira tal que formasen unha “pregunta trampa” co fin de:

- Por unha banda, saber o que estarían dispostos a pagar a cegas por un producto destas características.
- Por outra, saber o que estarían dispostos a pagar polo mesmo, dentro dun entorno acotado, é dicir, con información previa.
- E, por último e quizáis o máis importante, ser capaces de detectar posibles *outliers* que nos puidesen levar a engano.

A pregunta intermedia tamén axudaba no cometido, pois conseguía distraer ó usuario entre unha e outra e darlle tempo para volver a pensar antes de responder novamente.

Avaliando as respostas a ditas preguntas obtívose información moi valiosa:

- Que as respostas apenas varían dunha pregunta á outra.
- Que as respostas á primeira pregunta estaban xa acotadas dentro do intervalo exposto posteriormente.
- Que parecen existir dúas categorías diferenciadas con distinto centro. Intentando darlle explicación a dito suceso e fixándose na última petición de requisitos por parte dos usuarios (na que se fala do fol), chegouse á conclusión de que o más probable é que os usuarios que caen dentro da primeira categoría (precio menor) estaban a pensar únicamente no punteiro e os da segunda, na gaita completa, polo que, sen querer, obtivérонse requisitos económicos para dúas configuracións distintas.

Centrándonos na primeira dasas configuracións, actualizouse un dos requisitos económicos da seguinte maneira:

- Saída ó mercado do modelo básico por debaixo dos 600 euros.
 - *O estudio de viabilidade indica que sería aconsellable que o precio estivese por debaixo dos 500 euros.*
 - En ningún caso debería de sobrepasar os 600 euros, salvo inclusión de hardware adicional opcional (ver requisitos hardware).

E volvendo á información aportada polas respostas anteriores:

- Hai un claro caso de *outlier* en canto ás respostas do primeiro usuario no que á parte económica se refire. Con respecto á primeira pregunta, dá a cantidade máis alta de todas e, con respecto á segunda, ademais de ser a cantidade máis alta, sáese fóra do intervalo, o que acentúa máis se cabe a súa singularidade. Intentando buscarlle explicación a isto e ó mesmo tempo, saber se tiñan que desbotar as súas respostas, estudiamos o caso en máis profundidade. O resto das súas respostas non presentaban anomalías. Sen embargo, ó fixarse na resposta á primeira pregunta, detectamos que se trata dunha persoa de 14 anos, e polo tanto a priori sen experiencia en temas económicos, polo que é normal que as súas respostas no que a requisitos económicos se refire, sexan pouco acertadas. Por este motivo, decidiuse desbotar as súas respostas á parte económica da enquisa, mantendo iso si, as demás.

Para rematar o apartado, volver comentar unha vez máis que dado o tamaño mostral o estudio resulta non significativo, pero tendo evitado o condicionamento entre usuarios e vista a dispersión mostral, os resultados da enquisa parecen amosar unha forte correlación verdadeira, motivo polo cal tendemos a crer que a especificación de requisitos parece acorde ó que demanda o mercado.

5.3.3. Validación de requisitos

As probas a realizar neste nivel do producto consistiron en verificar sobre os dous prototipos previos todos aqueles requisitos software aplicables ós mesmos recollidos na especificación de requisitos, verificando por parte do proxectando e dos expertos que o producto é correcto.

Non se detectaron inconformidades por ningunha das partes, polo que se deu por correcta a especificación de requisitos, que finalmente quedou como segue:

- **Especificación de requisitos software:**
 - Principais:
 - Software libre.
-

- ◊ Todo o código desenvolvido durante o proxecto será liberado baixo unha licencia libre.
- ◊ Todo software que se empregue para a realización do proxecto será libre (sempre que sexa posible).
- Aplicación multiplataforma.
 - ◊ A aplicación debe de ser portable a distintos sistemas operativos.
 - ◊ Non debe de ser necesario recompilala manualmente para o seu correcto funcionamento. Non debe de ter dependencia de tecnoloxías ou ferramentas que limiten a súa portabilidade.
 - ◊ Garantiza a uniformidade da aplicación e, polo tanto, reduce a curva de aprendizaxe entre plataformas.
 - ◊ Reduce a complexidade e os custes do proxecto.
 - ◊ Reduce a aparición de erros e facilita a súa localización e corrección.
- Retardo mínimo.
 - ◊ O retardo existente entre a execución dunha nota e a súa audición debe de ser imperceptible ou aceptable polo usuario.
 - ◊ Isto implica o uso de tempo real.
 - ◊ Máis concretamente e, relacionándoo directamente co software, esixe o uso dun sistema operativo con soporte para tempo real.
- Emprego da mesma configuración en entornos distintos.
 - ◊ Non é de rigor ter que reconfigurar o punteiro de cada vez que o queremos empregar, nin ter que facelo se o empregamos nun equipo distinto ó habitual.
 - ◊ Implica que a configuración debe ser gardada no punteiro.
- Uso de varios punteiros sobre a mesma aplicación e receptor.
 - ◊ Uso de varios punteiros sobre a mesma aplicación e receptor.
 - ◊ Non é de rigor ter que empregar varias instancias da aplicación e/ou varios equipos para poder empregar varios punteiros.
- Regulación da presión do fol.

- ◊ Debe de ser posible a regulación da presión necesaria para facer soar o punteiro.
- ◊ Non tódolos usuarios apretan o mesmo.
- Regulación da sensibilidade dos sensores, tanto conxunta coma independentemente.
 - ◊ Non hai dúas persoas iguais e, incluso dentro da mesma persoa, non tódolos dedos son iguais, polo que é preciso poder regular os sensores para o correcto funcionamento do punteiro.
- Regulación de volume.
 - ◊ Debe ser posible regular o volume de saída do son, por motivos evidentes.
- Regulación da tonalidade ou nota base.
 - ◊ Debe ser posible tocar en calquera tonalidade sen limitación algunha de nota ou altura.
- Regulación da frecuencia base de afinación.
 - ◊ Debe ser posible regular a frecuencia base de afinación.
 - ◊ Dependendo da situación, é preciso afinar nunha frecuencia ou noutra.
- Regulación individual da frecuencia de afinación de cada nota.
 - ◊ Debe ser posible retocar individualmente a afinación de cada nota co fin de poder afinar correctamente cos bordóns “analóxicos” da gaita e/ou con outros instrumentos.
- Configuración da dixitación.
 - ◊ Empregarase a dixitación da gaita galega (aberta, pechada).
 - ◊ Darase a posibilidade de incluír outras dixitacións ou persoalizar as xa existentes.
- Tesitura ilimitada.
 - ◊ Alomenos dúas oitavas completas máis a sensible inferior.
 - ◊ A maioría das opcións comerciais existentes limitan moito a tesitura, o que limita gravemente ó usuario á hora de interpretar unha peza.
 - ◊ Debe ser o propio usuario o que decida ónde están os seus límites, non o sistema.

- Opcionais:
 - Menús navegables dende o punteiro.
 - ◊ Facilita a tarefa de variar a configuración do punteiro sen ter que achegarse á computadora.
 - Deshabilitación do sensor de presión.
 - ◊ O sensor de presión emprégase para evaluar cando debe de soar a gaita.
 - ◊ Sería interesante poder facelo tamén empregando unha “ditación especial” que nos permitise poder tocar sen ter que facer uso do fol.
 - ◊ Existen moitos casos nos que isto sería unha opción moi cómoda.
 - “Vibrato continuo”.
 - ◊ Posibilidade aplicar a técnica de “vibrato continuo” presente noutros punteiros comerciais.
 - Inclusión de bordóns.
 - ◊ Ronco, ronqueta e chillón.
 - ◊ Sobre afinación natural.
 - ◊ Con posibilidade de aplicar cortes.
 - Uso de *samples* reais.
 - ◊ Inclusión da posibilidade do uso de *samples* pregravados no canto de MIDI.
 - ◊ Posibilidade de distintas afinacións e/ou toques.

■ **Especificación de requisitos hardware:**

- Principais:
 - Uso de hardware libre.
 - ◊ Todo o hardware empregado na construción do punteiro ha de ser libre.
 - Punteiro ó máis parecido a un punteiro “analóxico”.
 - ◊ Mínima intrusión, mínimo rechazo.
 - Uso de sensores capacitivos.

- ◊ Son os más adecuados.
- Mesmas dimensíóns para tódolos buratos.
 - ◊ Facilita o uso e axuste dos sensores.
- Conexión sen fíos.
 - ◊ Facilita a mobilidade e a comodiade á hora de empregar o punteiro.
 - ◊ Mínima intrusión, mínimo rechazo.
 - ◊ Pode verse condicionada á falta de espacio no punteiro, debido á necesidade de incorporar unha batería recargable.
- Inclusión dunha toma para auriculares.
 - ◊ Debe permitírselle ó usuario poder tocar e escoritarse sen ter por qué causar molestias ó resto de persoas ó seu arredor.
 - ◊ Dita toma para auriculares xa debería de ir incluída no soporte ó que se conecte o punteiro, polo que non debería de plantear ningún problema.
- Opcionais:
 - Inclusión de botóns de selección.
 - ◊ Facilita a tarefa de variar a configuración do punteiro sen ter que achegarse á computadora.
 - ◊ Esta opción dependerá do espacio físico sobrante no interior do punteiro.
 - Entrega do producto en estuche de calidade.
 - ◊ Dar boa impresión incrementa a confianza no producto.
 - ◊ Dependerá dos requisitos económicos.
 - Entrega dun netbook/tablet ou similar debidamente preparado e configurado para empregar coa gaita.
 - ◊ Concepto “plug & play”.
 - ◊ A día de hoxe segue existindo unha ampla cota de que non está familiarizada coa informática básica, polo que sería interesante ofrecer dita opción.
 - ◊ Dependerá dos requisitos económicos.

■ Especificación de requisitos económicos:

- Principais:
 - Saída ó mercado do modelo básico por debaixo dos 600 euros.
 - ◊ O estudio de viabilidade indica que sería aconsellable que o precio estivese por debaixo dos 500 euros.
 - ◊ En ningún caso debería de sobrepasar os 600 euros, salvo inclusión de hardware adicional opcional.

5.4. Planificación da próxima fase ou ciclo

5.4.1. Planificación do deseño

A continuación (figura 5.13) exponse a planificación inicial do ciclo de deseño.

Name	Start	End
Deseño	11/25/11	1/17/12
Determinación 3	11/25/11	11/30/11
Obxectivos 3	11/25/11	11/28/11
Alternativas 3	11/28/11	11/29/11
Restriccóns 3	11/29/11	11/30/11
Avaliación de alternativas e resolución de riscos 3	11/30/11	12/8/11
Análise de riscos 3	11/30/11	12/1/11
Prototipo 3	12/1/11	12/8/11
Prototipo hardware 3	12/1/11	12/5/11
Prototipo software 3	12/1/11	12/8/11
Desenvolvemento e validación do seguinte nivel do producto 3	12/8/11	1/3/12
Simulacións, modelos e programas de proba 3	12/8/11	12/15/11
Deseño do producto hardware e software	12/15/11	12/29/11
Deseño do producto hardware	12/15/11	12/22/11
Deseño do producto software	12/22/11	12/29/11
Verificación e validación do deseño	12/29/11	1/3/12
Planificación da próxima fase ou ciclo 3	1/3/12	1/17/12
Planificación do desenvolvemento	1/3/12	1/17/12

Figura 5.13: Planificación inicial do ciclo de deseño.

Total: 148 horas.

Capítulo 6

Deseño do sistema

Índice xeral

6.1. Determinación	106
6.1.1. Obxectivos	106
6.1.2. Alternativas	106
6.1.3. Restriccóns	106
6.2. Avaliación de alternativas e resolución de riscos	107
6.2.1. Análise de riscos	107
6.2.2. Prototipo 3	107
6.3. Desenvolvemento e validación do seguinte nivel do producto	130
6.3.1. Simulacións, modelos e programas de proba	130
6.3.2. Deseño do producto hardware e software	130
6.3.3. Verificación e validación do deseño	131
6.4. Planificación da próxima fase ou ciclo	133
6.4.1. Planificación do desenvolvemento	133

NESTE capítulo exporase o deseño do proxecto baseándose no esquema proporcionalado pola planificación inicial: desde o deseño software e hardware do sistema ata o estudo das posibles plataformas hardware, pasando por un terceiro prototipo.

6.1. Determinación

6.1.1. Obxectivos

Establecerónse os obxectivos da fase de deseño do proxecto.

- Obter o deseño dunha gaita MIDI sen fíos en tempo real empregando software/hardware libre.

6.1.2. Alternativas

Establecerónse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se puidesen cumplir.

- Se non é posible obter o deseño do proxecto, pode optarse por:
 1. Determinar soamente o deseño daquelas partes que sexa posible.
 2. Cancelar e mudar de proxecto.

6.1.3. Restriccóns

Establecerónse restriccións aplicables a ditos obxectivos.

- As propias restriccións veñen dadas polo propio título do proxecto. A saber:
 1. Empregar o protocolo MIDI.
 2. Empregar tecnoloxía sen fíos.
 3. Empregar tempo real.
 4. Empregar software libre.
 5. Empregar hardware libre.
 6. E/ou as derivadas de calquera das súas alternativas.

6.2. Avaliación de alternativas e resolución de riscos

6.2.1. Análise de riscos

Determináronse os riscos que comportaban as distintas alternativas e as súas posibles solucións.

- Alternativas 1.

1. Riscos:

- a) Que o deseño das partes restantes obtido a posteriori:
 - 1) Dé como resultado un deseño de mala calidade.
 - 2) Sexa difícilmente factorizable.
 - 3) Eleven moito os custos temporais, económicos, de esforzo e/ou de recursos.
- b) Que o tempo restante para a execución do proxecto non sexa suficiente.

2. Solucións:

- a) Solucións propostas:
 - 1) Redeseñar.
 - 2) Factorizar por partes abordables ou redeseñar e reimplementar tendo en conta o deseño inherente da implementación previa.
 - 3) Aplicar medidas de mitigación para que a planificación non se vexa afectada en extremo, ou cancelar e mudar de proxecto se fan inviable o mesmo.
- b) Agardar a presentalo na seguinte convocatoria.

6.2.2. Prototipo 3

Como terceiro prototipo optouse por realizar tanto un prototipo hardware coma un prototipo software mediante os cales se puidese obter o deseño completo de alto nivel do sistema.

6.2.2.1. Prototipo hardware

Para a realización desta tarefa era preciso saber tanto o hardware coma o software de deseño a empregar, polo que foi precisa unha busca previa.

En canto ó hardware se refire, dita busca previa foi constante e extendeuse desde antes da propia formalización do proxecto coma tal, ata a realización desta tarefa; feito entendible, tendo en conta que o proxectando levaba anos coa idea de realizar dito proxecto por libre. Por non ser unha tarefa que requirise dun considerable esforzo diario durante o periodo que comprende ó proxecto, non se materializou coma tal na planificación, senón que se entendeu coma unha tarefa paralela implícita. Digamos que foi algo así coma “ler o xornal tódolos días”: non deixa de ser unha tarefa diaria, pero non leva aparexado un esforzo reseñable.

A lista de hardware avaliado (en base ás follas de especificacións dispoñibles) foi bastante extensa e diversa. Por orde alfabética:

- Arduino [29] (figura 6.1)
 - Plataforma de hardware libre por excelencia.
 - Prestacións reducidas, pero más ca suficientes.
 - Bo prezo.
 - Dimensións perfectas.
 - Alta dispoñibilidade. Tamén de componentes periféricos.
 - Soporte completo e de calidade.
- BeagleBone [30] (figura 6.2)
 - Boas prestacións.
 - Custo elevado, aínda que asumible.
 - Tamaño asumible.
 - Boa dispoñibilidade.
 - Falta de componentes periféricos.
- Clons especialiados de Arduino.

- Xduino.
 - Características similares ás de Arduino, pero adaptados para outras plataformas hardware/software e/ou especialiados para certos usos demasiado específicos.
 - Soporte máis ben escaso.
 - En certos casos, falta de componentes periféricos.
- PandaBoard [31] (figura 6.3)
 - Excelentes prestacións.
 - Custo moi elevado.
 - Dimensións elevadas.
 - Boa dispoñibilidade.
 - Existencia de componentes periféricos.
 - Compatibilidade con Arduino.
 - Tecnoloxía QTouch de Atmel [32] (figura 6.4)
 - Tecnoloxía punteira (consumo inferior ós 17 uA, velocidade de resposta de 12,6 ms, calibración automática).
 - Custo elevado.
 - Só dispoñible para maioristas.
 - Aínda que embebible, a priori, non o suficiente.
 - Alta dependencia do IDE de programación de Atmel.
 - Chegouse a mercar un kit de avaliación.
 - Raspberry Pi [33] (figura 6.5)
 - Boas prestacións.
 - Boa relación calidade/prezo.
 - Tamaño asumible.
 - Baixa dispoñibilidade.
 - Falta de componentes periféricos.



Figura 6.1: Arduino.

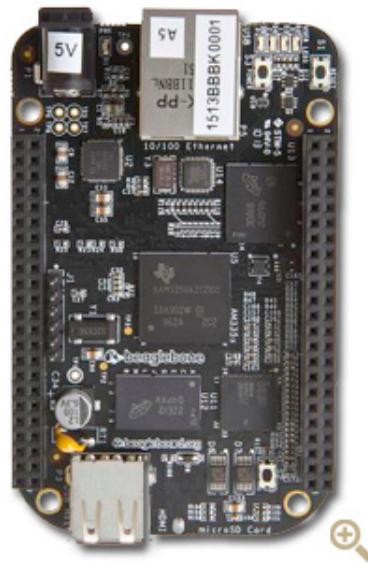


Figura 6.2: BeagleBone.

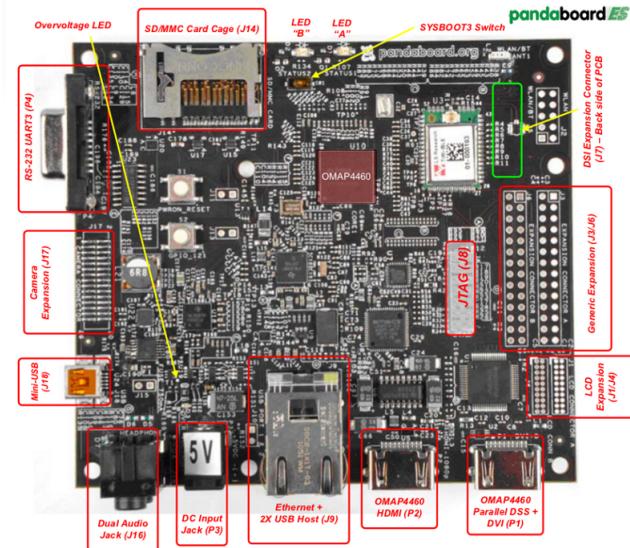


Figura 6.3: PandaBoard.



Figura 6.4: Atmel QTouch.



Figura 6.5: Raspberry Pi.

Tendo en conta os parámetros analizados e os requisitos do proxecto, determinouse que a plataforma Arduino¹ ofrecía a mellor solución.

Unha vez escollida a plataforma, había que, en base ó *Prototipo 1*, decidir qué componentes se necesitaban e cómo conectalos, é dicir, un deseño hardware inicial básico.

A mellor maneira de coñecer as distintas posibilidades existentes era visitar a sección de hardware da páxina oficial de Arduino [29], a tenda do maior fabricante de componentes para Arduino, de nome *SparkFun* [34] e as páxinas das dúas tendas oficiais principais dispoñibles en España, *BricoGeek* [35] e *Cooking Hacks* [36].

Para intentar acotar o hardware necesario, fíxose uso do *Prototipo 1*, que constaba de tres seccións ben diferenciadas:

- Sensor de presión.
- Sensores capacitivos.
- Sistemas de recepción, procesamento, almacenamento, transmisión e alimentación.

Para cada unha destas seccións precisaba de un ou varios componentes, polo que se abordaron por partes.

¹Para máis información sobre dita plataforma, consúltese o apéndice C.

No tocante á primeira sección, precisábase dun sensor de presión que cumprise coas seguintes características:

- De dimensíóns moi reducidas.
- Que empregase un protocolo que non tivera problemas coas distancias.
- Cunha boa sensibilidade e que traballe con valores de presión elevados.
- De baixo consumo.
- Cun tempo de resposta moi pequeno.
- Tamén estaría ben que fose somerxible (aínda que isto é solventable).

Como sobre sensores non hai información na páxina oficial de Arduino, tocou buscar na páxina de SparkFun, na que se pode ver que a mellor opción dispoñible era o *BMP085* [37] (figura 6.6). Mirando tamén nas tendas españolas, estaba tamén dispoñible.

Ademais da súa distribución en territorio nacional, o deseño da placa de SparkFun facilita moito o traballo con el en sistemas modulares, de prototipado rápido e onde non exista posibilidade de soldar compoñentes SMD/BGA facilmente.

Fixándose na súa folla de especificacións, vese que conta coas seguintes características:

- Dimensíóns: 16,5 x 16,5 mm.
 - Consumo: 5 uA (máx.).
 - Precisión: 300 a 1100 hPa (9000 m a -500 m) con baixo ruido: 0,03 hPa (0,25 m).
 - Protocolo: I2C.
 - Tempo de resposta: 7,5 ms (máx.).
-

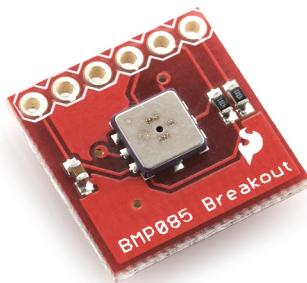


Figura 6.6: BMP085.

Polo que, cumprindo coas características solicitadas, deuse por bo e procedeu-se a incorporalo ó deseño.

Continuando coa segunda sección, precisábase dunha placa de sensores capacitivos que cumprise coas seguintes características:

- De dimensíons reducidas.
- De baixo consumo.
- Con moi boa sensibilidade.
- Cun tempo de resposta moi pequeno.
- Alomenos con sitio para 9 sensores (número de buratos onde situar os dedos).

Estando no mesmo caso ca no anterior, volveu tocar ir á súa procura en SparkFun. E de todos os disponibles (que en realidade son variantes do mesmo), a que mellor se axustaba era a *MPR121* [38] (figura 6.7). Fixándose na súa folla de especificacións, vese que conta coas seguintes características:

- Dimensíons: 20 mm x 30 mm.
- Consumo: 29 uA (máx.).
- Tempo de resposta: 1 ms a 128 ms (configurable en potencias de 2).

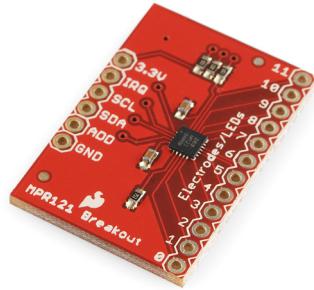


Figura 6.7: MPR121.

- Sensores: 12.

Polo que, cumprindo coas características solicitadas, deuse por boa e procedese a incorporala ó deseño.

E rematando pola terceira sección, precisábase dun sistema que puidera:

- Recibir, procesar e transmitir (sen fíos) os sinais dos periféricos.
- Almacenar as configuracións.
- Que fose autónomo.
- De dimensíóns reducidas.
- De baixo consumo.

Dadas estas características, o que procedía era buscar unha placa base Arduino que cumprise coa maioría delas. Mirando, esta vez si, na páxina oficial, a que máis se axustaba era a Arduino Fio [39] (figura 6.8). Entre as súas características:

- Microcontrolador ATmega328P (8 Mhz, 3,3V).
- Comunicación mediante módulos Xbee.
- Soporta varios protocolos: Serie, I2C e SPI.
- Conexión para unha batería de Polímero Litio (recargable a través do mi- niUSB).

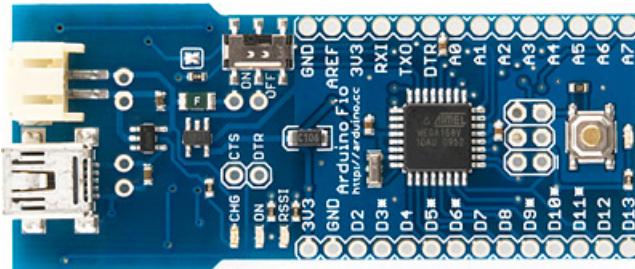


Figura 6.8: Arduino Fio.

- Dimensións: 28 mm x 66 mm.

Con respecto ós compoñentes que xa tiñamos, conta con soporte para I2C, polo que son perfectamente compatibles.

O máis salientable quizáis sexa a comunicación sen fíos mediante *XBee*². Para o caso que nos compete, precisábase un módulo *XBee* (non incluido coa placa *Arduino Fio*) de pouca potencia (polo tanto menor consumo) e que permitise topoloxía de rede en estrela (pois o receptor tiña que permitir a conexión de varios punteiros simultaneamente). Entre os módulos dispoñibles na páxina do fabricante, o máis axeitado é o *XBee ZB* [40] (figura 6.9):

- Protocolo: ZigBee (baseado en 802.15.4).
- Banda de transmisión: 2,4 GHz.
- Potencia de saída: 1,25/2 mW.
- Alcance: 120 m.
- Velocidade de transmisión: 250 Kbps.
- Consumo: 38 mA / < 1 uA.

Pero, ¿por que empregar *XBee* e non outras tecnoloxías coma *Bluetooth*³ ou *WiFi*⁴? Pois porque:

²Para máis información sobre dita tecnoloxía consúltese o apéndice D.

³Versión inferior á 4.0. No momento de atacar esta parte non existían shields para Arduino con soporte para *Bluetooth Low Energy* (BLE). As primeiras referencias datan de outubro de 2013.

⁴Estándares IEEE 802.11a/b/g/n.



Figura 6.9: XBee ZB.

- Consume moito menos ca calquera das outras dúas tecnoloxía. *XBee* ten un consumo de 30 mA transmitindo e 3 uA en repouso, fronte ós 40 mA transmitindo e 0,2 mA en repouso de *Bluetooth* [41]. Fronte á *WiFi* pasa exactamente o mesmo.
- A velocidade de transmisión é superior á de *Bluetooth* (115 Kbps, que é a velocidade á que poden chegar os módulos existentes para *Arduino* a través de conexión por porto serie [42]).
- Ante a mesma potencia de saída, ten moito máis alcance ca calquera das outras dúas: 1 m para *Bluetooth* [43] e 14 m para *WiFi* [44].
- A modo de resumo, aquí [45] hai unha boa comparativa.

Ben, a estas alturas estaba solucionado o tema do emisor, pero seguía faltando o receptor. Ningún equipo existente no mercado xeral conta a día de hoxe con conexión *XBee* integrada de serie, polo que foi preciso construír tamén un receptor.

O primeiro que se precisaba era outro módulo *XBee* exactamente igual para o receptor. Se non son exactamente iguais, a comunicación punto a punto non é posible [46]. A priori, a conexión punto a punto non era unha necesidade, pero mellor ter a opción por se era necesaria a posteriori.

E dito receptor había que conectalo a unha placa que nos permitise á súa vez conectalo a un equipo a través dalgún porto.

O normal para un dispositivo MIDI sería empregar un porto MIDI (figura 2.14). Sen embargo, dito porto está actualmente en desuso, dada a súa aparatosidade e polo tanto a súa difícil integración en dispositivos de dimensións reducidas. Aínda que sigue sendo unha opción minoritaria en interfaces de audio de alta gama para ordenadores de sobremesa. Actualmente, as casas máis importantes de controladores hardware MIDI (*Yamaha* e *Roland*, principalmente) están a substituír (ou complementar) dita conexión por un porto USB (facilmente contrastable na folla de especificacións de calquera dos seus teclados actuais, por exemplo).

É certo que existen placas complementarias para *Arduino* con porto MIDI, coma esta [47] (figura 6.10), por exemplo, pero desbotouse esta opción en favor do novo estándar de facto do mercado, o USB.

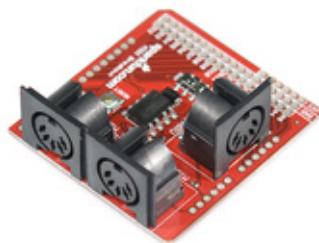


Figura 6.10: MIDI Breakout.

Novamente, voltando á páxina de *SparkFun*, atopamos o *XBee Explorer USB* [48] (figura 6.11) como a opción más sinxela. Simplemente montar e enchufar.

Pero esta opción plantexa un inconveniente con respecto a como o sería recoñecido o dispositivo por parte do equipo, polo que finalmente⁵ se optou por un empregar como base do receptor un *Arduino Uno* (figura 6.12) con firmware *Moco* (figura 6.13).

Unha vez solucionado o problema de recoñecer o receptor coma dispositivo MIDI, quedaba atopar a maneira de conectar o módulo *XBee* á placa *Arduino*

⁵Para máis información sobre cómo se solventou dito inconveniente, consúltese o capítulo 8.



Figura 6.11: XBee Explorer USB.

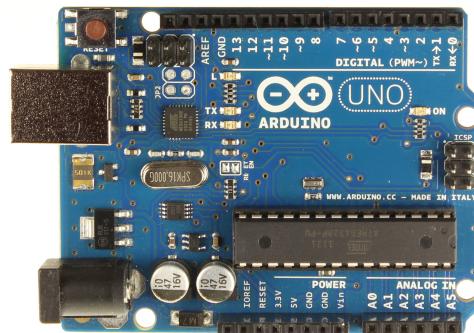


Figura 6.12: Arduino.

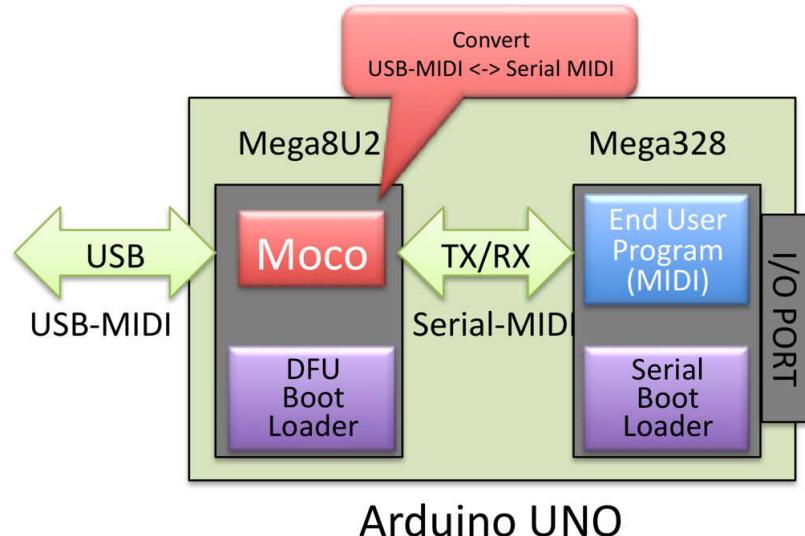


Figura 6.13: Moco.

Uno. Dita placa non ten zócalo para módulos *XBees* de serie, pero existen diversas soluciós en forma de placas accesorias (ou *shields*) que permiten conectar un módulo deste tipo. Entre as oficiais, contamos coa *Arduino Wireless SD Shield* [49] (figura 6.14) e a *Arduino Wireless Proto Shield* [50] (figura 6.15). As dúas son exactamente iguais coa excepción de que a primeira leva un lector de tarxetas de memoria de tipo SD e a segunda leva unha pequena zona de prototipado. Como resultaba máis interesante o lector de tarxetas para futuros usos, escolleuse a primeira.

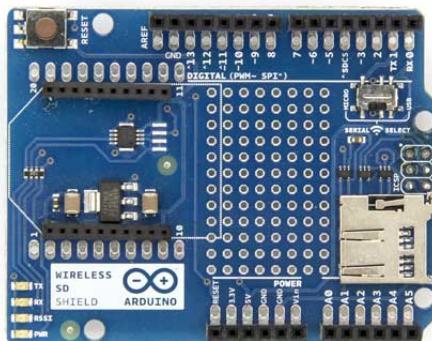


Figura 6.14: Arduino Wireless SD Shield.

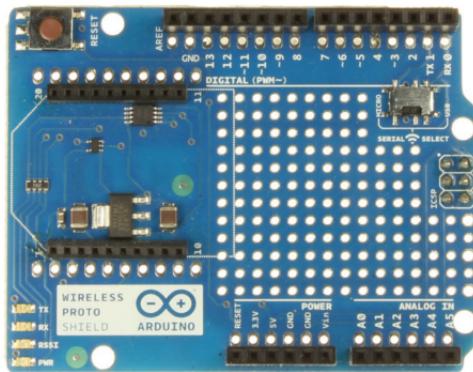


Figura 6.15: Arduino Wireless Proto Shield.

E con isto quedaba listo o receptor, cuxas pezas se incorporaron ó deseño.

Pero seguían quedando pendentes certas características do sistema embebido no punteiro: o almacenamento da configuración e a alimentación.

Para o almacenamento da configuración, nun primeiro momento podíase chegar a pensar no lector de tarxetas do receptor, pero tendo en conta que cada punteiro ten a súa configuración independente e que un mesmo receptor pode ser usado por varios punteiros aleatorios simultáneamente, esta opción quedaba invalidada.

Había que almacenar a información dentro do propio punteiro, pero ningún dos elementos incorporados previamente ó deseño (sensor de presión, placa de sensores capacitivos, Arduino Fio) contaba con zócalo para unha tarxeta de memoria, nin con ningunha placa accesoria que se puidese superpoñer á placa base do sistema.

Tocou buscar nas páxinas de referencia se existía algunha placa de pequenas dimensíóns e baixo consumo que tivese coma función exclusiva a de lector de tarxetas de memoria. De entre todas as opcións existentes nas distintas páxinas de referencia, a mellor opción con diferencia era a *MicroDrive G1* [51] (figura 6.16). Entre as súas características:

- Dimensíóns: 14,9 x 18,9 mm.
- Consumo: 25 mA.
- Almacenamento: tarxetas MicroSD de ata 4 GB ou MicroSD HC de tamaño superior.
- Formato: RAW ou FAT16.
- Protocolo: UART (serie).

Ten un consumo un pouco elevado, pero inferior ó do módulo *XBee* e a calquera das outras alternativas no mercado, polo que é asumible. Ademais, emprega o protocolo Serie, polo que nos plantexaba un problema á hora de conectarlo á placa *Arduino Fio*, que só conta cun porto serie (RX/TX) que xa estaba a ser empregado polo módulo *XBee*. Pero isto é facilmente solucionable mediante a biblioteca *SoftwareSerial* de *Arduino*, que nos permite xestionar múltiples periféricos con protocolo serie conectados a diferentes portos da mesma placa sen incrementar

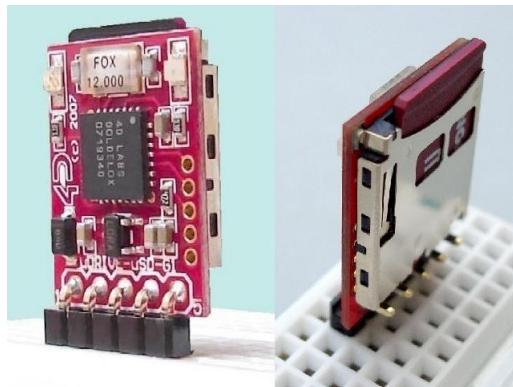


Figura 6.16: MicroDrive G1.

practicamente en nada a complexidade do código.

Solventados os problemas, procedeuse a incorporar esta última placa ó deseño. Agora só quedaba buscar unha batería axeitada para a placa base, a *Arduino Fio*. Debía de cumplir unha serie de requisitos:

- Ser de pequenas dimensións.
- Ser de Polímero Litio (LiPo).
- Ter bastante capacidade.
- Ter conector JST.

E entre todas as dispoñibles, a máis indicada era a *LiPo 850 mAh* (figura 6.17) disponible en *BricoGeek* [52]:

- Dimensións: 5,84 x 29,5 x 51 mm.
- Peso: 18,5 g.
- Tipo: 2C (3,7 V).
- Conector: JST.
- Material: LiPo.



Figura 6.17: Batería LiPo 850 mAh con conector JST.

Cumprindo coas características, incorporouse ó deseño. E con isto xa estaba case todo listo para poder pasar a realizar o **Prototipo 3**.

Faltaba por escoller a ferramenta de deseño software apropiada para dita tarefa. Facendo unha busca rápida na rede sobre *software de deseño hardware para Arduino*, damos con dúas solucións semellantes, pero ó mesmo tempo bastante diferentes: *Eagle* [53] e *Fritzing*⁶ [54].

- *Eagle* é o software CAD propietario baixo cuxo formato se poñen dispoñibles os deseños oficiais de *Arduino*. Aínda que na documentación diga que é sinxelo de usar, non o é tanto. Ten unha interface moi completa e bastante complexa que implica unha curva de aprendizaxe bastante lenta.
- *Fritzing* é a contrapartida libre e sinxela a *Eagle*. A interface é moito más intuitiva, ademais de contar cunha base de datos de pezas de *Arduino* (e periféricos das principais casas) bastante grande (o que aforra un traballo considerable á hora de facer os deseños) e cunha vista do resultado “real” coa que non conta *Eagle* (moi interesante á hora de incluír os deseños na documentación).

Evidentemente, por cuestión de requisitos (e de principios) escolleremos sempre a versión libre (e más sinxela) sobre a propietaria (sempre que a primeira

⁶Para máis información sobre esta ferramenta, consúltese o apéndice E.

funcione correctamente).

A instalación da aplicación deu lugar a unha serie de incidencias, reflexadas no capítulo 8.

Solventadas ditas incidencias, puido comezarse co deseño. O primeiro foi buscar tódalas pezas na base de datos e incorporalas ó proxecto:

- A placa *Arduino Fio*, non contaba expresamente co zócalo para o módulo *XBee*. Isto é debido a que só se pode reflexar unha cara das pezas no software, polo que o módulo, que quedaba na outra cara, quedou sen reflexar. O autor da mesma podía ter optado por desdobrar a peza e reflexala, pero non o fixo. Por este motivo, a conexión do módulo XBee tivo que ser directa ós pinos xenéricos da placa.
- A placa de sensores capacitivos *MPR121* non estaba disponible na súa versión *Breakout* pero si na versión *Touch*. A única diferencia é que esta última incorpora varios electrodos xa na propia placa, polo que o seu tamaño é un pouco maior, pero non inflúe no deseño, pois conta coas mesmas conexións, polo que para un deseño “non físico” é equivalente.
- A placa de almacenamento *MicroDrive G1* non estaba disponible na base de datos do software por ser dun fabricante menos coñecido. Afortunadamente, estaba disponible no apartado de contribucións do repositorio da aplicación [55]. Unha vez descargada, intentouse incorporala ó proxecto, pero non foi posible, pois tiña algún tipo de erro (e por iso non estaba incluida na base de datos). O como se solucionou esta incidencia coméntase tamén no capítulo 8.

Unha vez incorporadas todas as pezas ó proxecto, procedeuse a facer as conexións pertinentes mediante cable, previo estudo das follas de especificacións de cada peza (conexións, protocolos, etc.). Compre aclarar que moitos deses cables non aparecerán como tal no producto hardware final, dado que a maioría das pezas son ensamblables, pero esta característica non se pode reflexar na aplicación. A continuación expónense as distintas vistas do deseño (figuras 6.18, 6.18 e 6.20).

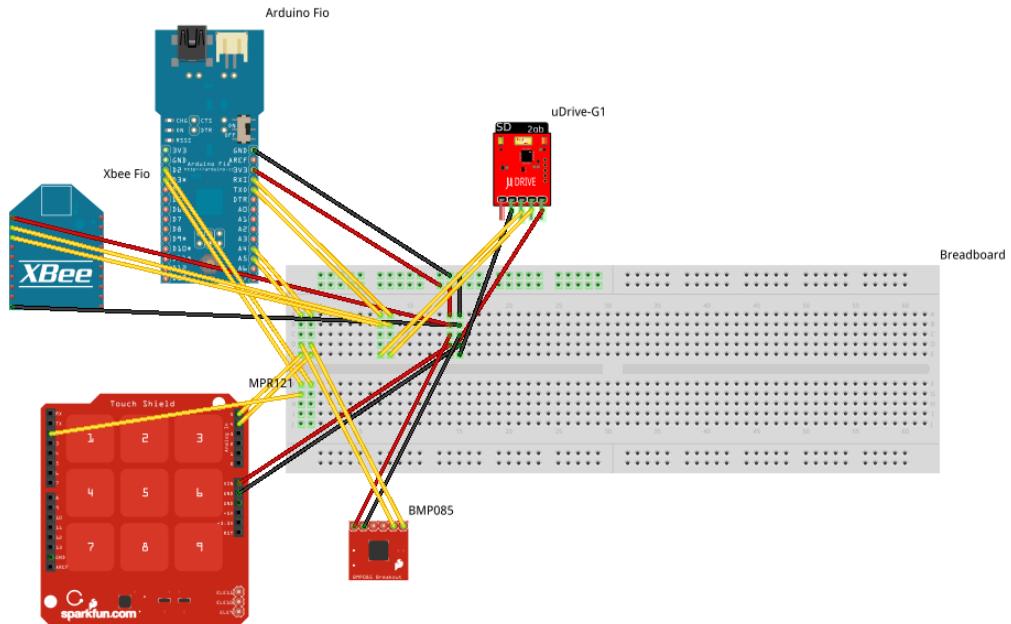


Figura 6.18: Prototipo 3 hardware, emisor: vista real.

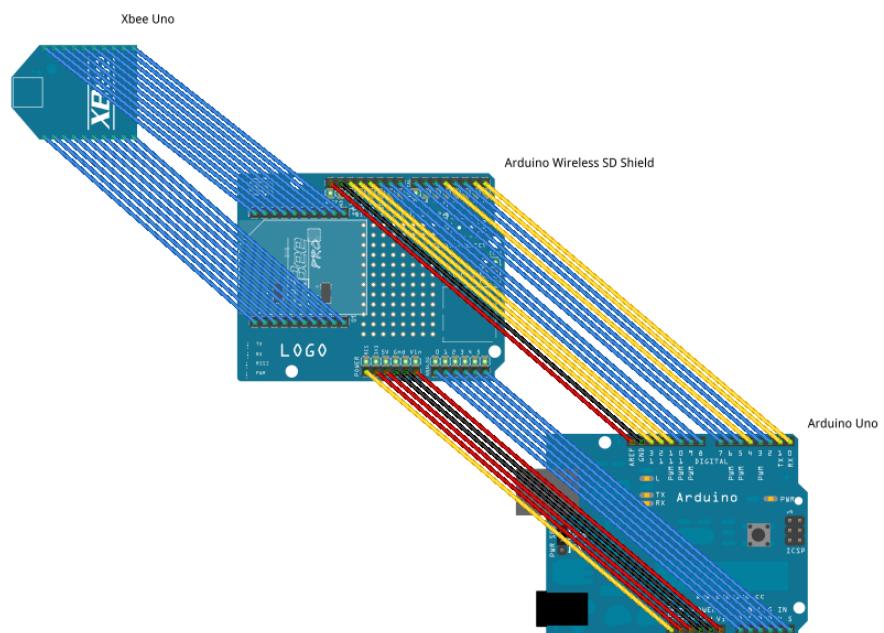


Figura 6.19: Prototipo 3 hardware, receptor: vista real.

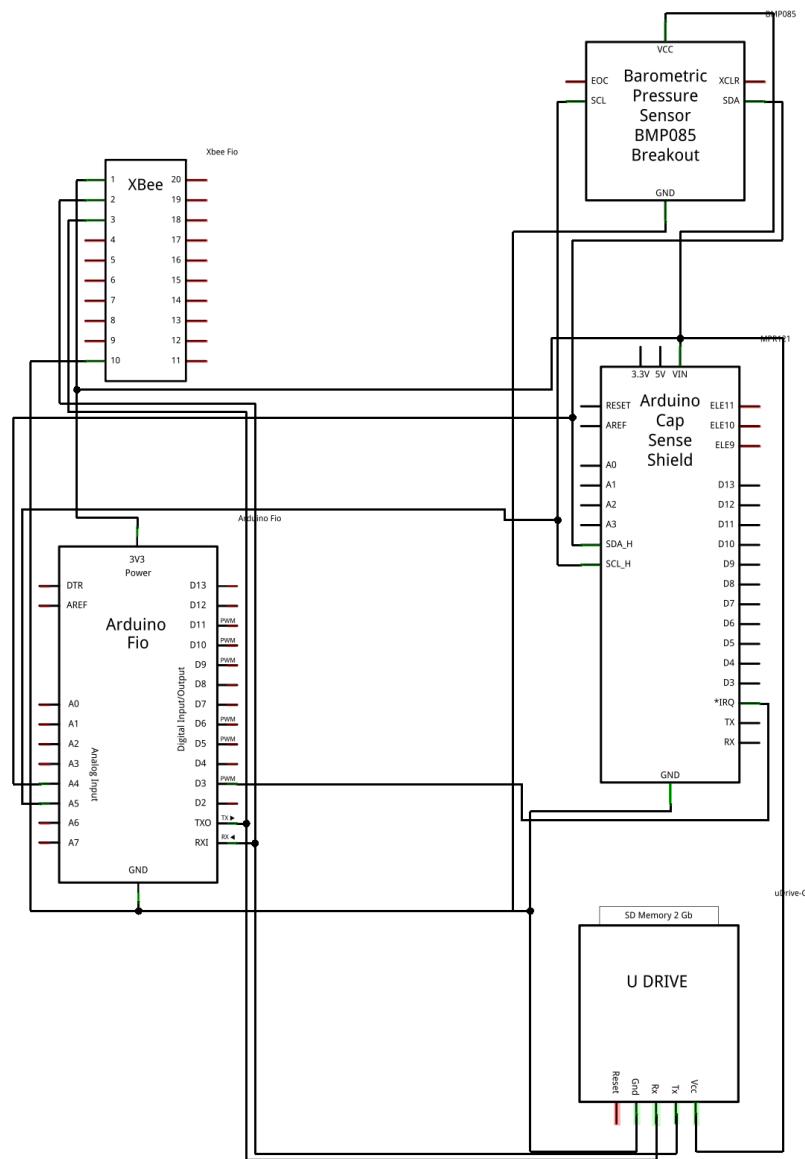


Figura 6.20: Prototipo 3 hardware: vista esquemática.

6.2.2.2. Prototipo software

O *Prototipo 3* na súa vertente software, é basicamente o *Prototipo 2* con moi lixeiras modificóns:

- Axustáronse as referencias ás oitavas ós valores dados polo estándar americano de notación musical, que é o máis empregado.
- Incluíuse a opción de poder modificar a sensibilidade de tódolos sensores á vez, que aínda que estaba contemplada, pasárase por alto metela no despregable.
- Despois de rematar o *Prototipo 3* na súa versión hardware, púidose comprobar que a placa de sensores capacitivos *MPR121* conta con dous valores axustables, o de *press* e o de *release*, polo que se incluíu outro *slider* na interface para poder controlar tamén este último a vontade.

Feitos todos estes pequenos cambios, o **Prototipo 3** software quedou como segue (figuras 6.21 a 6.25).

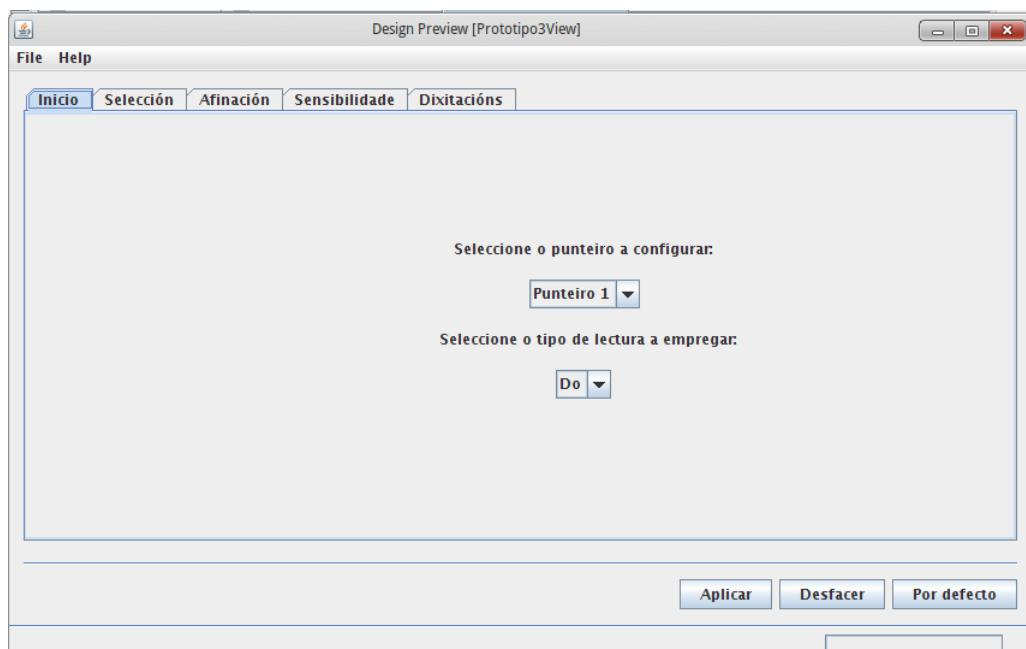


Figura 6.21: Prototipo 3: pantalla de inicio.

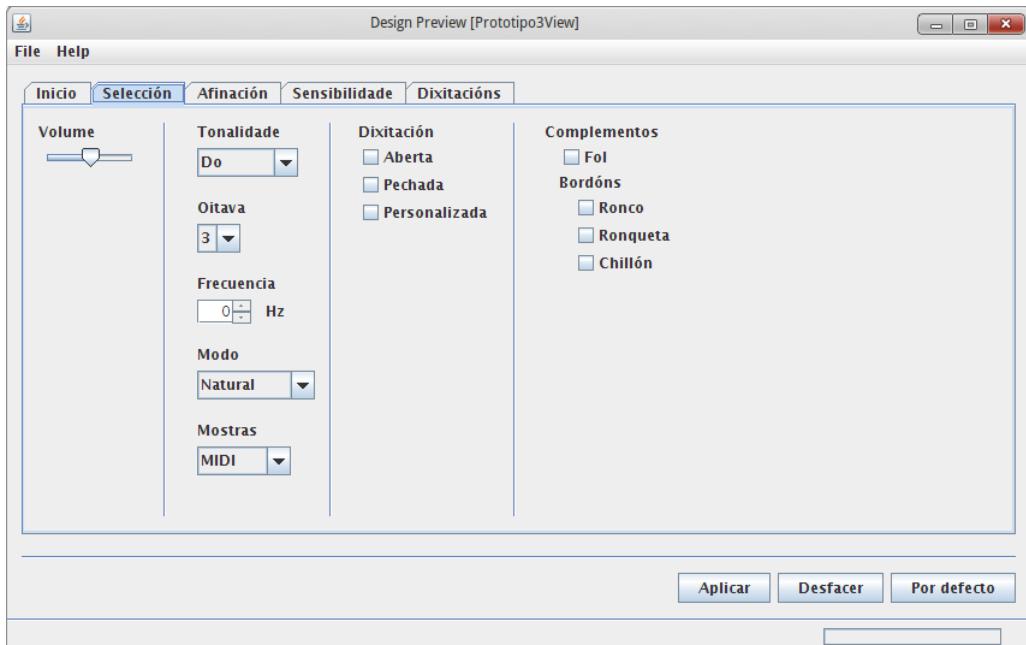


Figura 6.22: Prototipo 3: pantalla de selección.

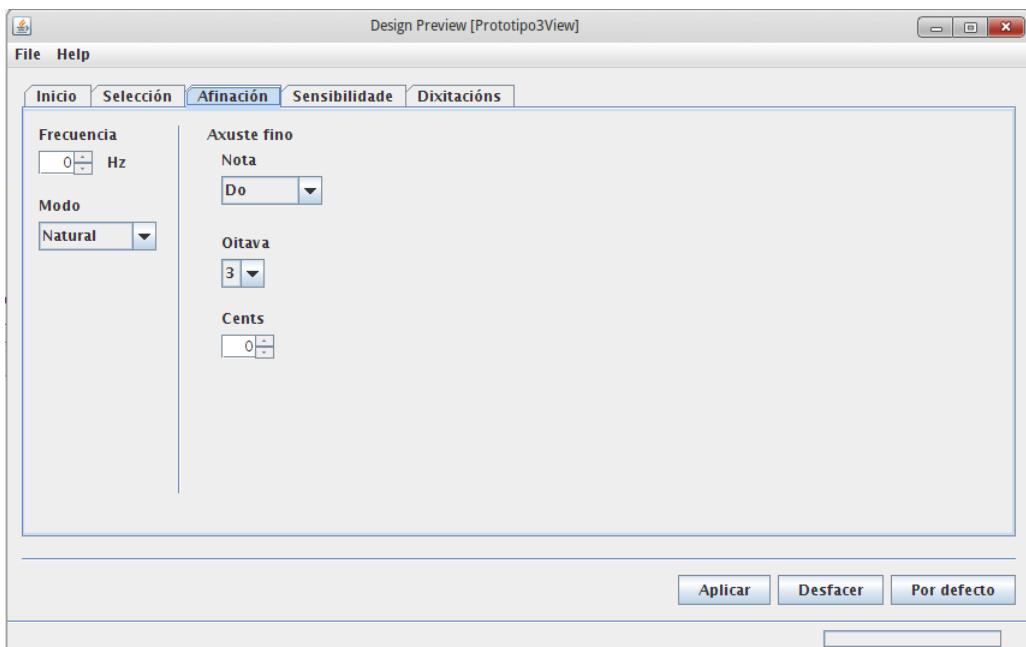


Figura 6.23: Prototipo 3: pantalla de afinación.

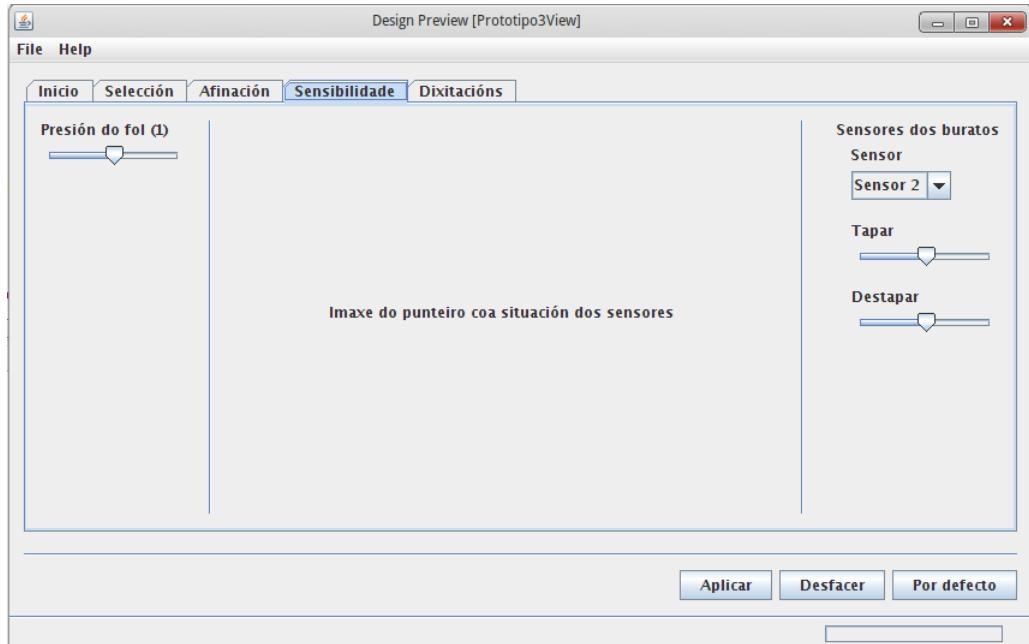


Figura 6.24: Prototipo 3: pantalla de sensibilidad.

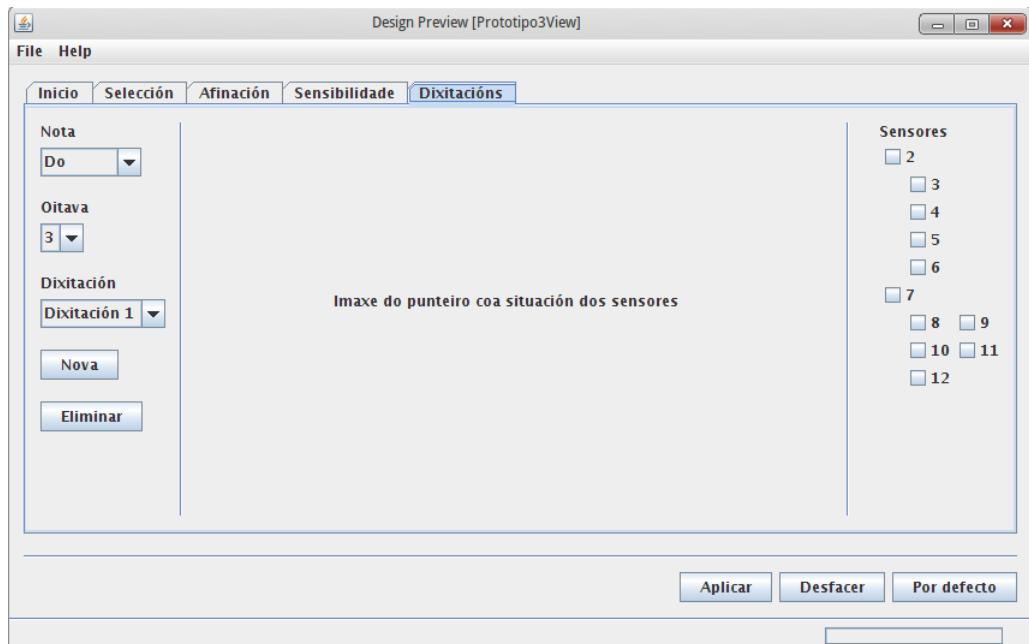


Figura 6.25: Prototipo 3: pantalla de dixitación.

6.3. Desenvolvemento e validación do seguinte nivel do producto

6.3.1. Simulacións, modelos e programas de proba

As probas a realizar neste nivel do producto dividíronse en dous grupos:

- Por unha banda, consistiron en verificar sobre o *Prototipo 3 hardware* todos aqueles requisitos hardware aplicables ó mesmo recollidos na especificación de requisitos.
- Por outra, verificouse novamente o *Prototipo 3 software* para que comprobar que os pequenos cambios introducidos non inducisen a unha inconformidade coa especificación de requisitos.
- E finalmente, verificouse contra a especificación de requisitos o deseño obtido a partir de ambos prototipos e validous o seu funcionamento atendendo ás funcionalidades solicitadas.

6.3.2. Deseño do producto hardware e software

6.3.2.1. Deseño do producto hardware

Contando xa cun deseño formal dende a fase de prototipado e solventados todos os pequenos problemas que se atoparon durante o desenvolvemento do mesmo, decidiuse empregar o propio *Prototipo 3 hardware* coma deseño do producto hardware.

6.3.2.2. Deseño do producto software

Atendendo ó deseño dos prototipos anteriores, téñense dúas aplicacións ben diferenciadas:

- A aplicación embebida dentro do punteiro (*Prototipo 3 hardware*), que se encarga de do control e configuración do hardware e de proporcionar as funcionalidades do mesmo.

- É a aplicación de escritorio, encargada da configuración tanto do punteiro coma do sintetizador.

Polo tanto, houbo que tratalas coma tal e realizar dous deseños independentes, pero ó mesmo tempo relacionados. O resultado ó que se chegou foi o da figura 6.26.

Como se pode apreciar, existe un patrón Modelo-Vista-Controlador, onde:

- A Vista correspóndese coa interface gráfica ou GUI.
- O Controlador é un conxunto de clases, agrupadas segundo a división en pestanas da interface gráfica e que inclúen tódalas funcións necesarias para o correcto funcionamento da Vista.
- O Modelo correspóndese coa aplicación embebida no punteiro. O deseño da mesma é un pouco peculiar, xa que nun principio non vai ser orientada a obxectos (aínda que podería, dada a versatilidade de Arduino), pois non aporta demasiado, como se pode comprobar no propio deseño; pero non deixa de ser unha maneira equivalente de representalo á hora de plasmalo en UML.

Así mesmo, cómpre lembrar que se trata dun deseño de alto nivel e, polo tanto, só se representaron as clases máis significativas. O que se pretende é dar unha noción básica do deseño da aplicación, facilitando a súa comprensión, evitando a complexidade dun deseño recargado de clases secundarias que pouco ou nada aportarían.

6.3.3. Verificación e validación do deseño

Realizadas as probas anteriormente definidas e verificados e validados os dous deseños previos, non se atoparon inconformidades, polo que se deron por bos e se continuou coa seguinte fase.

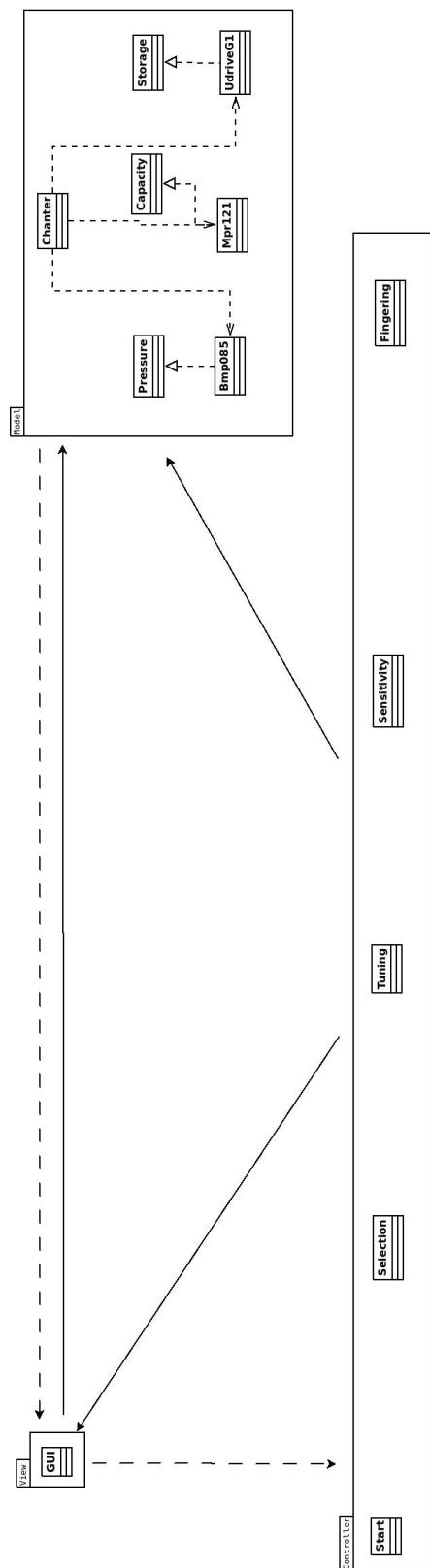


Figura 6.26: Deseño de alto nivel.

6.4. Planificación da próxima fase ou ciclo

6.4.1. Planificación do desenvolvemento

A continuación (figura 6.27) exponse a planificación inicial do ciclo de desenvolvemento.

The screenshot shows a hierarchical tree view of a project plan. The root node is 'Desenvolvemento'. Below it are several sub-tasks: 'Determinación 4', 'Obxectivos 4', 'Alternativas 4', 'Restriccóns 4', 'Avaliación de alternativas e resolución de riscos 4' (which further branches into 'Análise de riscos 4', 'Prototipado operacional', 'Prototipo hardware 4' which has sub-tasks 'Integración do hardware' and 'Encapsulamento do hardware', 'Prototipo software 4' which has sub-tasks 'Desenvolvemento do prototipo' and 'Gravación dos samples', and 'Desenvolvemento e validación do seguinte nivel do producto 4' which has sub-tasks 'Simulacións, modelos e programas de proba 4', 'Deseño detallado' which has sub-tasks 'Deseño hardware' and 'Deseño software', and 'Ensamblado e codificación' which has sub-tasks 'Ensamblado' and 'Codificación', followed by 'Probas de unidade', 'Integración e probas', 'Probas de aceptación', and 'Implantación'. Each task has a checkbox next to its name. To the right of the tree view is a table with three columns: 'Nome' (Name), 'Inicio' (Start Date), and 'Fin' (End Date). The start date for 'Desenvolvemento' is 1/21/12 and the end date is 5/26/12. The start date for 'Determinación 4' is 1/21/12 and the end date is 1/24/12. The start date for 'Obxectivos 4' is 1/21/12 and the end date is 1/23/12. The start date for 'Alternativas 4' is 1/23/12 and the end date is 1/23/12. The start date for 'Restriccóns 4' is 1/23/12 and the end date is 1/24/12. The start date for 'Avaliación de alternativas e resolución de riscos 4' is 1/24/12 and the end date is 3/3/12. The start date for 'Análise de riscos 4' is 1/24/12 and the end date is 1/25/12. The start date for 'Prototipado operacional' is 1/25/12 and the end date is 3/3/12. The start date for 'Prototipo hardware 4' is 1/25/12 and the end date is 2/10/12. The start date for 'Integración do hardware' is 1/25/12 and the end date is 2/4/12. The start date for 'Encapsulamento do hardware' is 2/4/12 and the end date is 2/10/12. The start date for 'Prototipo software 4' is 2/10/12 and the end date is 3/3/12. The start date for 'Desenvolvemento do prototipo' is 2/10/12 and the end date is 3/3/12. The start date for 'Gravación dos samples' is 2/10/12 and the end date is 2/16/12. The start date for 'Desenvolvemento e validación do seguinte nivel do producto 4' is 3/3/12 and the end date is 5/26/12. The start date for 'Simulacións, modelos e programas de proba 4' is 3/3/12 and the end date is 3/10/12. The start date for 'Deseño detallado' is 3/10/12 and the end date is 3/24/12. The start date for 'Deseño hardware' is 3/10/12 and the end date is 3/17/12. The start date for 'Deseño software' is 3/17/12 and the end date is 3/24/12. The start date for 'Ensamblado e codificación' is 3/24/12 and the end date is 5/5/12. The start date for 'Ensamblado' is 3/24/12 and the end date is 4/7/12. The start date for 'Codificación' is 4/7/12 and the end date is 5/5/12. The start date for 'Probas de unidade' is 5/5/12 and the end date is 5/12/12. The start date for 'Integración e probas' is 5/12/12 and the end date is 5/19/12. The start date for 'Probas de aceptación' is 5/19/12 and the end date is 5/24/12. The start date for 'Implantación' is 5/24/12 and the end date is 5/26/12.

Nome	Inicio	Fin
Desenvolvemento	1/21/12	5/26/12
Determinación 4	1/21/12	1/24/12
Obxectivos 4	1/21/12	1/23/12
Alternativas 4	1/23/12	1/23/12
Restriccóns 4	1/23/12	1/24/12
Avaliación de alternativas e resolución de riscos 4	1/24/12	3/3/12
Análise de riscos 4	1/24/12	1/25/12
Prototipado operacional	1/25/12	3/3/12
Prototipo hardware 4	1/25/12	2/10/12
Integración do hardware	1/25/12	2/4/12
Encapsulamento do hardware	2/4/12	2/10/12
Prototipo software 4	2/10/12	3/3/12
Desenvolvemento do prototipo	2/10/12	3/3/12
Gravación dos samples	2/10/12	2/16/12
Desenvolvemento e validación do seguinte nivel do producto 4	3/3/12	5/26/12
Simulacións, modelos e programas de proba 4	3/3/12	3/10/12
Deseño detallado	3/10/12	3/24/12
Deseño hardware	3/10/12	3/17/12
Deseño software	3/17/12	3/24/12
Ensamblado e codificación	3/24/12	5/5/12
Ensamblado	3/24/12	4/7/12
Codificación	4/7/12	5/5/12
Probas de unidade	5/5/12	5/12/12
Integración e probas	5/12/12	5/19/12
Probas de aceptación	5/19/12	5/24/12
Implantación	5/24/12	5/26/12

Figura 6.27: Planificación inicial do ciclo de desenvolvemento.

Total: 344 horas.

Capítulo 7

Desenvolvemento do sistema

Índice xeral

7.1. Determinación	136
7.1.1. Obxectivos	136
7.1.2. Alternativas	136
7.1.3. Restriccóns	136
7.2. Avaliación de alternativas e resolución de riscos	137
7.2.1. Análise de riscos	137
7.2.2. Prototipo operacional	137
7.3. Desenvolvemento e validación do seguinte nivel do producto	168
7.3.1. Simulacións, modelos e programas de proba	168
7.3.2. Deseño detallado	169
7.3.3. Ensamblado e codificación	174
7.3.4. Probas de unidade	193
7.3.5. Integración e probas	202
7.3.6. Probas de aceptación	205
7.3.7. Documentación	210

NESTE capítulo exporase o desenvolvemento do proxecto baseándose no esquema proporcionado pola planificación inicial: desde o deseño software e hardware de baixo nível do sistema ata a implementación e o ensamblado do producto, pasando por un prototipo operacional.

7.1. Determinación

7.1.1. Obxectivos

Establecerónse os obxectivos da fase de desenvolvemento do proxecto.

Obxectivos:

- Implementar unha gaita MIDI sen fíos en tempo real empregando software/hardware libre.

7.1.2. Alternativas

Establecerónse posibles alternativas a eses obxectivos, aplicables no caso de que estes non se puidesen cumplir.

Alternativas:

- Se non é posible implementar completamente o proxecto, pode optarse por:
 1. Se é por causa de que o hardware non o permite, cambiar as pezas correspondentes por outras que si o permitan.
 2. Se as partes que non é posible implementar son opcionais ou de pouco peso, desbotalas e implementar o resto.
 3. Cancelar e mudar de proxecto.

7.1.3. Restriccóns

Establecerónse restriccóns aplicables a ditos obxectivos.

1. As propias restriccóns veñen dadas polo propio título do proxecto. A saber:
 - a) Empregar o protocolo MIDI.
 - b) Empregar tecnoloxía sen fíos.
 - c) Empregar tempo real.
 - d) Empregar software libre.

- e) Empregar hardware libre.
- f) E/ou as derivadas de calquera das súas alternativas.

7.2. Avaliación de alternativas e resolución de riscos

7.2.1. Análise de riscos

Determináronse os riscos que comportaban as distintas alternativas e as súas posibles solucións.

1. Alternativas 1.

- a) Riscos:
 - 1) Que non exista hardware alternativo que soporte a implementación das características restantes.
 - 2) Que as partes a desbotar sexan partes importantes ou incluso críticas.
 - 3) Que o tempo restante para a execución do proxecto non sexa suficiente.
- b) Solucións:
 - 1) Recortar características ou cancelar e mudar de proxecto.
 - 2) Aplicar medidas de mitigación para que a planificación non se vexa afectada en extremo, ou cancelar e mudar de proxecto se fan inviable o mesmo.
 - 3) Agardar a presentalo na seguinte convocatoria.

7.2.2. Prototipo operacional

Nesta fase desenvolveuse o seguinte nivel de prototipado, tanto hardware como software, sendo xa ambos operacionais.

A aproximación seguida foi aplicar *BDD* [56] ou *TDD* [57] segundo o caso, en dirección *top-down* obtendo así as sinaturas dos servizos e o comportamento

desexado dos mesmos primeiro, para logo facer unha implementación *bottom-up* respectando o deseño conseguido previamente.

7.2.2.1. Prototipo hardware

Para a o prototipo hardware operacional tiramos do prototipo deseñado na fase anterior, para o cal empregamos ferramentas CAD co fin de poder facer probas en formato dixital antes de pasar a formato físico, evitando así os múltiples problemas relacionados.

7.2.2.1.1. Integración do hardware

Unha vez claro o deseño sobre o papel, montouse un prototipo físico con cada compoñente por separado, de maneira que inicialmente se puidera probar cada un de maneira unitaria, antes de pasar á integración final.

Desta maneira, dividiuse a montaxe completa en varias submontaxes:

- Router (figura 7.1).
- Receptor (figura 7.2).
- Sensor de presión (figura 7.3).
- Sensores capacitivos (figura 7.6).
- Lector de tarxetas (figura 7.9).

A primeira montaxe foi a do router por resultar a máis sinxela, xa que consistiu nunha placa Arduino Uno sobre a que se montou unha placa auxiliar onde se montou o transmisor XBee, todo sen maior problema por ser altamente acoplables, sendo logo embebidos nunha caixa a medida e conectados por USB ó ordenador onde se executa a aplicación de configuración e o servidor MIDI.

A nivel de sinatura de servizos, ó funcionar coma un simple router automatizado, non tiña máis que probar a transmisión correcta de datos, que decidiu deixarse coma último punto da integración hardware por razóns de fiabilidade na



Figura 7.1: Router.

transmisión dos mesmos.

A seguinte montaxe foi a do receptor, moi similar á do router, pois non deixa de ser unha placa Arduino Fio, que xa dispón dun zócalo para o módulo XBee e polo que non precisa de placa auxiliar.

Polo mesmo motivo que o anterior, deixouse para más adiante coma o último punto ou nivel de integración, pois resulta máis robusto probar e medir primeiro de maneira cableada e logo pasar ó modo sen fíos e poder comparar medicións e resultados.

No seu lugar, para desenvolver a lóxica de negocio da gaita MIDI, empregouse unha placa Arduino Uno, que conta coas mesmas características, salvo pola conexión directa do módulo XBee e que conta coa vantaxe de que podemos descargar nela o firmware da nosa implementación directamente a través dun cable USB.

Neste caso a interface pública veu definida polo propio funcionamento de

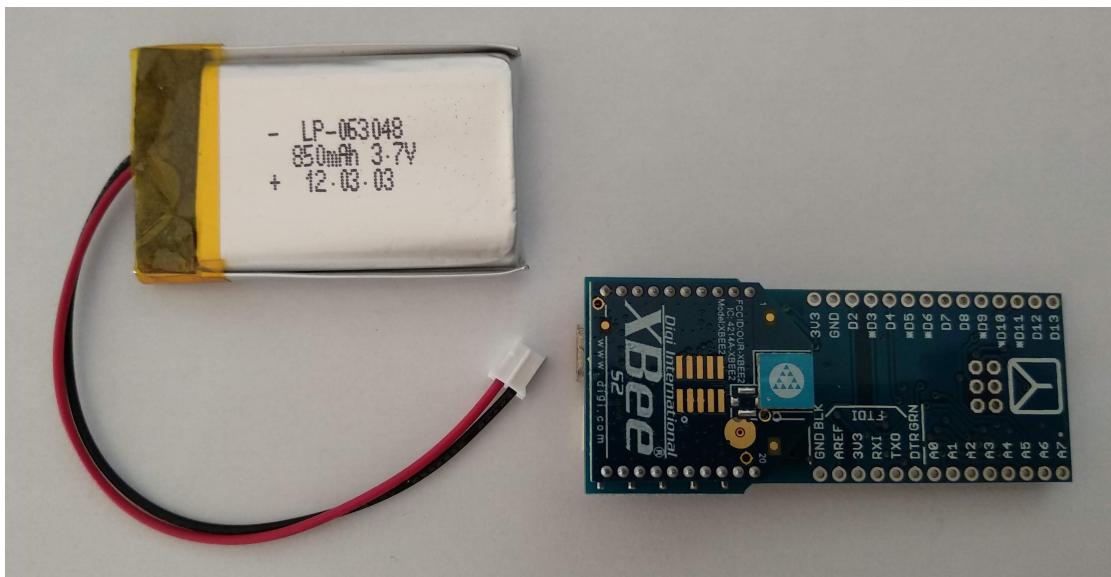


Figura 7.2: Receptor.

Arduino, contando cun método de configuración inicial e outro de execución continua.

O que fixemos neste caso foi facer unha primeira implementación en pseudo-código do comportamento xeral que tería o dispositivo, intentando ver a qué periférico tería que chamar en cada caso, cando e como, dando como resultado dúas partes claramente diferenciadas:

- Configuración das características propias da gaita.
- E reproducción ou envío de mensaxes MIDI.

Na parte de configuración, realizaríase tanto a carga dos parámetros configurables ó inicio coma o intercambio de configuracións entre o dispositivo e a aplicación de configuración, para o que sería preciso tirar do lector de tarxetas para persistir a información, así coma unha librería JSON para darlle formato un formato usable á mesma e da que falaremos máis adiante.

En canto á parte de reproducción, sería preciso avaliar a presión do fol para decidir se o dispositivo debe soar ou non e a dixitación actual a través dos sensores capacitivos para ver se corresponde con algúna válida e por tanto producir

o son relativo á mesma mediante o emprego dunha librería MIDI da que tamén falaremos máis adiante.

Ampliaremos esta lóxica polo miúdo cando chegemos á parte software, facendo uso de diagramas de fluxo.

A continuación procedeuse co sensor de presión, por ser o seguinte más sínxelo en orde de funcionalidade, pois o único que precisamos obter del é a presión dentro do fol, aínda que pode devolver máis parámetros relacionados coa mesma.

Como se pode ver na imaxe, conectouse a unha placa Arduino Uno empregando o bus I2C do mesmo.

Ademáis de definir a súa interface pública (figura 7.4) para a obtención da presión e un ficheiro de proba (figura 7.5) para validar a implementación completa posterior.

O turno seguinte foi para os sensores capacitivos. Neste caso, o que precisamos saber é cáles están acesos en cada momento, de maneira que poidamos facer *pattern matching* contra a configuración da gaita e saber se teríamos algúns son asociado a dita dixitación, para posteriormente reproducilo.

Como se pode ver na imaxe, conectouse a unha placa Arduino Uno empregando o bus I2C do mesmo.

Ademáis de definir a súa interface pública (figura 7.7) para a obtención da dixitación e un ficheiro de proba (figura 7.8) para validar a implementación completa posterior.

Para rematar cos periféricos, procedeuse co lector de tarxetas. Neste caso o que precisamos é poder ler e almacenar a configuración variable do dispositivo, de maneira que sexa independente e única para cada un dos mesmos.

Como se pode ver na imaxe, conectouse a unha placa Arduino Uno empregando o bus I2C do mesmo.

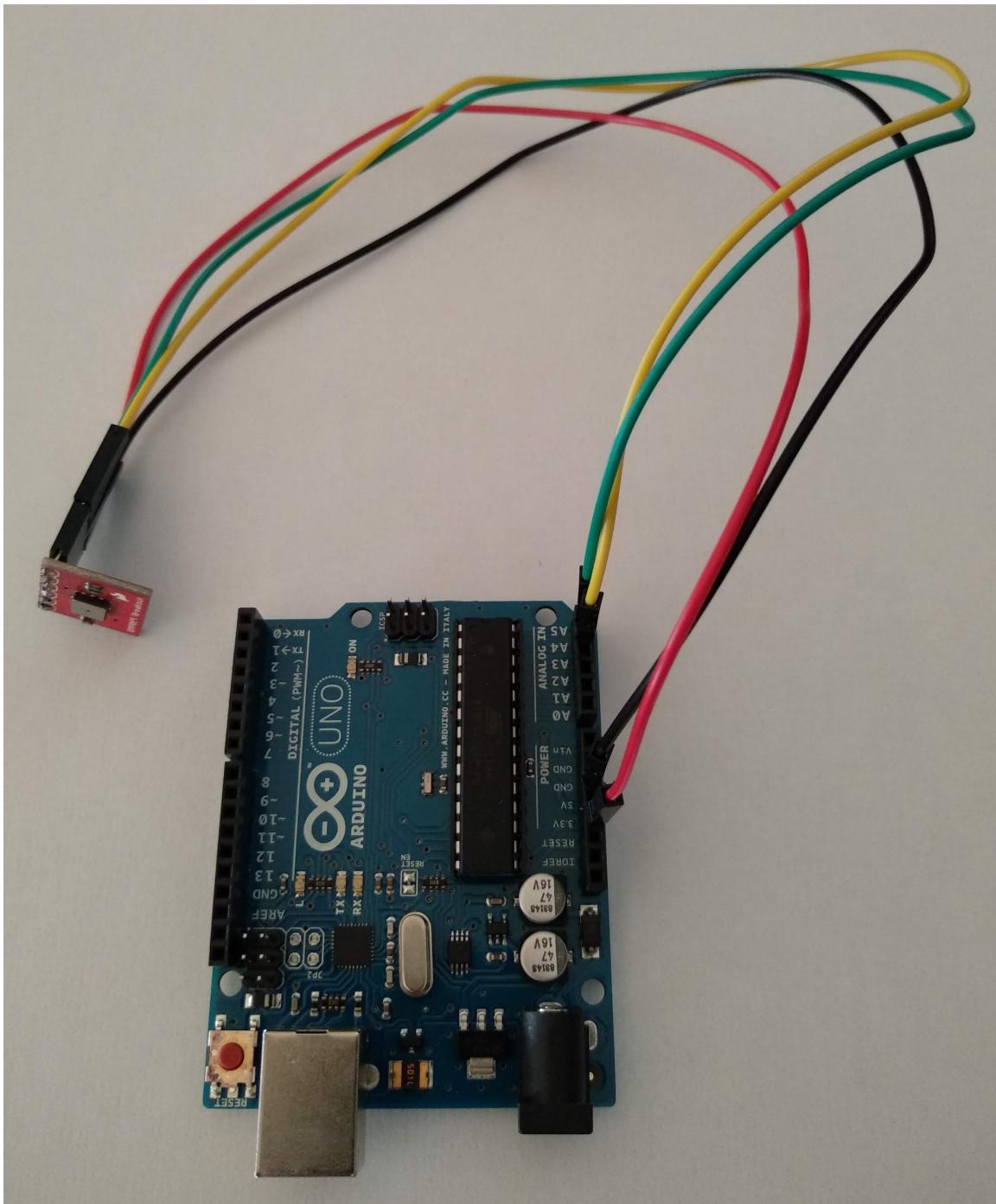


Figura 7.3: Sensor de presión.

```
Bmp085.h
/*
 * registers.
 */
void getRegisters(byte address, int quantity, byte *data);

/** @brief Read uncompensated pressure value.
 *
 * @return Uncompensated pressure value.
 */
long getUP();

/** @brief Read uncompensated temperature value.
 *
 * @return Uncompensated temperature value.
 */
long getUT();

/** @brief Setup the I2C communication.
 *
 * Initialize the I2C communication via Wire library.
 */
void setI2C();

/** @brief Write a 1 byte value into the BMP085 write address.
 *
 * @param value Content to write into the register.
 */
void setRegister(byte value);

/** @brief Setup the Arduino serial port.
 *
 * Initialize the Arduino serial port setting the communication velocity.
 */
public:

/** @brief Constructor.
 */
Bmp085();

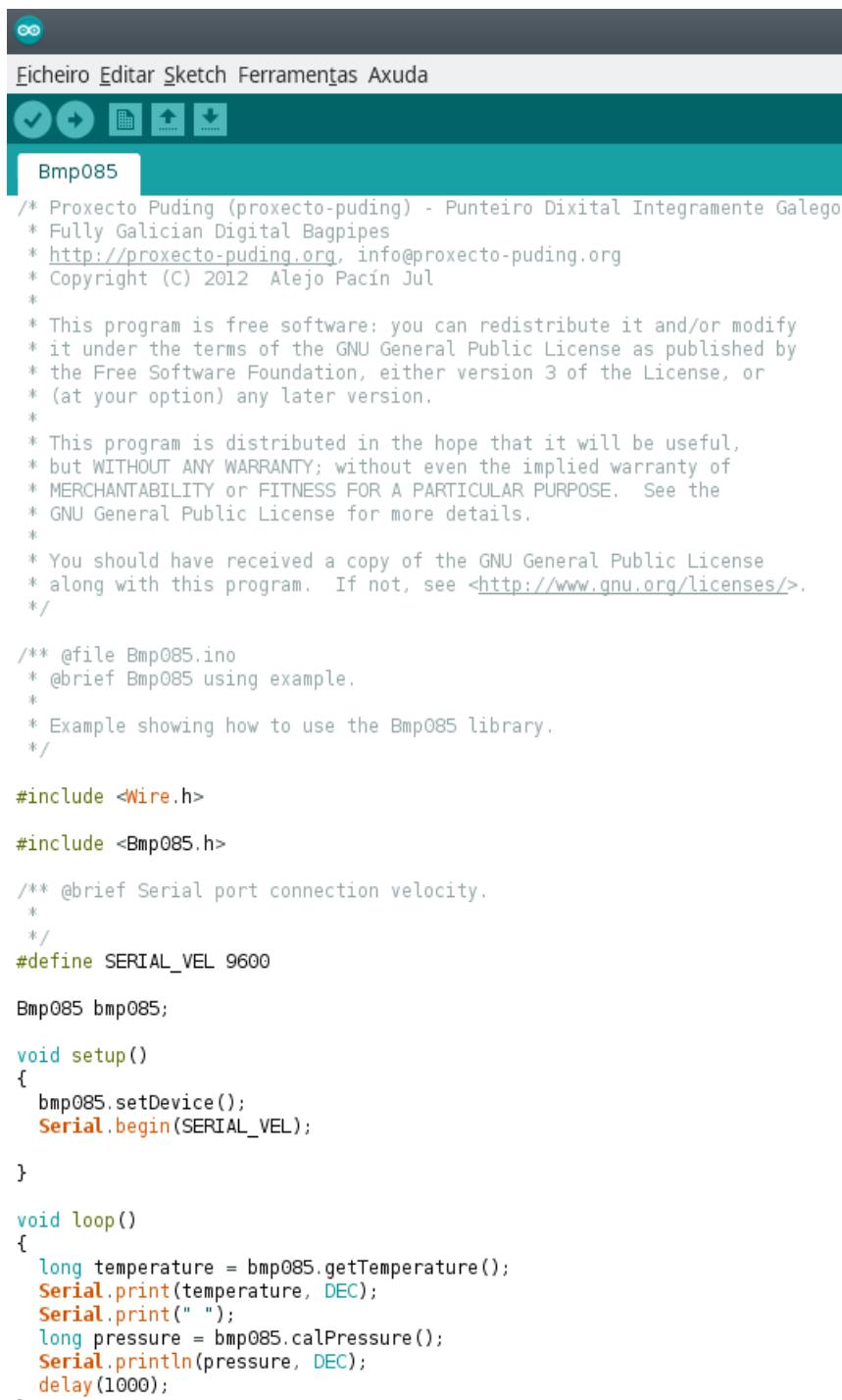
/** @brief Calculate true pressure.
 *
 * @return Pressure in Pa.
 */
long calPressure();

/** @brief Calculate true temperature.
 *
 * Function getUT() must be called first.
 *
 * @return Temperature in 0.1 °C.
 */
long getTemperature();

/** @brief Setup the BMP085 pressure sensor.
 *
 * Initialize the communication ports and the configuration parameters.
 */
void setDevice();
};

#endif
```

Figura 7.4: Interface do sensor de presión.



```

/*
 * Proxecto Puding (proxecto-puding) - Punteiro Dixital Integramente Galego
 * Fully Galician Digital Bagpipes
 * http://proxecto-puding.org, info@proxecto-puding.org
 * Copyright (C) 2012 Alejo Pacín Jul
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/** @file Bmp085.ino
 * @brief Bmp085 using example.
 *
 * Example showing how to use the Bmp085 library.
 */

#include <Wire.h>

#include <Bmp085.h>

/** @brief Serial port connection velocity.
 */
#define SERIAL_VEL 9600

Bmp085 bmp085;

void setup()
{
    bmp085.setDevice();
    Serial.begin(SERIAL_VEL);
}

void loop()
{
    long temperature = bmp085.getTemperature();
    Serial.print(temperature, DEC);
    Serial.print(" ");
    long pressure = bmp085.calPressure();
    Serial.println(pressure, DEC);
    delay(1000);
}

```

Figura 7.5: Ficheiro de proba do sensor de presión.

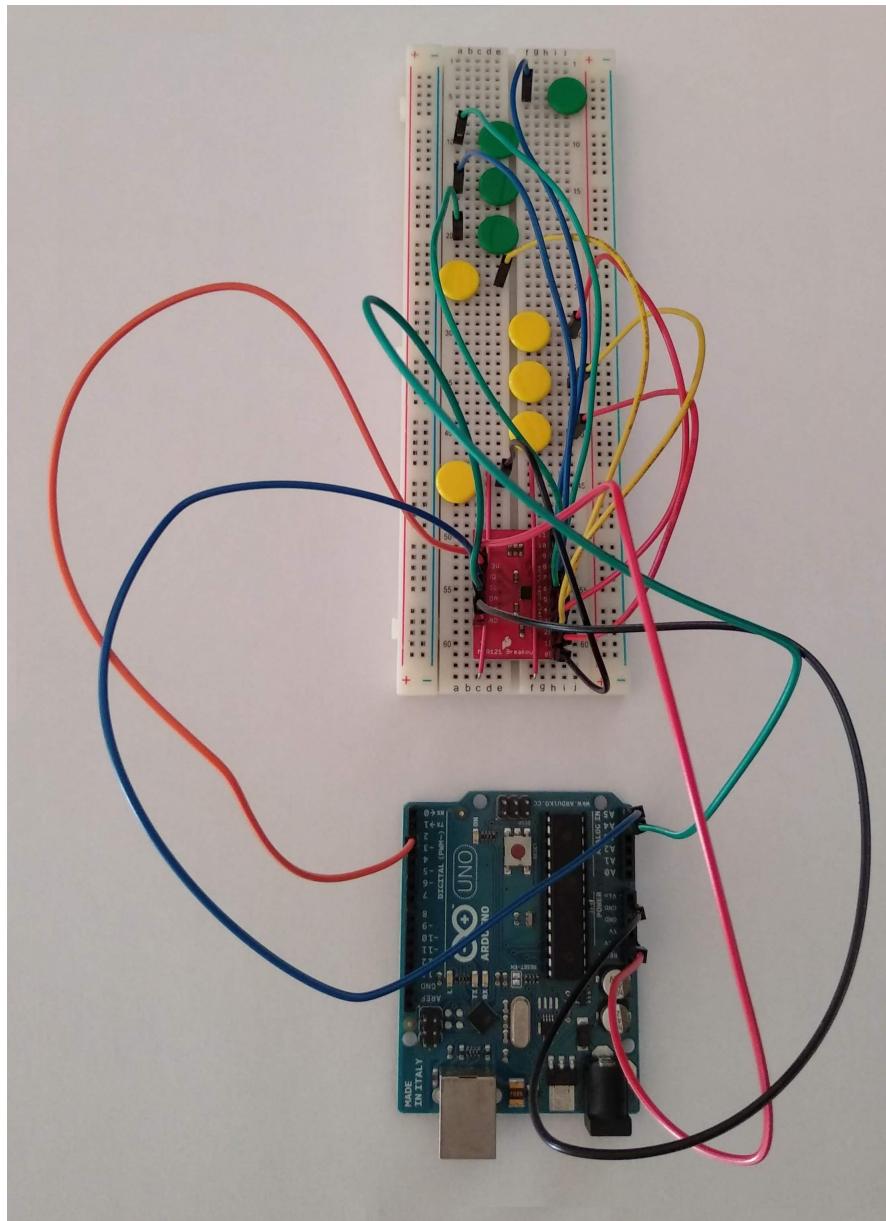


Figura 7.6: Sensores capacitivos.

```

Mpr121.h
X

/** @brief Configure the calibration data.
 *
 * See the [Freescale MPR121] datasheet (Table 2) for more info.
 *
 * Also, see the following application notes for even more info:
 * [AN3994]: Sections A, B, C, D, E and F.
 * [AN3893]: Sections 4.0 and 5.0.
 * [AN3889]: Section Auto-configuration.
 *
 * [Freescale MPR121]: http://cache.freescale.com/files/sensors/doc/
 * data_sheet/MPR121.pdf "Freescale MPR121"
 * [AN3994]: http://cache.freescale.com/files/sensors/doc/app_note/
 * AN3944.pdf "AN3994"
 * [AN3893]: http://cache.freescale.com/files/sensors/doc/app_note/
 * AN3893.pdf "AN3893"
 * [AN3889]: http://cache.freescale.com/files/sensors/doc/app_note/
 * AN3889.pdf "AN3889"
 */
void setCalParam();

/** @brief Setup the I2C communication.
 *
 * Initialize the I2C communication via Wire library and assign an Arduino
 * pin to the MPR121 IRQ pin initializing it.
 */
void setI2C();

/** @brief Write a 1 byte value into the MPR121 write address.
 *
 * @param address Register address.
 * @param value Content to write into the register.
 */
void setRegister(byte address, byte value);

public:

/** @brief Constructor.
 */
Mpr121();

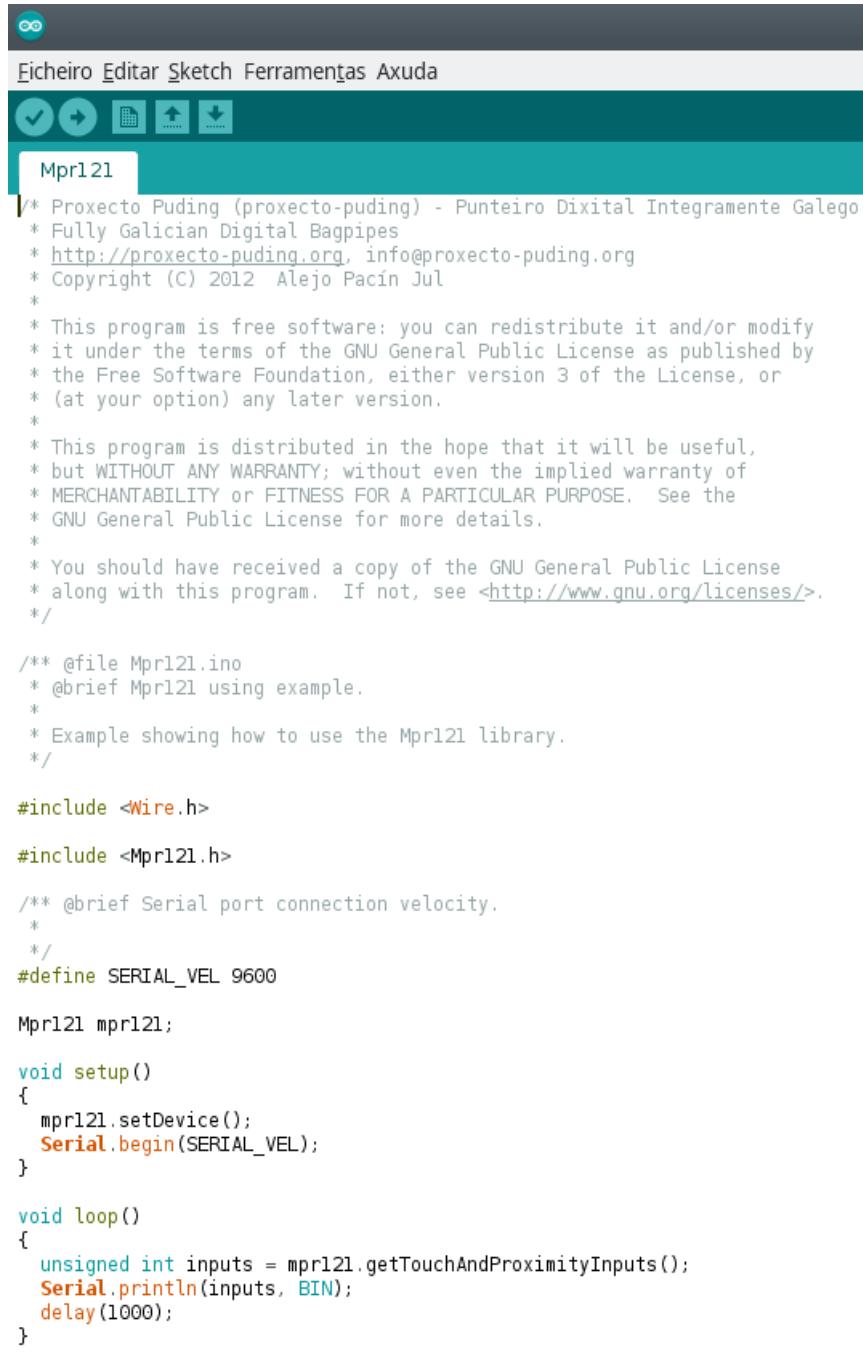
/** @brief Read MPR121 touch and proximity input values.
 *
 * Format: -|-|E12|E11|...|E0 (16 bits)
 * E12: proximity input.
 * E11-0: respective touch values.
 * 1 = touched/proximate.
 * 0 = untouched/not proximate.
 *
 * @return All inputs in an unique 16 bits value. If there is a new value
 * available (IRQ enabled) return this, else return the old value.
 */
unsigned int getTouchAndProximityInputs();

/** @brief Setup the MPR121 pressure sensor.
 *
 * Initialize the communication ports and the configuration parameters.
 */
void setDevice();
};

#endif

```

Figura 7.7: Interface dos sensores capacitivos.

A screenshot of the Arduino IDE interface. The title bar says "Ficheiro Editar Sketch Ferramentas Axuda". Below the toolbar, the sketch name "Mpr121" is displayed in a teal header bar. The main code area contains the following C++ code:

```
/* Proxecto Puding (proxecto-puding) - Punteiro Dixital Integramente Galego
 * Fully Galician Digital Bagpipes
 * http://proxecto-puding.org, info@proxecto-puding.org
 * Copyright (C) 2012 Alejo Pacín Jul
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
 
/** @file Mpr121.ino
 * @brief Mpr121 using example.
 *
 * Example showing how to use the Mpr121 library.
 */
 
#include <Wire.h>
#include <Mpr121.h>

/** @brief Serial port connection velocity.
 */
#define SERIAL_VEL 9600

Mpr121 mpr121;

void setup()
{
    mpr121.setDevice();
    Serial.begin(SERIAL_VEL);
}

void loop()
{
    unsigned int inputs = mpr121.getTouchAndProximityInputs();
    Serial.println(inputs, BIN);
    delay(1000);
}
```

Figura 7.8: Ficheiro de proba dos sensores capacitivos.

gando o porto UART do mesmo.

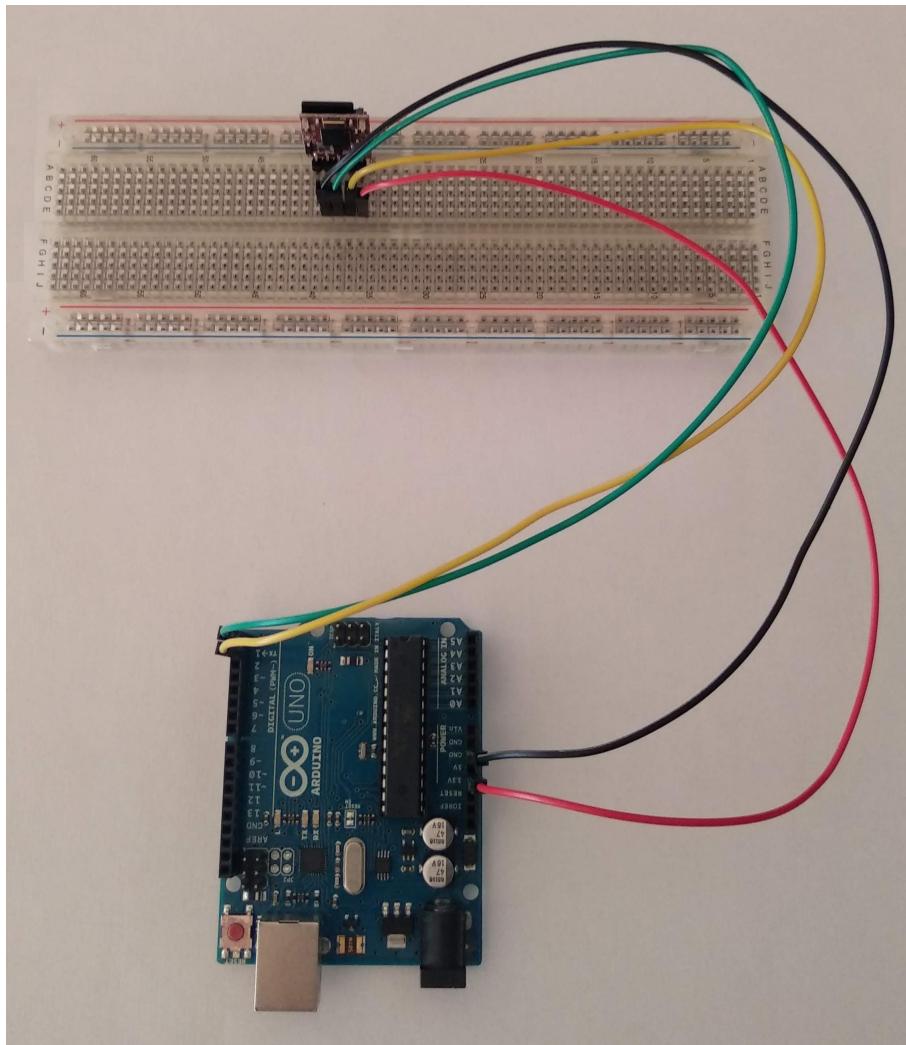


Figura 7.9: Lector de tarxetas.

Ademáis de definir a súa interface pública (figura 7.10) para a obtención da configuración e un ficheiro de proba (figura 7.11) para validar a implementación completa posterior.

7.2.2.1.2. Encapsulamento do hardware

```
G1.h  
public:  
    /** @brief Constructor.  
     *  
     */  
    G1();  
  
    /** @brief Create a copy of the first file into the second.  
     *  
     * @param fromFile String up to 12 chars long. If it is not specified, a  
     * '.' will be assumed between chars 8 and 9.  
     * @param toFile String up to 12 chars long. If it is not specified, a '.'  
     * will be assumed between chars 8 and 9.  
     * @return A boolean indicating if it was copied.  
     */  
    boolean copyFile(char *fromFile, char *toFile);  
  
    /** @brief Get the size of the specified file.  
     *  
     * @param name String up to 12 chars long. If it is not specified, a '.'  
     * will be assumed between chars 8 and 9.  
     * @return Size of the file. If blank or error, return zero.  
     */  
    unsigned long getFileSize(char *name);  
  
    /** @brief Print files into the directory matching the specified name.  
     *  
     * @param name String up to 12 chars long. If it is not specified, a '.'  
     * will be assumed between chars 8 and 9. Wildcards ('*', '?') allowed.  
     * @return A boolean indicating if the operation was successful.  
     */  
    boolean listDirectory(char *name);  
  
    /** @brief Change the name of the fisrt file to the second one.  
     *  
     * @param fromFile String up to 12 chars long. If it is not specified, a  
     * '.' will be assumed between chars 8 and 9.  
     * @param toFile String up to 12 chars long. If it is not specified, a '.'  
     * will be assumed between chars 8 and 9.  
     * @return A boolean indicating if it was moved.  
     */  
    boolean moveFile(char *fromFile, char *toFile);  
  
    /** @brief Obtain and print the device version information.  
     *  
     * Print all the device information (characteristics, capability, etc.).  
     */  
    void printDeviceVersionInfo();  
  
    /** @brief Read the specified file.  
     *  
     * @param name String up to 12 chars long. If it is not specified, a '.'  
     * will be assumed between chars 8 and 9.  
     * @return The file data contained into a String.  
     */  
    String readFile(char *name);  
  
    /** @brief Delete the specified file.  
     *  
     * @return A boolean indicating if the file was removed.  
     */  
    boolean removeFile(char *name);  
  
    /** @brief Return the current user of the device.
```

Figura 7.10: Interface do lector de tarxetas.

```

/*
 * Proxecto Puding (proxecto-puding) - Punteiro Dixital Integramente Galego
 * Fully Galician Digital Bagpipes
 * http://proxecto-puding.org, info@proxecto-puding.org
 * Copyright (C) 2012 Alejo Pacín Jul
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/** @file G1.ino
 * @brief G1 using example.
 *
 * Example showing how to use the G1 library.
 */

#include <Wire.h>

#include <G1.h>

/** @brief Serial port connection velocity.
 */
#define SERIAL_VEL 9600

G1 g1;

void setup()
{
    g1.setDevice();
    Serial.begin(SERIAL_VEL);
}

void loop()
{
    char *directoryName = ".";
    char *fileName = "test.txt";
    boolean append = false;
    String data = "test";
    char *newFileName = "moved.txt";

    g1.printDeviceVersionInfo();
    g1.listDirectory(directoryName);
    g1.writeFile(fileName, append, data);
    g1.moveFile(fileName, newFileName);
    g1.removeFile(newFileName);
    delay(10000);
}

```

Figura 7.11: Ficheiro de proba do lector de tarxetas.

Debido a atoparse nunha situación temperá do prototipado físico, o encapsulamento do hardware realizouse da maneira más leve posible, dada a necesidade de poder montar e desmontar a vontade durante as probas.

Como pode comprobarse nas imaxes do apartado anterior, o único elemento encapsulado por completo sería o router e o resto estaría ó aire, facendo uso de cables de prototipado para evitar ter que soldar e desoldar durante as probas.

Ademáis, cada un dos periféricos foi montado por separado ata a integración completa do hardware, logo da correcta implementación, verificación e validación do mesmo.

7.2.2.2. Prototipo software

Para a o prototipo software operacional tiramos do prototipo deseñado na fase anterior e dos diagramas UML relacionados.

7.2.2.2.1. Desenvolvimento do prototipo

No tocante ó desenvolvemento do software, contamos con dúas partes ben diferenciadas:

- O firmware do dispositivo.
- E a aplicación de configuración.

Como o firmware do dispositivo xa o tratamos na sección anterior, neste apartado centrarémonos no desenvolvemento dun prototipo operacional da aplicación de configuración.

Como xa comentamos anteriormente, para ilo tiramos dos deseños das pantallas e do UML do prototipo da fase de deseño. Combinando os mesmos e facendo uso da lóxica de negocio obtida na fase de análise, chegouse a un deseño UML máis detallado (figura 7.12, onde se incluíron os servizos necesarios para dar cobertura a dita lóxica).

Ditos servizos son:

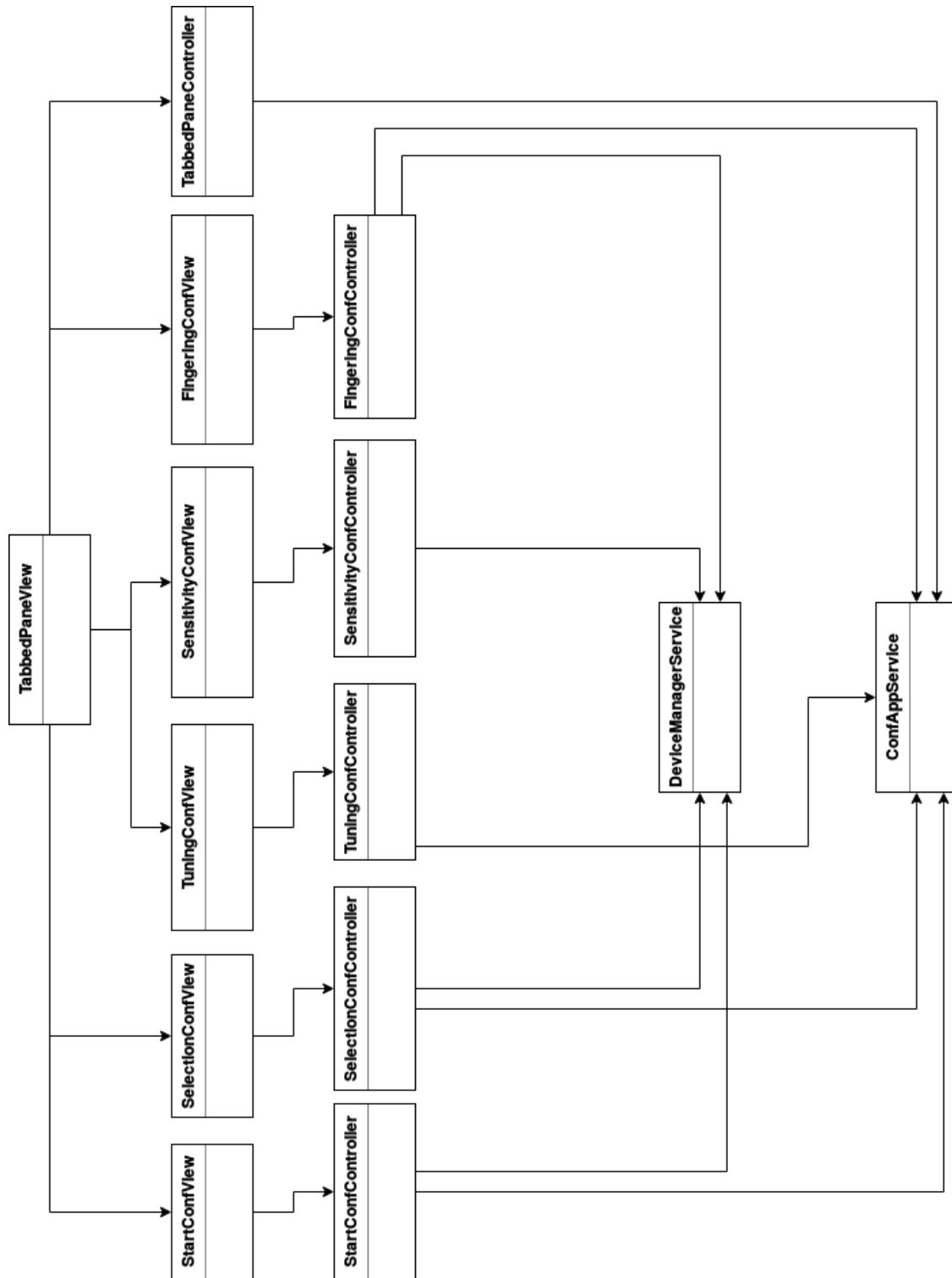


Figura 7.12: Deseño de nivel intermedio.

- O servizo de dispositivos, encargado de xestionar todo o relacionado cos dispositivos hardware en uso: detección, comunicación, etc.
- O servizo de configuración, encargado de xestionar todo o relacionado coa mesma.

Aplicando BDD e baseándonos na interacción do usuario coas pantallas fóronse definindo os distintos casos de uso e a representación conceptual das entidades do modelo para o seu uso polos distintos servizos.

A interface do servizo de dispositivos quedou como reflexan as figuras 7.13 a 7.16.

A interface do servizo de configuración quedou como reflexan as figuras 7.17 a 7.23.

Como se pode apreciar nas interfaces dos servizos, as principais entidades resultantes do modelo son o dispositivo, os distintos tipos de configuración do mesmo e o servidor MIDI, que quedarían como amosan as figuras 7.24 a 7.28.

Definidos xa servizos e entidades do modelo, procedeuse a conectar vista e modelo a través do controlador, tal e como se indica no UML. Desta maneira comprobamos de maneira práctica se encaixaban ben ambas partes e se non esqueciamos nada importante que puidese implicar un retraballo severo de detectarse nunha fase posterior.

7.2.2.2.2. Gravación das mostras

Debido á cantidade de horas a invertir neste apartado, entre a gravación das mostras e a xeración da fonte de son e á dispoñibilidade de outras fontes xa dispoñibles públicamente, aínda que de menor calidade, decidiuse adiar este apartado a unha fase posterior dada a pouca relevancia académica do mesmo e invertir ditas horas en apartados de maior calado.

```
DeviceManagerService.java
1 package org.proxectopuding.gui.model.services;
2
3@ import java.util.List;
9
10 public interface DeviceManagerService {
11
12@     /**
13     * Find all the available bagpipe devices.
14     * @return A set containing all the devices found.
15     */
16     public Set<BagpipeDevice> findBagpipeDevices();
17
18@     /**
19     * Get the ids of all the registered bagpipe devices.
20     * @return A list of bagpipe device ids.
21     */
22     public List<String> getBagpipeDeviceIds();
23
24@     /**
25     * Get the current bagpipe device in use.
26     * @return A bagpipe device.
27     */
28     public BagpipeDevice getSelectedBagpipeDevice();
29
30@     /**
31     * Set the current bagpipe device in use.
32     */
33     public void setSelectedBagpipeDevice(String productId);
34
35@     /**
36     * Find all the device configurations given device id.
37     * @param productId Device id.
38     * @return The requested configurations.
39     */
40     public Set<BagpipeConfiguration> findBagpipeConfigurations(String productId);
41
42@     /**
43     * Find a device configuration given a device id and type.
44     * @param productId Device id.
45     * @param type Device type.
46     * @return A device configuration if found. Null otherwise.
47     */
48     public BagpipeConfiguration findBagpipeConfiguration(String productId, String type);
49
50@     /**
51     * Find a device configuration given a preinitialized one.
52     * @param configuration Device configuration containing at least id and type.
53     * @return A device configuration if found. Null otherwise.
54     */
55     public BagpipeConfiguration findBagpipeConfiguration(BagpipeConfiguration configuration);
56
```

Figura 7.13: Servizo de dispositivos.

```
DeviceManagerService.java 23
57@    /**
58     * Set the configuration for a given device.
59     * @param configuration Device configuration containing at least id and type.
60     * @throws IllegalArgumentException If the provided configuration is null.
61     */
62    public void sendBagpipeConfiguration(BagpipeConfiguration configuration) throws IllegalArgumentException
63
64@    /**
65     * Get a device configuration by id and type.
66     * @param productId Device id.
67     * @param type Device type.
68     * @return The requested configuration if found. Null otherwise.
69     */
70    public BagpipeConfiguration getBagpipeConfiguration(String productId, String type);
71
72@    /**
73     * Get the volume for a given device.
74     * @param productId Device id.
75     * @return A volume value.
76     * @throws IllegalArgumentException If the provided device id is null.
77     */
78    public int getVolume(String productId) throws IllegalArgumentException;
79
80@    /**
81     * Set the volume for a given device.
82     * @param productId Device Id.
83     * @param volume Volume.
84     * @throws IllegalArgumentException If the provided device id is null.
85     */
86    public void setVolume(String productId, int volume) throws IllegalArgumentException;
87
88@    /**
89     * Get the tuning tone for a given device.
90     * @param productId Device id.
91     * @return A tuning tone.
92     * @throws IllegalArgumentException If the provided device id is null.
93     */
94    public int getTuningTone(String productId) throws IllegalArgumentException;
95
96@    /**
97     * Set the tuning tone for a given device.
98     * @param productId Device id.
99     * @param tuningTone Tuning tone.
100    * @throws IllegalArgumentException If the provided device id is null.
101    */
102   public void setTuningTone(String productId, int tuningTone) throws IllegalArgumentException;
```

Figura 7.14: Servizo de dispositivos.

```
DeviceManagerService.java 
104@    /**
105     * Get the tuning octave for a given device.
106     * @param productId Device id.
107     * @return A tuning octave.
108     * @throws IllegalArgumentException If the provided device id is null or
109     * the device is not found.
110    */
111   public int getTuningOctave(String productId) throws IllegalArgumentException;
112
113@ /**
114     * Set the tuning octave for a given device.
115     * @param productId Device id.
116     * @param tuningOctave Tuning octave.
117     * @throws IllegalArgumentException If the provided device id is null.
118    */
119   public void setTuningOctave(String productId, int tuningOctave) throws IllegalArgumentException;
120
121@ /**
122     * Get the fingering types for a given device.
123     * @param productId Device id.
124     * @return Device fingering types.
125     * @throws IllegalArgumentException If the provided device id is null.
126    */
127   public List<Boolean> getFingeringTypesEnabled(String productId) throws IllegalArgumentException;
128
129@ /**
130     * Set the fingering types for a given device.
131     * @param productId Device id.
132     * @param fingeringTypes Device fingering types.
133     * @throws IllegalArgumentException If the provided device id is null.
134    */
135   public void setFingeringTypesEnabled(String productId, List<Boolean> fingeringTypes) throws IllegalArgumentException;
136
137@ /**
138     * Check if the bag is enabled for a given device.
139     * @param productId Device id.
140     * @return A boolean indicating if the bag is enabled.
141     * @throws IllegalArgumentException If the provided device id is null.
142    */
143   public Boolean isBagEnabled(String productId) throws IllegalArgumentException;
144
145@ /**
146     * Set the bag enabled/disabled for a given device.
147     * @param productId Device id.
148     * @param bagEnabled A boolean indicating if the bag is enabled/disabled.
149     * @throws IllegalArgumentException If the provided device id is null.
150    */
151   public void setBagEnabled(String productId, boolean bagEnabled) throws IllegalArgumentException;
```

Figura 7.15: Servizo de dispositivos.

```
DeviceManagerService.java
150  */
151  public void setBagEnabled(String productId, boolean bagEnabled) throws IllegalArgumentException;
152  /**
153@   * Get the drones status for a given device.
154@   * @param productId Device id.
155@   * @return Device drones status.
156@   * @throws IllegalArgumentException If the provided device id is null.
157@   */
158@   public List<Boolean> getDronesEnabled(String productId) throws IllegalArgumentException;
159
160 /**
161@   * Set the drones status for a given device.
162@   * @param productId Device id.
163@   * @param drones Drones statuses.
164@   * @throws IllegalArgumentException If the provided device id is null.
165@   */
166@   public void setDronesEnabled(String productId, List<Boolean> drones) throws IllegalArgumentException;
167
168 /**
169@   * Get the bag pressure limit for a given device.
170@   * @param productId Device id.
171@   * @return A bag pressure limit.
172@   * @throws IllegalArgumentException If the provided device id is null.
173@   */
174@   public int getBagPressure(String productId) throws IllegalArgumentException;
175
176 /**
177@   * Set the bag pressure limit for a given device.
178@   * @param productId Device id.
179@   * @param bagPressure Bag pressure limit.
180@   * @throws IllegalArgumentException If the provided device id is null.
181@   */
182@   public void setBagPressure(String productId, int bagPressure) throws IllegalArgumentException;
183
184 /**
185@   * Get the list of available custom fingerings.
186@   * @param productId Device id.
187@   * @return A list of custom fingerings.
188@   * @throws IllegalArgumentException If the provided device id is null.
189@   */
190@   public List<FingeringsOffset> getFingerings(String productId) throws IllegalArgumentException;
191
192 /**
193@   * Set the list of custom fingerings.
194@   * @param productId Device id.
195@   * @param fingerings A list of fingerings.
196@   */
197@   public void setFingerings(String productId, List<FingeringsOffset> fingerings);
```

Figura 7.16: Servizo de dispositivos.

```
ConfigurationApplicationService.java 33
1 package org.proxectopuding.gui.model.services;
2
3+import java.util.List;
4
5 public interface ConfigurationApplicationService {
6
7+    /**
8     * Get the current device configuration type.
9     * @return A bagpipe configuration type.
10    */
11+   public BagpipeConfigurationType getSelectedBagpipeConfigurationType();
12
13+   /**
14     * Set the current device configuration type.
15     * @param bagpipeConfigurationType A bagpipe configuration type.
16    */
17+   public void setSelectedBagpipeConfigurationType(BagpipeConfigurationType bagpipeConfigurationType);
18
19+   /**
20     * Get the list of reading tones.
21     * @return A list of reading tones.
22    */
23+   public List<String> getReadingTones();
24
25+   /**
26     * Get the current reading tone.
27     * @return The current reading tone.
28    */
29+   public String getReadingTone();
30
31+   /**
32     * Set the current reading tone.
33     * @param readingTone A reading tone.
34    */
35+   public void setReadingTone(String readingTone);
36
37+   /**
38     * Get the list of tuning tones.
39     * @return A list of tuning tones.
40    */
41+   public List<String> getTuningTones();
42
43+   /**
44     * Get a string representation of a given tuning tone.
45     * @param tuningTone A tuning tone as integer between [0, 11].
46     * @return A tuning tone as string.
47     * @throws IllegalArgumentException If the given tuning tone is not in the
48     * expected range.
49    */
50+   public String getTuningTone(int tuningTone) throws IllegalArgumentException;
51
52+   /**
53
54
55
56+   /**
57
```

Figura 7.17: Servizo de configuración.

```
ConfigurationApplicationService.java ✘
56@    /**
57     * Get an integer representation of a given tuning tone.
58     * @param tuningTone A tuning tone as string.
59     * @return A tuning tone as integer.
60     * @throws IllegalArgumentException If the given tuning tone does not
61     * correspond to any mapped string representation.
62     */
63    public int getTuningTone(String tuningTone) throws IllegalArgumentException;
64
65@    /**
66     * Get the default tuning tone.
67     * @return A tuning tone.
68     */
69    public String getDefaultTuningTone();
70
71@    /**
72     * Get the list of tuning octaves.
73     * @return A list of tuning octaves.
74     */
75    public List<Integer> getTuningOctaves();
76
77@    /**
78     * Get the default tuning octave.
79     * @return A tuning octave.
80     */
81    public int getDefaultTuningOctave();
82
83@    /**
84     * Get the list of samples.
85     * @return A list of samples.
86     */
87    public List<String> getSamples();
88
89@    /**
90     * Get the default fingering types.
91     * @return A list of fingering types.
92     */
93    public List<Boolean> getDefaultFingeringTypesEnabled();
94
95@    /**
96     * Get the current sample.
97     * @return A sample.
98     */
99    public String getSample();
100
101@   /**
102     * Set the current sample.
103     * @param sample A sample.
104     */
105    public void setSample(String sample);
```

Figura 7.18: Servizo de configuración.

```
ConfigurationApplicationService.java
~~~
107@    /**
108     * Check the default value for the bag.
109     * @return A boolean indicating if the default bag is enabled.
110     */
111    public Boolean isDefaultBagEnabled();
112
113@    /**
114     * Get the default drones status.
115     * @return A list of drones.
116     */
117    public List<Boolean> getDefaultDronesEnabled();
118
119@    /**
120     * Get the current tuning frequency.
121     * @return A tuning frequency.
122     */
123    public int getTuningFrequency();
124
125@    /**
126     * Set the current tuning frequency.
127     * @param tuningFrequency A tuning frequency.
128     */
129    public void setTuningFrequency(int tuningFrequency);
130
131@    /**
132     * Get the list of tuning modes.
133     * @return A list of tuning modes.
134     */
135    public List<String> getTuningModes();
136
137@    /**
138     * Get the current tuning mode.
139     * @return A tuning mode.
140     */
141    public String getTuningMode();
142
143@    /**
144     * Set the current tuning mode.
145     * @param tuningMode A tuning mode.
146     */
147    public void setTuningMode(String tuningMode);
148
149@    /**
150     * Get the default tuning mode.
151     * @return A tuning mode.
152     */
153    public String getDefaultTuningMode();
154
155@    /**
156     * Get the list of precise tuning notes.
157     * @return A list of precise tuning notes.
158     */

```

Figura 7.19: Servizo de configuración.

```
ConfigurationApplicationService.java 23
155@    /**
156     * Get the list of precise tuning notes.
157     * @return A list of precise tuning notes.
158     */
159    public List<String> getPreciseTuningNotes();
160
161@    /**
162     * Get the current precise tuning note.
163     * @return A precise tuning note.
164     */
165    public String getPreciseTuningNote();
166
167@    /**
168     * Set the current precise tuning note.
169     * @param preciseTuningNote A precise tuning note name.
170     */
171    public void setPreciseTuningNote(String preciseTuningNote);
172
173@    /**
174     * Set the current precise tuning note.
175     * @param preciseTuningNote A precise tuning note value.
176     */
177    public void setPreciseTuningNote(int preciseTuningNote);
178
179@    /**
180     * Reset the list of precise tuning notes.
181     * @return A list of precise tuning notes.
182     */
183    public List<String> resetPreciseTuningNotes();
184
185@    /**
186     * Get the list of precise tuning octaves.
187     * @return A list of precise tuning octaves.
188     */
189    public List<Integer> getPreciseTuningOctaves();
190
191@    /**
192     * Get the current precise tuning octave.
193     * @return A precise tuning octave.
194     */
195    public int getPreciseTuningOctave();
196
197@    /**
198     * Set the current precise tuning octave.
199     * @param preciseTuningOctave A precise tuning octave.
200     */
201    public void setPreciseTuningOctave(int preciseTuningOctave);
202
```

Figura 7.20: Servizo de configuración.

```
ConfigurationApplicationService.java ✘
203@    /**
204     * Get the current precise tuning cents.
205     * @return Precise tuning cents.
206     */
207    public int getPreciseTuningCents();

208@    /**
209     * Set the current precise tuning cents.
210     * @param preciseTuningCents Precise tuning cents.
211     */
212    public void setPreciseTuningCents(int preciseTuningCents);

213@    /**
214     * Get the list of custom fingering notes.
215     * @return A list of custom fingering notes.
216     */
217    public List<String> getCustomFingeringNotes();

218@    /**
219     * Get the current custom fingering note.
220     * @return A custom fingering note.
221     */
222    public String getCustomFingeringNote();

223@    /**
224     * Set the current custom fingering note.
225     * @param customFingeringNote A custom fingering note name.
226     */
227    public void setCustomFingeringNote(String customFingeringNote);

228@    /**
229     * Set the current custom fingering note.
230     * @param customFingeringNote A custom fingering note value.
231     */
232    public void setCustomFingeringNote(int customFingeringNote);

233@    /**
234     * Reset the list of custom fingering notes.
235     * @return A list of custom fingering notes.
236     */
237    public List<String> resetCustomFingeringNotes();

238@    /**
239     * Get the list of custom fingering octaves.
240     * @return A list of custom fingering octaves.
241     */
242    public List<Integer> getCustomFingeringOctaves();
```

Figura 7.21: Servizo de configuración.

```
ConfigurationApplicationService.java
251  /**
252   * Get the current custom fingering octave.
253   * @return A custom fingering octave.
254   */
255  public int getCustomFingeringOctave();
256
257  /**
258   * Set the current custom fingering octave.
259   * @param customFingeringOctave A custom fingering octave.
260   */
261  public void setCustomFingeringOctave(Integer customFingeringOctave);
262
263  /**
264   * Get a list of custom fingering numbers.
265   * @param fingerings A list of custom fingerings.
266   * @return A list of custom fingering numbers.
267   */
268  public List<Integer> getCustomFingeringNumbers(List<FingeringOffset> fingerings);
269
270  /**
271   * Get the current custom fingering number.
272   * @return A custom fingering number.
273   */
274  public int getCustomFingeringNumber();
275
276  /**
277   * Set the current custom fingering number.
278   * @param customFingeringNumber A custom fingering number.
279   */
280  public void setCustomFingeringNumber(int customFingeringNumber);
281
282  /**
283   * Add a new custom fingering number to the list.
284   * @return The new custom fingering number.
285   */
286  public int addCustomFingeringNumber();
287
288  /**
289   * Get a custom fingering by number.
290   * @param customFingeringNumber A custom fingering number.
291   * @return A custom fingering.
292   */
293  public FingeringOffset getCustomFingering(int customFingeringNumber);
294
295  /**
296   * Remove a custom fingering from the list.
297   * @param customFingeringNumber A custom fingering number.
298   */
299  public void removeCustomFingeringNumber(int customFingeringNumber);
300
```

Figura 7.22: Servizo de configuración.

```

ConfigurationApplicationService.java ✘
272     * @return A custom fingering number.
273     */
274     public int getCustomFingeringNumber();
275
276     /**
277      * Set the current custom fingering number.
278      * @param customFingeringNumber A custom fingering number.
279      */
280     public void setCustomFingeringNumber(int customFingeringNumber);
281
282     /**
283      * Add a new custom fingering number to the list.
284      * @return The new custom fingering number.
285      */
286     public int addCustomFingeringNumber();
287
288     /**
289      * Get a custom fingering by number.
290      * @param customFingeringNumber A custom fingering number.
291      * @return A custom fingering.
292      */
293     public FingeringOffset getCustomFingering(int customFingeringNumber);
294
295     /**
296      * Remove a custom fingering from the list.
297      * @param customFingeringNumber A custom fingering number.
298      */
299     public void removeCustomFingeringNumber(int customFingeringNumber);
300
301     /**
302      * Check if the given sensor is selected.
303      * @param sensor Sensor number.
304      * @return A boolean indicating if the sensor is selected.
305      */
306     public boolean isCustomFingeringSensorSelected(int sensor);
307
308     /**
309      * Select or not the provided sensor.
310      * @param sensor Sensor number.
311      * @param isSelected Boolean indicating if the sensor is selected.
312      * @return The associated fingering.
313      */
314     public FingeringOffset setCustomFingeringSensor(int sensor, boolean isSelected);
315
316     /**
317      * Get a MIDI server configuration.
318      * @return A MIDI server configuration.
319      */
320     public MidiServerConfiguration getMidiServerConfiguration();
321

```

Figura 7.23: Servizo de configuración.

```
BagpipeDevice.java
1 package org.proxectopuding.gui.model.entities;
2
3@ import java.util.Set;
4
5 public class BagpipeDevice implements Comparable<BagpipeDevice> {
6
7     private String productId;
8     private Set<BagpipeConfiguration> configurations;
9
10    public BagpipeDevice() {
11        configurations = new TreeSet<BagpipeConfiguration>();
12    }
13
14    public String getProductId() {
15        return productId;
16    }
17
18    public void setProductId(String productId) {
19        this.productId = productId;
20    }
21
22    public Set<BagpipeConfiguration> getConfigurations() {
23        return configurations;
24    }
25
26    public void setConfigurations(Set<BagpipeConfiguration> configurations) {
27        this.configurations = configurations;
28    }
29
30
```

Figura 7.24: Dispositivo.

```
BagpipeConfiguration.java
1 package org.proxectopuding.gui.model.entities;
2
3 public class BagpipeConfiguration implements Comparable<BagpipeConfiguration> {
4
5     private String productId;
6     private String action;
7     private String type;
8
9     public String getProductId() {
10        return productId;
11    }
12
13     public void setProductId(String productId) {
14        this.productId = productId;
15    }
16
17     public String getAction() {
18        return action;
19    }
20
21     public void setAction(String action) {
22        this.action = action;
23    }
24
25     public String getType() {
26        return type;
27    }
28
29     public void setType(String type) {
30        this.type = type;
31    }
32
```

Figura 7.25: Configuración.

```
SelectionConfiguration.java ☐
1 package org.proxectopuding.gui.model.entities;
2
3 import java.util.List;
4
5 public class SelectionConfiguration extends BagpipeConfiguration {
6
7     private int volume;
8     private List<Boolean> fingeringTypes;
9     private boolean bagEnabled;
10    private List<Boolean> dronesEnabled;
11
12    public int getVolume() {
13        return volume;
14    }
15
16    public void setVolume(int volume) {
17        this.volume = volume;
18    }
19
20    public List<Boolean> getFingeringTypes() {
21        return fingeringTypes;
22    }
23
24    public void setFingeringTypes(List<Boolean> fingeringTypes) {
25        this.fingeringTypes = fingeringTypes;
26    }
27
28    public boolean isBagEnabled() {
29        return bagEnabled;
30    }
31
32    public void setBagEnabled(boolean bagEnabled) {
33        this.bagEnabled = bagEnabled;
34    }
35
36    public List<Boolean> getDronesEnabled() {
37        return dronesEnabled;
38    }
39
```

Figura 7.26: Configuración de selección.

```
① MidiServer.java ②
1 package org.proxeclapudin.gui.model.entities.midiServer;
2
3④ import java.io.IOException;
4
5 public interface MidiServer {
6
7     /**
8      * Check if the SoundFont file is downloaded.
9      * @return A boolean indicating if the file is downloaded.
10     * @throws IOException If the SoundFont file is still being downloaded.
11     */
12     /**
13     * default boolean isSoundFontFileDownloaded() throws IOException {
14         return false;
15     }
16
17     /**
18     * Download the SoundFont file to use from the Internet.
19     */
20     /**
21     * default void downloadSoundFontFile() {
22         // Skip.
23     }
24     /**
25     * Check if the SoundFont file is placed into the temporal directory.
26     * @return A boolean indicating if the SoundFont is placed into the
27     * destination directory.
28     */
29     /**
30     * default boolean isSoundFontFileCopied() {
31         return false;
32     }
33     /**
34     * Copy the SoundFont file into the temporal directory.
35     * @return The path where the SoundFont file is placed in. Null otherwise.
36     * @throws IOException If a problem comes up when copying the file.
37     */
38     /**
39     * default String copySoundFontFile() throws IOException {
40         return null;
41     }
42     /**
43     * Get the MIDI server configuration.
44     * @return A MIDI server configuration.
45     */
46     /**
47     * default MidiServerConfiguration getConfiguration() {
48         return null;
49     }
50     /**
51     * Set the MIDI server configuration.
52     * @param configuration A MIDI server configuration.
53     */
54     /**
55     * default void setConfiguration(MidiServerConfiguration configuration) {
56         // Skip.
57     }
58     /**
59     * Generate a list of strings containing the command and the parameters
60     * needed to execute the MIDI server with the current configuration.
61     * @return A command list.
62     */
63     List<String> getCommand();
64 }
```

Figura 7.27: Servidor MIDI.

```

1 package org.proxectopuding.gui.model.entities.midiServer;
2
3 import java.util.Set;
4
5 public class MidiServerConfiguration {
6
7     private boolean useRealSamples;
8     private int tuningTone;
9     private int tuningOctave;
10    private int tuningFrequency;
11    private boolean usePureIntonationMode;
12    private boolean useContinuousVibrato;
13    private Set<PreciseTuning> preciseTunings;
14
15    public MidiServerConfiguration() {
16
17        useRealSamples = false;
18        tuningTone = 0;
19        tuningOctave = 4;
20        tuningFrequency = 440;
21        usePureIntonationMode = false;
22        useContinuousVibrato = false;
23        preciseTunings = new TreeSet<PreciseTuning>();
24    }
25
26    public boolean useRealSamples() {
27        return useRealSamples;
28    }
29
30    public void setUseRealSamples(boolean useRealSamples) {
31        this.useRealSamples = useRealSamples;
32    }
33}
34
35
36

```

Figura 7.28: Configuración do servidor MIDI.

7.3. Desenvolvemento e validación do seguinte nivel do producto

7.3.1. Simulacóns, modelos e programas de proba

As probas realizadas neste nivel do producto consistiron inicialmente en realizar as mesmas probas da fase anterior sobre os novos prototipos para verificar que a implementación cumpría coa especificación de requisitos e de que non pasabamos nada por alto.

Posteriormente, de maneira máis formal e seguindo a metodoloxía de desenvolvemento citada (BDD/TDD), definíronse unha serie de probas funcionais a nivel de unidade, integración e aceptación a nivel de servizos e para ambos sistemas, de maneira que se puidesen validar e verificar o seu funcionamento tanto antes coma despois de implementar a lóxica do mesmos. Ditas probas serán explicadas en detalle ó final deste capítulo.

7.3.2. Deseño detallado

7.3.2.1. Deseño hardware

Contando xa cun deseño formal e detallado do producto hardware dende fases anteriores e non tendo detectado novos problemas, decidimos seguir empregando o *Prototipo 3 hardware* (figura 6.18) coma deseño do mesmo.

7.3.2.2. Deseño software

Baseándose no deseño de nivel medio do prototipo software operacional (figura 7.12) e nas carencias e melloras detectadas durante as probas realizadas sobre a implementación de dito prototipo, chegouse a un deseño de nivel medio máis detallado onde se incluíron novos servizos que agrupan funcionalidades relacionadas que ven non estaban reflexadas de maneira explícita ou teñen un peso máis secundario.

Na figura 7.29 pode apreciarse o diagrama UML deste deseño máis completo. Entre ditos servizos figuran:

- O servizo de internacionalización, encargado de proporcionar as mensaxes da aplicación no idioma do usuario.
- O servizo de notificacións, encargado de xestionar a comunicación entre as distintas partes da interface mediante eventos.
- O servizo de navegación web, que permite xestionar a apertura da documentación da aplicación directamente no navegador do sistema.
- O servizo MIDI, que se encarga de xestionar todo o relacionado coas operacións sobre o servidor MIDI.

Aplicando BDD e baseándonos na interacción do usuario coas pantallas fóronse definindo os distintos casos de uso e a representación conceptual das entidades do modelo para o seu uso polos distintos servizos.

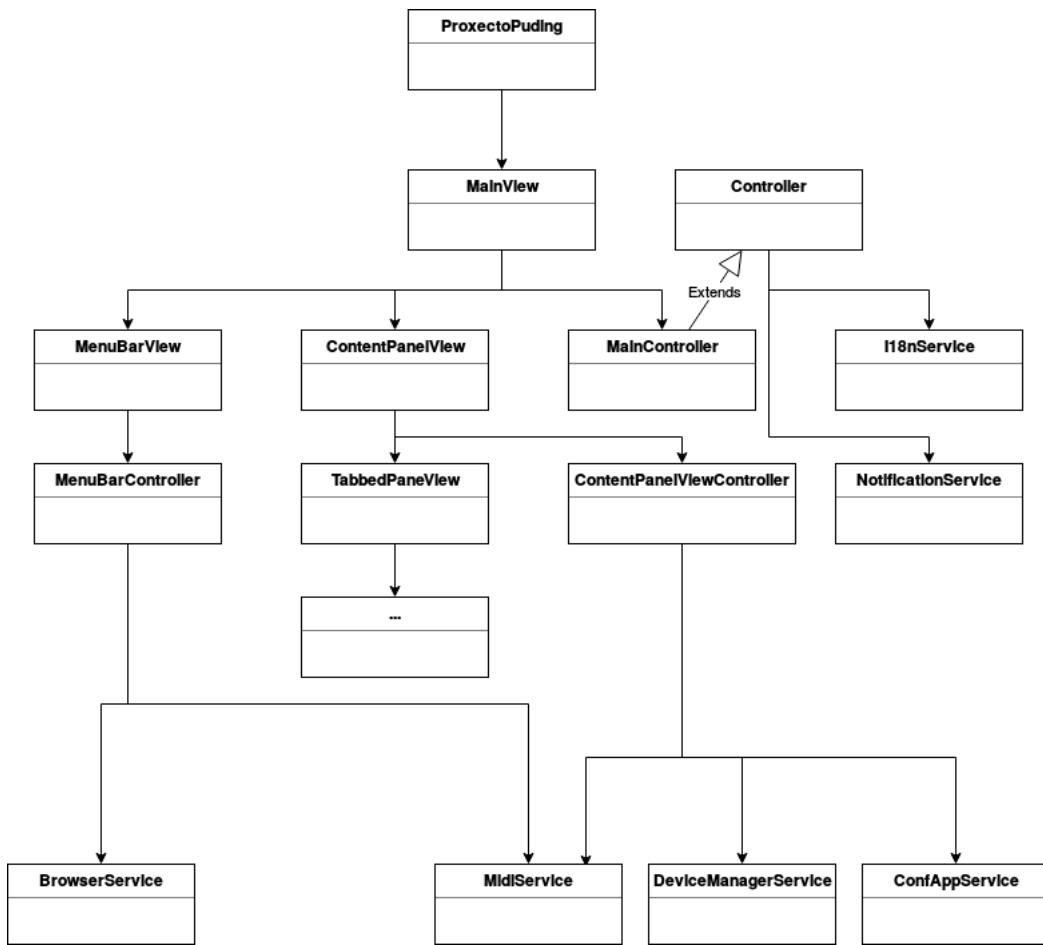


Figura 7.29: Deseño detallado.

A interfaces dos distintos servizos quedaron tal e como reflexan as figuras 7.30 a 7.33.

```

1 package org.proxectopuding.gui.model.services;
2
3 public interface I18nService {
4
5     /**
6      * Get the text for a provided id located to the OS language.
7      * @param translationId Key that identifies the text to be retrieved.
8      * @return A i18n text.
9     */
10    public String getTranslation(String translationId);
11
12 }

```

Figura 7.30: Servizo de internacionalización.

```

1 package org.proxectopuding.gui.model.services;
2
3 import java.beans.PropertyChangeListener;
4
5 public interface NotificationService {
6
7     /**
8      * Send a notification to all the listeners subscribed to a property change event.
9      * @param source The bean that fired the event.
10     * @param propertyName - The programmatic name of the property that was changed.
11     * @param propertyName The programmatic name of the property that was changed.
12     * @param newValue The new value of the property.
13     */
14    public void sendNotification(Object source, Notification propertyName, Object newValue);
15
16    /**
17     * Register a listener to be notified of any bound property updates.
18     * @param propertyName The programmatic name of the property that is going to be listened.
19     * @param listener A source bean to be notified of any bound property updates.
20     */
21    public void addNotificationListener(Notification propertyName, PropertyChangeListener listener);
22
23    /**
24     * Unregister a listener from being notified of any bound property updates.
25     * @param propertyName The programmatic name of the property it wants to stop being notified about.
26     * @param listener The source bean to stop being notified of any bound property updates.
27     */
28    public void removeNotificationListener(Notification propertyName, PropertyChangeListener listener);
29
30
31 }

```

Figura 7.31: Servizo de notificacións.

Redefinidos xa servizos e entidades do modelo, procedeuse a reconectar vista e modelo a través do controlador facendo uso dos mesmos, tal e como se indica no UML. Desta maneira recomprobamos de maneira práctica se encaixaban ben ambas partes e se non esqueciamos nada importante que puidese implicar un retraballo severo de detectarse nunha fase posterior.

```
1 package org.proxectopuding.gui.model.services;
2
3 public interface BrowserService {
4
5     /**
6      * Open the "About" using the default browser.
7      */
8     void openAboutUrl();
9
10    /**
11     * Open the "Bagpipe API" using the default browser.
12     */
13    void openBagpipeApiUrl();
14
15    /**
16     * Open the "Configuration Application API" using the default browser.
17     */
18    void openConfAppApiUrl();
19
20    /**
21     * Open the "User Manual" using the default browser.
22     */
23    void openUserManualUrl();
24
25    /**
26     * Open the "Technical Manual" using the default browser.
27     */
28    void openTechnicalManualUrl();
29
30    /**
31     * Open the provided URI using the default browser.
32     * @param uri URI to be shown.
33     */
34    void openUri(String uri);
35 }
```

Figura 7.32: Servizo de navegación web.

```
MidiService.java 23
1 package org.proxectopuding.gui.model.services;
2
3 import org.proxectopuding.gui.model.entities.midiServer.MidiServerConfiguration;
4
5 public interface MidiService {
6
7     /**
8      * Start the MIDI service.
9      * @return The process within the MIDI service is running.
10     */
11    public Process start();
12
13    /**
14     * Restart the MIDI service.
15     * @return The process within the MIDI service is running.
16     */
17    public Process restart();
18
19    /**
20     * Stop the MIDI service.
21     */
22    public void stop();
23
24    /**
25     * Get the MIDI server configuration.
26     * @return A MIDI server configuration.
27     */
28    public MidiServerConfiguration getConfiguration();
29
30    /**
31     * Set the MIDI server configuration.
32     * @param configuration A MIDI server configuration.
33     */
34    public void setConfiguration(MidiServerConfiguration configuration);
35
36 }
```

Figura 7.33: Servizo MIDI.

7.3.3. Ensamblado e codificación

7.3.3.1. Ensamblado

Debido a unha serie de razóns de peso (detallados na sección de incidencias), descartouse a integración completa e ensamblado de todo o hardware nunha única montaxe:

- Con respecto á integración, durante a fase final de codificación e integración software xurdiron unha serie de problemas que simplemente aconsellaron non acoplar todos os componentes na mesma montaxe.
- Con respecto ó encapsulado, ó non ter toda a montaxe integrada, perde a súa razón de ser, ademáis de ser contraproducente mentres non se rematen as probas, tal e como se comentou antes. Este sería o paso final en calquera caso.

7.3.3.2. Codificación

Como se comentou con anterioridade, debido á metodoloxía empregada (BDD ou TDD segundo o caso), a codificación tanto do firmware coma da aplicación de configuración foi o último paso de todos, dado que primeiramente se definiron as probas.

Unha vez tivemos definidas as probas, para o que fomos en sentido *top-down* ata o nivel de servizo, cambiamos o sentido a *bottom-up* para facer a implementación dos mesmos en ámbolos casos.

7.3.3.2.1. Hardware

A codificación do firmware do hardware levouse a cabo por partes e de menor a maior dificultade:

- Sensor de presión.
 - Sensores capacitivos.
 - Lector de tarxetas.
-

- Receptor: configuración e reproducción MIDI.

Aínda que a maior dificultade común a todas elas foi a complexidade da documentación e a falta dun entorno de desenvolvemento serio.

Comezando polo **sensor de presión**, este caso é a excepción que confirma a regra. Ó ser unha peza desenvolvida por Bosh (compañía moi seria en canto a desenvolvemento de sensores), a folla de especificacións [37] é impecable.

O proceso consistiu en:

- Solicitar a lectura da presión.
- Definir e ler os rexistros da ROM onde se almacenan os coeficientes de calibración do dispositivo.
- Definir e ler os rexistros da ROM onde se almacenan os coeficientes de calibración do dispositivo.
- Ler a temperatura sen compensar (4,5 ms).
- Definir o modo de lectura da presión e ler a presión sen compensar. Como nos interesaba a velocidade e o baixo consumo máis que a precisión, optouse polo modo de ultra baixo consumo (4,5 ms), co que se obteñen valores bastante desviados, pero como o que precisamos a nivel de lóxica de negocio é un valor umbral (ou dous se optamos por un comparador con histérese), que o valor sexa máis ou menos real non é o importante.
- Calcular a temperatura real aplicando os coeficientes de calibración.
- Calcular a presión real aplicando os coeficientes de calibración.
- Devolver a presión real, que é o valor que realmente nos interesa, invertindo un tempo de arredor de 9 ms, que está por debaixo dos 100 ms que se adoitan considerar como tempo real duro para intrumentación MIDI e incluso por debaixo dos 10 ms que se considera o óptimo e que serían imperceptibles.

Dado que tanto as direccións dos rexistros dos coeficientes de calibración e a aplicación dos mesmos e o pseudo-código especificado (figura 7.34) estaban moi

ben explicados, foi relativamente sinxelo facer funcionar o sensor e que pasase as probas definidas anteriormente. Os resultados das mesmas comentaranse no apartado de probas unitarias.

Finalmente, entre documentación, código e probas, obtivemos un módulo de moi baixo nível en C de arredor de 500 liñas.

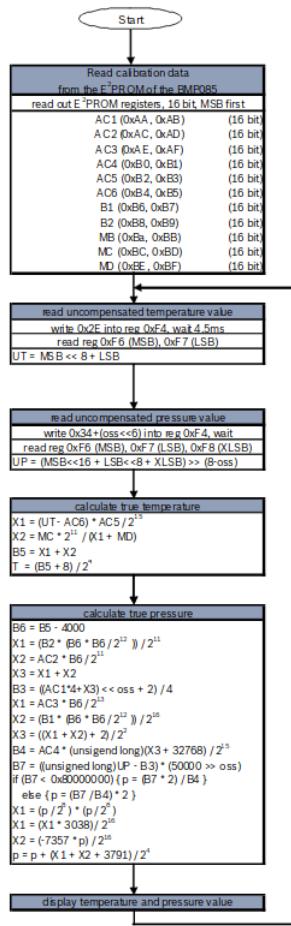


Figura 7.34: Pseudo-código do sensor de presión.

Continuando polos **sensores capacitivos**, a folla de especificacións [38], de Freescale neste caso, tamén é de calidade, polo que facilitou o desenvolvemento.

O proceso consistiu en:

- Solicitar a lectura da presión mediante I2C.
- Definir e ler os rexistros da ROM onde se almacenan as configuracións e os valores medidos do dispositivo.
- Ler os valores dos sensores.
- Devolver os valores medidos.

A configuración dos rexistros foi de lonxe a parte más complexa e tediosa. Houbo que configurar medio cento de rexistros agrupados nunha decena de grupos funcionais e facelo nunha orde concreta, antes de poder facer calquera outra operación.

En xeral, precisamos obter o valor dos 9 bits/sensores menos significativos da placa (que empregaremos para as dixitacións), máis o 13º, sensor virtual de presencia (que empregaremos para o vibrato).

A lectura dos valores dos sensores realiza-se primeiro en bruto, sendo pasados a continuación por tres filtros de distinto tipo, como se pode apreciar na figura 7.35.

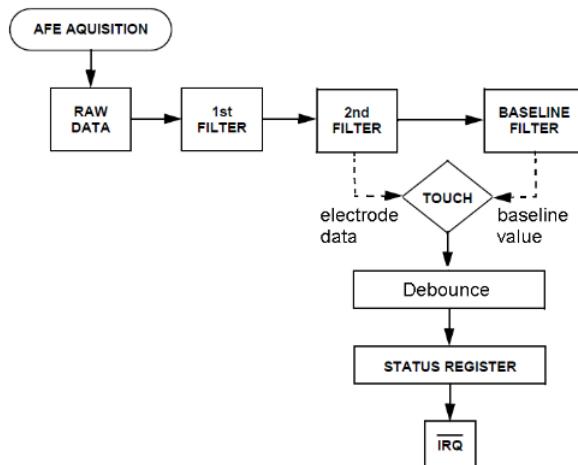


Figura 7.35: Medición dos sensores capacitivos.

O sensor de proximidade é un sensor virtual que calcula o seu valor en función do valor dos outros 12 sensores físicos, de maneira que podemos empregalo para

calcular pequenas variacións ou vibracións dos son en función de se hai outros dedos próximos ós sensores que non están sendo empregados na dixitación.

Outro punto moi importante que descubrimos na folla de especificacións é que os sensores son auto-calibrables, polo que o requisito sobre o axuste da sensibilidade do sensores, logo de facer probas con distinta xente e vendo que os valores eran correctos, quedou cuberto pola parte hardware sen necesidade de cubrilo dende o software de configuración, motivo polo que se revisou a pantalla de configuración da sensibilidade do punteiro, eliminando a sección de configuración da sensibilidade dos sensores e quedando como se amosa na figura 7.36.

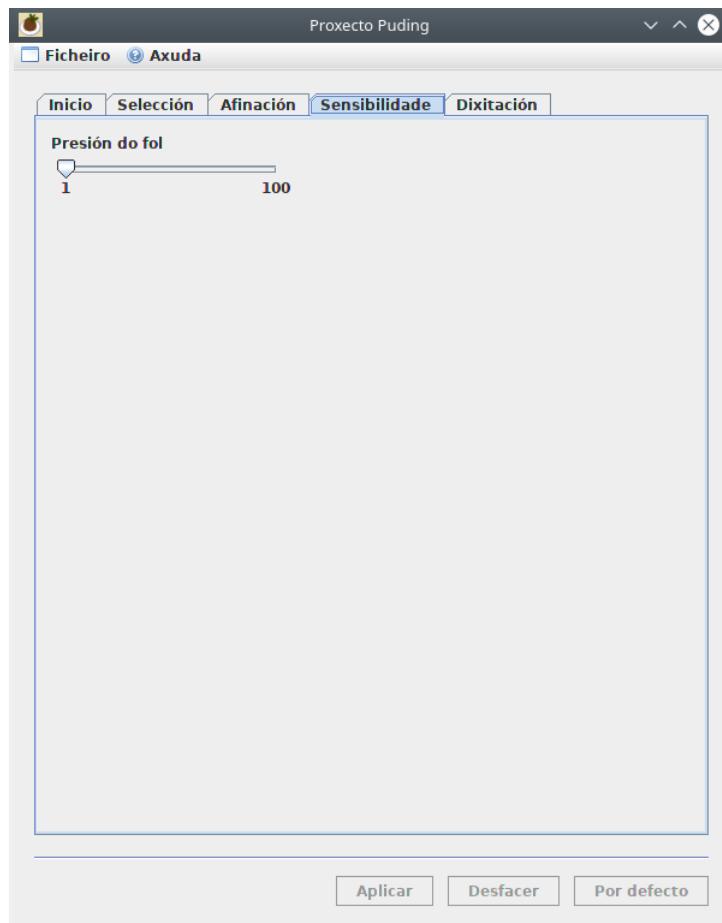


Figura 7.36: Configuración da sensibilidade do punteiro.

Feitos funcionar os sensores e que pasasen as probas definidas anteriormente,

os resultados das mesmas comentaranse no apartado de probas unitarias.

Finalmente, entre documentación, código e probas, obtivemos un módulo de moi baixo nivel en C de arredor de 500 liñas.

Segundo polo **lector de tarxetas**, a documentación proporcionada por 4D Systems [51] é claramente escasa e incompleta. En xeral, especifica relativamente ben os casos *happy path*, ou o camiño típico, pero en moitos casos non especifica qué pasa en caso de erro ou a descripción resulta ambigua, casos que cando falamos de hardware, son custosos cando non imposibles de reproducir de maneira forzada, o cal dificulta unha implementación completa e correcta do firmware.

O proceso consistiu en:

- Inicializar a comunicación por porto serie.
- Esperar a que o dispositivo arrinque (500 ms).
- Esperar a que o dispositivo detecte a velocidade do porto serie.
- Ler ou escribir información na tarxeta de memoria.

A complexidade deste módulo residiu principalmente na incertidume na que nos movemos todo o rato debido á falta de documentación, que nos obrigou a estar probando cada pequeno paso que dabamos e a tomar decisións un pouco drásticas para cubrirnos en saúde.

Basicamente, as operacións que precisamos principalmente de todas as disponíveis, serían as de lectura e escritura dun ficheiro.

Como exemplo, a documentación di que se poden ler/escribir datos en bloques de 1 a 50/100 bytes (con/sen ACK), pero non indica en ningún momento qué pasa co último bloque se o tamaño total non é múltiplo do tamaño de bloque.

Neste caso concreto, por exemplo, optamos pola opción más segura pero menos eficiente de todas: ACKs por bloques de 1 byte. Como o tamaño dos ficheiros

é de poucos centos de bytes, a penalización apenas é perceptible.

Neste caso, aínda que se fixo a implementación completa do módulo, non foi posible facelo funcionar, dado que logo de aplicar o reset inicial, o dispositivo non consigue atopar a velocidade da conexión automaticamente, motivo polo cal o resto de operacións non poden continuar.

Finalmente, entre documentación, código e probas, obtivemos un módulo de moi baixo nivel en C de arredor de 700 liñas.

E rematando polo **receptor**, que á súa vez se divide entre a lóxica de configuración e a de reproducción.

Feitos funcionar os outros módulos, era o momento de implementar a lóxica de negocio da capa inmediatamente superior, é dicir, a do receptor, onde se empregrarían o módulo do lector de tarxetas para a configuración e os módulos de presión e capacitivos para a reproducción.

Cabe dicir que tampouco foi tarefa fácil, dado que a documentación oficial de Arduino deixa moito que desexar. Conta coa información básica más ou menos ben estructurada, pero en canto se quere ir un pouco máis alá, a única opción é consultar os foros técnicos ou aplicar o proceso de proba-erro. Moitas das dúbihidas que se plantexan nos foros e que se ve que son bastante repetitivas poderían pasar a formar parte da documentación oficial coma proceso de mellora.

Comezamos pola parte de reproducción, que a priori semellaba máis sinxela a nivel de comunicación, aínda que requería da consulta de máis sensores.

O proceso consistiu en:

- Configurar o dispositivo. Cargar os módulos e mapear as dixitacións.
 - Ler os valores dos módulos (presión e sensores capacitivos activos).
 - Determinar se a nota e os bordóns deben soar ou non en función do valor do sensor de presión.
-

- Calcular a nota a reproducir en función da dixitación equivalente ós sensores activos.
- Enviar o comando MIDI correspondente para reproducir o parar nota e bordóns.

Para o primeiro o punto, definíronse un par de matrices que conteñen as dixitacións típicas da gaita galega (aberto -figura 7.37- e pechado -figura 7.38-), sendo a primeira parte a representación binaria equivalente dos valores aportados polos sensores capacitivos e a segunda o desprazamento en semitonos respecto da nota base ou tonalidade na que se está tocando.

Para o segundo punto simplemente lemos os valores dos sensores coma durante as probas.

Para determinar se deben soar ou non nota e bordóns, comparamos o valor obtido de presión co umbral definido como límite. Neste caso poderíamos incluso facer un comparador con histérese definindo un límite alto e un límite baixo, en función de se estamos enchendo ou baleirando o fol, pero non o vimos preciso.

Para calcular a nota e os bordóns, partimos da nota base ou tonalidade e aplicamos o desprazamento en semitonos obtido da matriz xeral de dixitacións co que obtemos a nota a reproducir (no caso dos bordóns, o desprazamento é sempre fixo).

Para xerar o comando MIDI a enviar ó servidor MIDI, o que fixemos foi facer uso dunha librería referenciada entre as da documentación de Arduino [58]. Desta librería empregamos basicamente tres comandos:

- Note On.
- Note Off.
- Send pitch bend. Este último empregouse xunto co sensor de proximidade para saber cando aplicar unha pequena variación na nota que se denomina vibrato.

```

▼/** @brief Matrix containing all the "aberto" offsets.
 *
 * Matrix containing the correspondent offset for all the possible "aberto" inputs.
 */
▼const short aberto[][] = {((B00000001 * 256) + B11111111, -1),
    ((B00000001 * 256) + B11111110, 0),
    ((B00000001 * 256) + B11101110, 1),
    ((B00000001 * 256) + B01111110, 2),
    ((B00000001 * 256) + B11111101, 3),
    ((B00000001 * 256) + B11111000, 4),
    ((B00000001 * 256) + B11110000, 5),
    ((B00000001 * 256) + B11110001, 5),
    ((B00000001 * 256) + B11110010, 5),
    ((B00000001 * 256) + B11110011, 5),
    ((B00000001 * 256) + B11011000, 6),
    ((B00000001 * 256) + B11010001, 6),
    ((B00000001 * 256) + B11010010, 6),
    ((B00000001 * 256) + B11011011, 6),
    ((B00000001 * 256) + B11011100, 6),
    ((B00000001 * 256) + B11011101, 6),
    ((B00000001 * 256) + B11011110, 6),
    ((B00000001 * 256) + B11011111, 6),
    ((B00000001 * 256) + B11010000, 7),
    ((B00000001 * 256) + B11010001, 7),
    ((B00000001 * 256) + B11010010, 7),
    ((B00000001 * 256) + B11010011, 7),
    ((B00000001 * 256) + B11010011, 7),
    ((B00000001 * 256) + B11010000, 8),
    ((B00000001 * 256) + B10110000, 8),
    ((B00000001 * 256) + B10110001, 8),
    ((B00000001 * 256) + B10110010, 8),
    ((B00000001 * 256) + B10110011, 8),
    ((B00000000 * 256) + B11010000, 8),
    ((B00000000 * 256) + B11010001, 8),
    ((B00000000 * 256) + B11010010, 8),
    ((B00000000 * 256) + B11010011, 8),
    ((B00000000 * 256) + B10010000, 9),
    ((B00000000 * 256) + B10010001, 9),
    ((B00000000 * 256) + B10010010, 9),
    ((B00000000 * 256) + B10010011, 9),
    ((B00000000 * 256) + B10010011, 9),
    ((B00000000 * 256) + B01010000, 10),
    ((B00000001 * 256) + B01010001, 10),
    ((B00000001 * 256) + B01010010, 10),
    ((B00000001 * 256) + B01010011, 10),
    ((B00000000 * 256) + B10010000, 10),
    ((B00000000 * 256) + B10010001, 10),
    ((B00000000 * 256) + B10010010, 10),
    ((B00000000 * 256) + B10010011, 10),
    ((B00000000 * 256) + B10010011, 10),
    ((B00000000 * 256) + B00010000, 11),
    ((B00000001 * 256) + B00010001, 11),
    ((B00000001 * 256) + B00010010, 11),
    ((B00000001 * 256) + B00010011, 11),
    ((B00000000 * 256) + B11111111, 11),
    ((B00000000 * 256) + B00000000, 12),
    ((B00000000 * 256) + B00000001, 12),
    ((B00000000 * 256) + B00000010, 12),
    ((B00000000 * 256) + B00000011, 12),
    ((B00000001 * 256) + B01111110, 12),
    ((B00000000 * 256) + B11111110, 12),
    ((B00000000 * 256) + B11101110, 13),
    ((B00000000 * 256) + B11111100, 14),
    ((B00000000 * 256) + B11111010, 15),
    ((B00000000 * 256) + B11111000, 16),
    ((B00000000 * 256) + B11110000, 17)};

```

Figura 7.37: Matriz de dixitación aberta.

```

/** @brief Matrix containing all the "pechado" offsets.
 *
 * Matrix containing the correspondent offset for all the possible "pechado" inputs.
 */
const short pechado[][2] = {((B00000001 * 256) + B11111111, -1),
                           ((B00000001 * 256) + B11111110, 0),
                           ((B00000001 * 256) + B11101110, 1),
                           ((B00000001 * 256) + B11111100, 2),
                           ((B00000001 * 256) + B11111010, 3),
                           ((B00000001 * 256) + B11111011, 4),
                           ((B00000001 * 256) + B11110110, 5),
                           ((B00000001 * 256) + B11011010, 6),
                           ((B00000001 * 256) + B11011110, 7),
                           ((B00000001 * 256) + B10110110, 8),
                           ((B00000000 * 256) + B11011110, 8),
                           ((B00000001 * 256) + B10011110, 9),
                           ((B00000001 * 256) + B10011110, 10),
                           ((B00000000 * 256) + B10011110, 10),
                           ((B00000001 * 256) + B00011110, 11),
                           ((B00000000 * 256) + B11111111, 11),
                           ((B00000001 * 256) + B11111110, 12),
                           ((B00000000 * 256) + B11111110, 12),
                           ((B00000001 * 256) + B01101110, 13),
                           ((B00000000 * 256) + B11101110, 13),
                           ((B00000000 * 256) + B11111100, 14),
                           ((B00000000 * 256) + B11111010, 15),
                           ((B00000000 * 256) + B11111000, 16),
                           ((B00000000 * 256) + B11110000, 17)};
```

Figura 7.38: Matriz de dixitación pechada.

Lista a parte de reproducción, continuamos coa parte de configuración.

O proceso consistiu en:

- Ler a configuración almacenada no dispositivo xa cargado previamente.
- Configurar o dispositivo en función da última configuración gardada lida no paso anterior.
- Comprobar se a aplicación de configuración nos pide que nos identifiquemos.
- Comprobar se a aplicación nos solicita a configuración actual, previa ou por defecto e enviala no seu caso.
- Comprobar se a aplicación nos envía unha nova configuración.

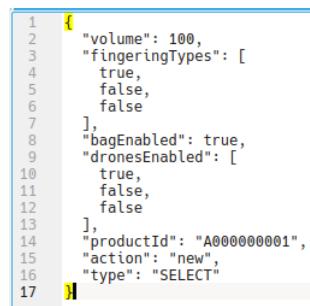
Para o intercambio e almacenamento da configuración precisábase un formato lixeiro pero ó mesmo tempo facilmente flexible e modificable, polo que se optou por empregar JSON.

O tratamento de JSON en Arduino está todavía un pouco verde. No momento no que se avaliou a disponibilidade de librerías para o seu tratamiento a máis coñecida era *aJSON* [59] cuxo desenvolvemento parece morto a día de hoxe e que ainda conta con decenas de incidencias abertas.

Actualmente, a máis empregada aínda que tamén con decenas de incidencias abertas é *Arduino Json* [60]. Pero como a implementación se fixo antes de que isto ocorrese, empregouse aJSON.

Como xa comentamos anteriormente, cada dispositivo garda a súa propia configuración. A nivel de implementación, garda un ficheiro por cada tipo de configuración (selección, sensibilidade, etc.) e de acción (actual, previa ou por defecto), de maneira que cada dispositivo conta con máis dunha ducia de pequenos ficheiros de configuración almacenados.

Cada un deses ficheiros de configuración non é máis ca un ficheiro de texto plano que contén unha tipo de configuración en formato JSON (figura 7.39).



```

1  "volume": 100,
2  "fingeringTypes": [
3    true,
4    false,
5    false
6  ],
7  "bagEnabled": true,
8  "dronesEnabled": [
9    true,
10   false,
11   false
12 ],
13 "productId": "A000000001",
14 "action": "new",
15 "type": "SELECT"
16
17

```

Figura 7.39: Ficheiro de configuración en formato JSON.

Os fluxos de identificación e configuración comentaranse na parte software, xa que é máis doadoo entendelos relacionándoos coa interacción do usuario sobre as pantallas.

Finalmente, entre documentación e código, obtivemos un módulo de baixo nível en C de arredor de 1100 liñas.

Sen embargo, á hora de querer cargar o código na placa atopámonos con problemas, que explicaremos no apartado de incidencias, e que fixeron que tiveramos que, logo de aplicar o análise de riscos e alternativas, optaramos por empregar unha versión simplificada, de arredor de 700 liñas.

As probas de integración da parte hardware leváronse a cabo xunto coa parte software, motivo polo que se postergan os resultados obtidos ata a finalización da aplicación de configuración que, ademáis de xestionar a mesma é a encargada de levantar o servidor MIDI cos parámetros correctos.

7.3.3.2.2. Software

Lista a implementación da parte hardware, tocoulle o turno á parte software.

En xeral, para toda a implementación e as probas da aplicación empregouse (ou está en trámites de actualizarse pouco a pouco) inxección de dependencias (mediante *Guice* [61]), obxectos inmutables (mediante *Guava* [62]) e programación funcional (mediante expresións lambda de Java 8).

Se lembramos dos apartados anteriores, tiñamos pendente a implementación da capa de servizos hacia abaixo. Concretamente:

- O servizo de dispositivos.
- O servizo de configuración.
- O servizo de internacionalización.
- O servizo de notificacións.
- O servizo de navegación web.
- E o servizo MIDI.

O **servizo de dispositivos** é o encargado de xestionar todo o relacionado cos dispositivos hardware en uso: detección, comunicación, etc.

Para poder comunicarnos cos dispositivos a través do router XBee, que se conecta ó ordenador mediante USB, precisabamos dunha librería de comunicación serie para Java.

A base de buscar e de falar con xente do mundillo, demos coa librería RXTX de GNU [63], ó parecer, a única dispoñible daquelas e constatamos que a comunicación serie é un dos grandes olvidados en Java.

A instalación da mesma foi farragosa e o seu uso infructuoso, pois non fomos quen de conseguir conectividade, polo cal a implementación final quedou en espera a falta de alternativas.

Finalmente, uns anos despois, apareceu unha nova librería, *Java Simple Serial Connector* ou *jSSC* [64] que, lonxe de ser perfecta, ofrece menos dificultades tanto para a súa instalación coma para o seu uso. En todo caso, leva parada hai moitos anos.

Ademáis da librería de comunicación serie, era precisa unha librería para o tratamento de JSON, pois era o formato a empregar durante as comunicacóns.

Neste caso, a elección estivo bastante clara dende un principio. A librería *Gson* [65] foi a escollida, por ser amplamente empregada no mundo Java.

Tendo claras xa as librerías, a outra parte interesante é o fluxo de datos entre a aplicación e os dispositivos. Concretamente para os casos de:

- Detección de dispositivos.
- Recuperación da configuración dun dispositivo.
- Envío dunha nova configuración a un dispositivo.

Os diagramas de fluxo dos casos anteriores poden apreciarse nas figuras 7.40 a 7.42.

No diagrama de detección de dispositivos (figura 7.40) podemos apreciar como cando o usuario busca dispositivos se envía unha baliza de recoñecemento que chega a todos os dispositivos conectados ó router mediante broadcasting. Dita baliza indica ós dispositivos que deben identificarse.

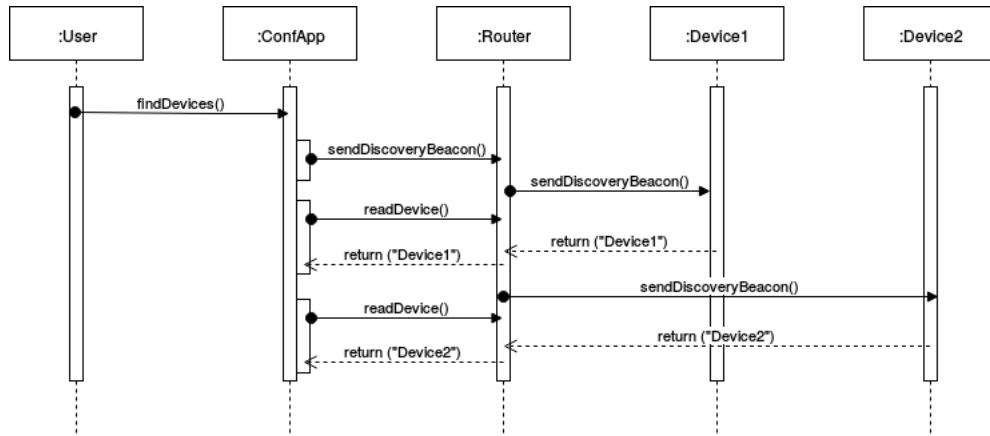


Figura 7.40: Detección de dispositivos.

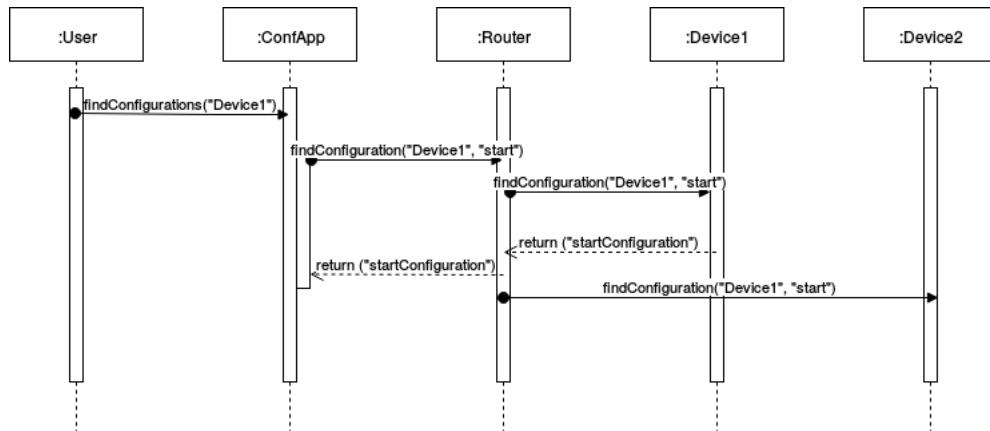


Figura 7.41: Recuperación da configuración dun dispositivo.

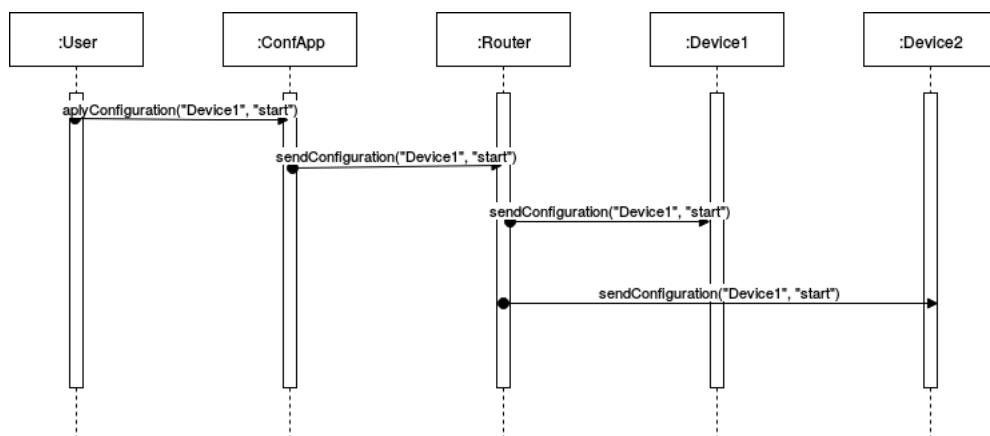


Figura 7.42: Envío dunha nova configuración a un dispositivo.

Unha vez identificados pola aplicación e seleccionado un deles polo usuario, recupérase a configuración do mesmo, indicando na petición o identificador do dispositivo, que queremos a configuración actual e o tipo en cada caso (figura 7.41).

Cando o usuario precise axustar algúin parámetro, enviará a nova configuración ó dispositivo, indicando na petición o identificador do dispositivo, que é unha nova configuración e o tipo en cada caso, que acto seguido será aplicada polo dispositivo, segundo se amosa na figura 7.42.

O **servizo de configuración** é o encargado de xestionar todo o relacionado coa configuración da aplicación, servindo de ponte entre esta e outros servizos coma o de dispositivos ou o MIDI.

Encárgase de cousas como de precargar os datos de configuración dos menús da aplicación ou de xerar a configuración do servidor MIDI en función dos valores da configuración actual.

En canto á precarga dos datos de configuración dos menús da aplicación, a dificultade estribaba en que é dependente da tonalidade de lectura do usuario polo que para as mesmas claves/notas, temos valores/nomes distintos en función da tonalidade na que o usuario teña configurada a aplicación.

A xeración da configuración para o servidor MIDI non reviste máis complexidade que a de facer un mapeo dos valores da configuración dos que depende o mesmo para funcionar o máis precisamente posible (figura 7.43).

O **servizo de internacionalización** é o encargado de proporcionar as mensaxes da aplicación no idioma do usuario.

Actualmente adopta o idioma do sistema e conta con soporte para galego e inglés (por defecto).

O **servizo de notificacións**, encargado de xestionar a comunciación entre as



```

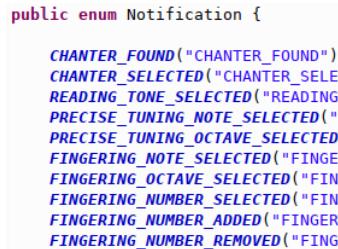
614@  @Override
615  public MidiServerConfiguration getMidiServerConfiguration() {
616
617      LOGGER.log(Level.INFO, "Getting MIDI server configuration");
618      MidiServerConfiguration configuration = new MidiServerConfiguration();
619
620      String productId =
621          deviceManagerService.getSelectedBagpipeDevice().getProductId();
622      int tuningTone = deviceManagerService.getTuningTone(productId);
623      int tuningOctave = deviceManagerService.getTuningOctave(productId);
624      boolean usePureIntonationMode =
625          tuningModes.get(tuningMode).equals(TuningMode.PURE);
626      boolean useRealSamples = !samples.get(sample).equals(Sample.MIDI);
627      boolean useContinuousVibrate = false;
628
629      configuration.setTuningTone(tuningTone);
630      configuration.setTuningOctave(tuningOctave);
631      configuration.setTuningFrequency(tuningFrequency);
632      configuration.setUsePureIntonationMode(usePureIntonationMode);
633      configuration.setUseRealSamples(useRealSamples);
634      configuration.setUseContinuousVibrate(useContinuousVibrate);
635      configuration.setPreciseTunings(
636          new LinkedHashSet<PreciseTuning>(preciseTunings.values()));
637
638      return configuration;
639  }

```

Figura 7.43: Mapeo da configuración do servidor MIDI.

distintas partes da interface mediante eventos.

Basicamente trátase dun patrón observador cun conxunto de notificacións predefinidas (figura 7.44) ás que se pode subscribir calquera compoñente da aplicación que o precise.



```

public enum Notification {
    CHANTER_FOUND("CHANTER_FOUND"),
    CHANTER_SELECTED("CHANTER_SELECTED"),
    READING_TONE_SELECTED("READING_TONE_SELECTED"),
    PRECISE_TUNING_NOTE_SELECTED("PRECISE_TUNING_NOTE_SELECTED"),
    PRECISE_TUNING_OCTAVE_SELECTED("PRECISE_TUNING_OCTAVE_SELECTED"),
    FINGERING_NOTE_SELECTED("FINGERING_NOTE_SELECTED"),
    FINGERING_OCTAVE_SELECTED("FINGERING_OCTAVE_SELECTED"),
    FINGERING_NUMBER_SELECTED("FINGERING_NUMBER_SELECTED"),
    FINGERING_NUMBER_ADDED("FINGERING_NUMBER_ADDED"),
    FINGERING_NUMBER_REMOVED("FINGERING_NUMBER_REMOVED")
}

```

Figura 7.44: Notificacións.

Resulta moi útil e simplifica enormemente a actualización de valores entre pestanas da aplicación.

O **servizo de navegación web**, xestiona a apertura da documentación da aplicación (web do proxecto, manual de usuario, APIs, etc.) directamente no navegador do sistema (figura 7.45).

```

1 package org.proxectopuding.gui.model.services.impl;
2
3@import java.util.logging.Level;
4
5
6 public class BrowserServiceImpl implements BrowserService {
7
8     private static final Logger LOGGER = Logger.getLogger(BrowserServiceImpl.class.getName());
9
10    private final BrowserManager browserManager;
11
12    private final String aboutUrl;
13    private final String bagpipeApiUrl;
14    private final String confAppApiUrl;
15    private final String userManualUrl;
16    private final String technicalManualUrl;
17
18    @Inject
19    public BrowserServiceImpl(BrowserManager browserManager,
20                             ConfigurationManager configurationManager) {
21
22         this.browserManager = browserManager;
23
24         aboutUrl = configurationManager.getAboutUrl();
25         bagpipeApiUrl = configurationManager.getBagpipeApiUrl();
26         confAppApiUrl = configurationManager.getConfAppApiUrl();
27         userManualUrl = configurationManager.getUserManualUrl();
28         technicalManualUrl = configurationManager.getTechnicalManualUrl();
29     }
30
31
32    @Override
33    public void openAboutUrl() {
34
35        LOGGER.log(Level.INFO, "Opening about url");
36        browserManager.openUri(aboutUrl);
37    }
38}

```

Figura 7.45: Servizo de navegación web.

O **servizo MIDI**, xestiona todo o relacionado coas operacións sobre o servidor MIDI: configurar, levantar e parar o servidor.

Neste caso, o que se buscou foi contar cunha representación única xenérica do servidor MIDI e da súa configuración que logo se particularizou segundo o sistema operativo anfitrión e o servidor MIDI particular mediante o uso dun patrón estratexia.

Para o servidor MIDI, aínda que poderíamos ter aplicado un distinto para cada sistema operativo, optamos por buscar e empregar o mesmo para todos, de ser posible, particularizando logo o comando de execución para cada sistema operativo segundo o caso.

A opción escollida foi **Timidity** [66], un sintetizador MIDI de amplo uso e con soporte para gran cantidade de sistemas operativos.

Como alternativa dispoñemos de FluidSynth [67], máis moderno, pero con menos funcionalidades relacionadas co proxecto ca Timidity, motivo polo que apostamos por este último.

Timidity conta con soporte para uso de fontes de son alternativas (son real), táboas de frecuencias personalizadas (afinación natural e axuste fino por nota), vibrato continuo, etc., tal e como se pode comprobar na figura 7.46.



```

1 package org.proxectopuding.gui.model.entities.midiServer;
2
3 import java.io.File;
4
5 public class MidiServerUnix implements MidiServer {
6     private static final Logger LOGGER = Logger.getLogger(MidiServerUnix.class);
7     private static final String REAL_SAMPLES = "--config-file=";
8     private static final String FREQ_TABLE = "--freq-table=";
9     private static final String VIBRATO = "--vibrato";
10    private static final String CHORUS = "-chorus=n";
11    private static final String DEF_CONFIG_FILE_PATH =
12        "/etc/timidity/timidity.cfg";
13    private static final String EXECUTABLE_PATH = "/usr/bin/timidity";
14    private static final String TEMP_CONFIG_FILE_PATH = "/tmp";
15
16    private final MidiServerGeneral midiServer;
17    private String configFilePath;
18    private String tablePath;
19
20    public MidiServerUnix(MidiServerGeneral midiServer) {
21        this.midiServer = midiServer;
22        this.midiServer.setTempConfigFilePath(TEMP_CONFIG_FILE_PATH);
23    }
24
25    @Override
26    public List<String> getCommand() {
27        List<String> command = new ArrayList<String>();
28
29        // Executable.
30        command.add(EXECUTABLE_PATH);
31
32        if (getConfiguration() != null) {
33            // Real samples.
34            if (getConfiguration().useRealSamples()) {
35                configFilePath = getRealSamplesConfigFilePath();
36                if (configFilePath != null) {
37                    command.add(REAL_SAMPLES + configFilePath);
38                }
39            }
40
41            // Tuning mode, tuning frequency and precise tunings.
42            tablePath = getFrequencyTablePath(getConfiguration());
43            if (tablePath != null) {
44                command.add(FREQ_TABLE + tablePath);
45            }
46
47            // Continuous vibrato.
48            if (getConfiguration().useContinuousVibrato()) {
49                // Only for hardware not supporting manual vibrato.
50                command.add(VIBRATO);
51                command.add(CHORUS);
52            }
53        }
54
55        return command;
56    }
57}

```

Figura 7.46: Servizo MIDI: sistemas UNIX.

Como partes interesantes desta implementación, destacar por unha banda o xestor de descargas de fontes de son, dado que para fontes de son realistas o peso

é demasiado elevado como para almacenala no propio repositorio da aplicación e, ademáis, gañamos poder manter actualizada a fonte de son en todos os clientes.

Tamén é destacable o cálculo da táboa de frecuencias para afinación natural. É teoría musical, pero que precisamos plasmar na nosa implementación de maneira programática.

A nivel código non deixa de ser unha relación matemática entre a tonalidade base e o resto da súa escala musical.

No caso dunha escala temperada, estas relacóns as mesmas para toda a escala, é dicir, se un tono se divide en 9 comas, os semitonos están equiespaciados a razón de 4,5 comas por semitono [68].

No caso dunha escala natural (tamén chamada xusta ou pura), os semitonos están espaciados a razón de 4 ou 5 comas segundo o caso [69].

Isto fai que os harmónicos (ou frecuencias harmónicas) producidos por cada nota varíen moito dun tipo de escala ó outro e, polo tanto, a mesma escala reproducida cunha afinación ou a outra soe de maneira moi distinta.

Dentro da propia afinación xusta, existen moitas variantes xa dende os tempos de Pitágoras pero, en xeral, o que se busca é obter o maior número de tríadas posibles que xeren intervalos consonantes, é dicir, que os harmónicos ou frecuencias harmónicas xerados polas tres notas do acorde ou tríada, sexan consonantes entre si [70].

As relacóns más empregadas para a afinación xusta son as da figura 7.47.

Finalmente, entre documentación, código (de alto, medio e baixo nivel) e probas (de unidade, integración e aceptación), obtivemos unha aplicación de arredor de 15000 liñas de código en Java.

A parte de probas será reflexada con detalle na sección correspondente.

```
1 package org.proxectopuding.gui.model.utils;
2
3@ import java.io.File;[]
11
12 public class MidiUtils {
13
14     private static final Logger LOGGER = Logger.getLogger(MidiUtils.class.getName());
15
16     private static final String FREQ_TABLE = "freetable.tbl";
17@     private static double[] pureIntonationRatios =
18         {1.0/1, 16.0/15, 8.0/7, 6.0/5, 5.0/4, 4.0/3, 16.0/11, 3.0/2, 8.0/5,
19         5.0/3, 7.0/4, 15.0/8};
20
21@     /**
22      * Generate a table of tuning frequencies to be used by the MIDI server.
23      * @param tone Base tone.
24      * @param octave Base octave.
25      * @param frequency Base frequency.
26      * @param usePureIntonation Indicate if it is going to be used pure/just or
27      * tempered intonation.
28      * @param preciseTunings List of precise custom tunings for particular
29      * notes.
30      * @param path Directory path where the table is going to be stored.
31      * @return The path to the generated table file.
32      */
33@     public String generateFrequencyTable(int tone, int octave,
34             int frequency, boolean usePureIntonation,
35             Set<PreciseTuning> preciseTunings, String path) {
```

Figura 7.47: Relaciones afinación xusta.

7.3.4. Probas de unidade

Segundo os principios de TDD, as probas de unidade foron paso previo á implementación real tanto dos módulos hardware coma dos servizos software.

7.3.4.1. Hardware

Como comentamos anteriormente, durante o desenvolvemento do prototipo hardware operacional creáronse uns pequenos sketches de exemplo a modo de probas unitarias, para poder validar o funcionamiento de cada módulo por separado antes de proceder á súa integración na montaxe do receptor.

No caso do sensor de presión, probamos a obtención da temperatura e presión ambiente, ainda que para o caso que nos ocupa, con probar a presión sería suficiente (figura 7.48).

Os resultados obtidos pódense ver na figura 7.49. Salientar que arroxa uns valores algo altos froito de que está calibrado para obter o resultado o máis rápido posible, reducindo polo tanto a precisión ó mínimo. Pero como o que precisamos é básicamente un valor para comparar con un umbral, é perfectamente válido.



```

/* Proxecto Puding (proxecto-puding) - Punteiro Dixital Integramente Galego
 * Fully Galician Digital Bagpipes
 * http://proxecto-puding.org, info@proxecto-puding.org
 * Copyright (C) 2012 Alejo Pacín Jul
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/** @file Bmp085.ino
 * @brief Bmp085 using example.
 *
 * Example showing how to use the Bmp085 library.
 */

#include <Wire.h>

#include <Bmp085.h>

/** @brief Serial port connection velocity.
 */
#define SERIAL_VEL 9600

Bmp085 bmp085;

void setup()
{
    bmp085.setDevice();
    Serial.begin(SERIAL_VEL);
}

void loop()
{
    long temperature = bmp085.getTemperature();
    Serial.print(temperature, DEC);
    Serial.print(" ");
    long pressure = bmp085.calPressure();
    Serial.println(pressure, DEC);
    delay(1000);
}

```

Figura 7.48: Probas de unidade do sensor de presión.

No caso dos sensores capacitivos, probamos a ler os valores dos nove bits menos significativos máis o bit do sensor de proximidade. Neste caso, a entrada é manual e precisa da intervención do usuario, que debe validar visualmente que o resultado obtivo se corresponde coa entrada manual (figura 7.50).

Os resultados obtidos pódense ver na figura 7.51. Validados os resultados de forma visual, comprobouse que eran correctos.

No caso do lector de tarxetas, probamos a ler a información do dispositivo, listar o directorio principal, escribir, mover e ler un ficheiro. Aínda que con probar a ler e escribir sería suficiente para cubrir as nosas necesidades (figura 7.52).

Os resultados obtidos pódense ver na figura 7.53. Validados os resultados de forma visual e revisada externamente a tarxeta de memoria, comprobouse que neste caso os resultados eran incorrectos, pois logo de facer o reset correctamente, non foi posible conseguir que o dispositivo fose quen de atopar a velocidade da conexión automaticamente. Na imaxe pódese apreciar como envía continuamente o valor U (ou 0x55), pero nunca recibe confirmación por parte do lector de tarxetas, motivo polo cal non pode continuar operando.

7.3.4.2. Software

Como comentamos tamén anteriormente, durante o desenvolvemento do prototipo software operacional creáronse baterías de tests unitarios a nivel de servizo.

Creouse unha batería de tests por servizo (figura 7.54) e cubriuse alomenos o *happy path* de cada unha das súas sinaturas.

Para simular o comportamento de cada método empregouse unha librería de *mocking*. A escollida, pola súa popularidade dentro do mundo Java foi *Mockito* [71].

Vexamos un exemplo. Como se pode apreciar na figura 7.55, nas precondicións

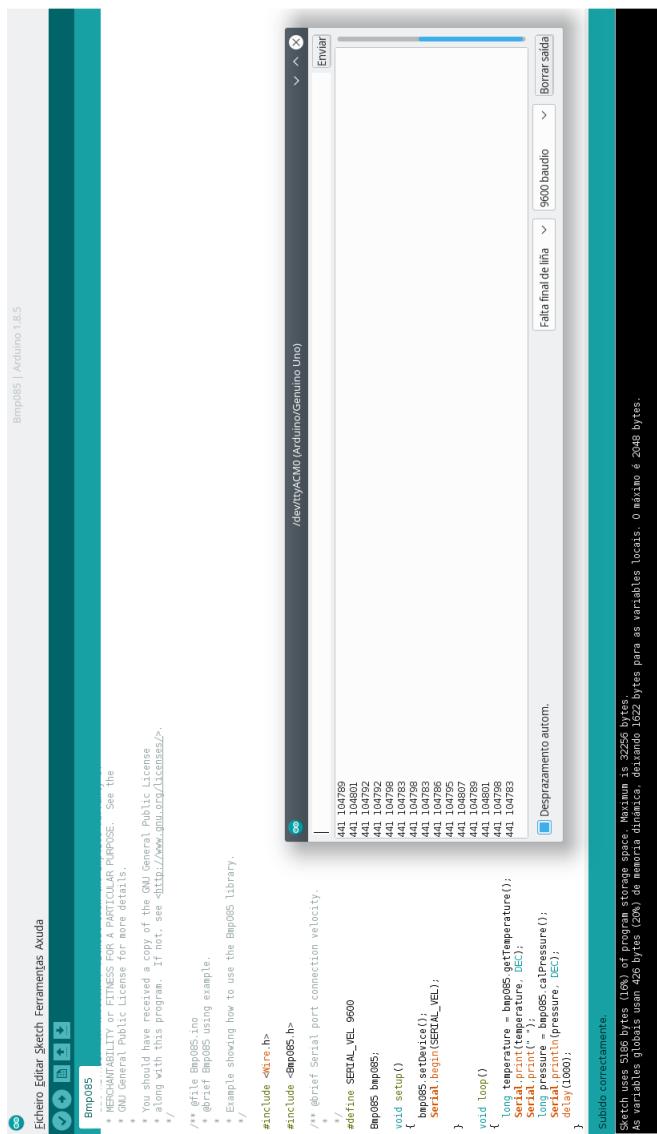
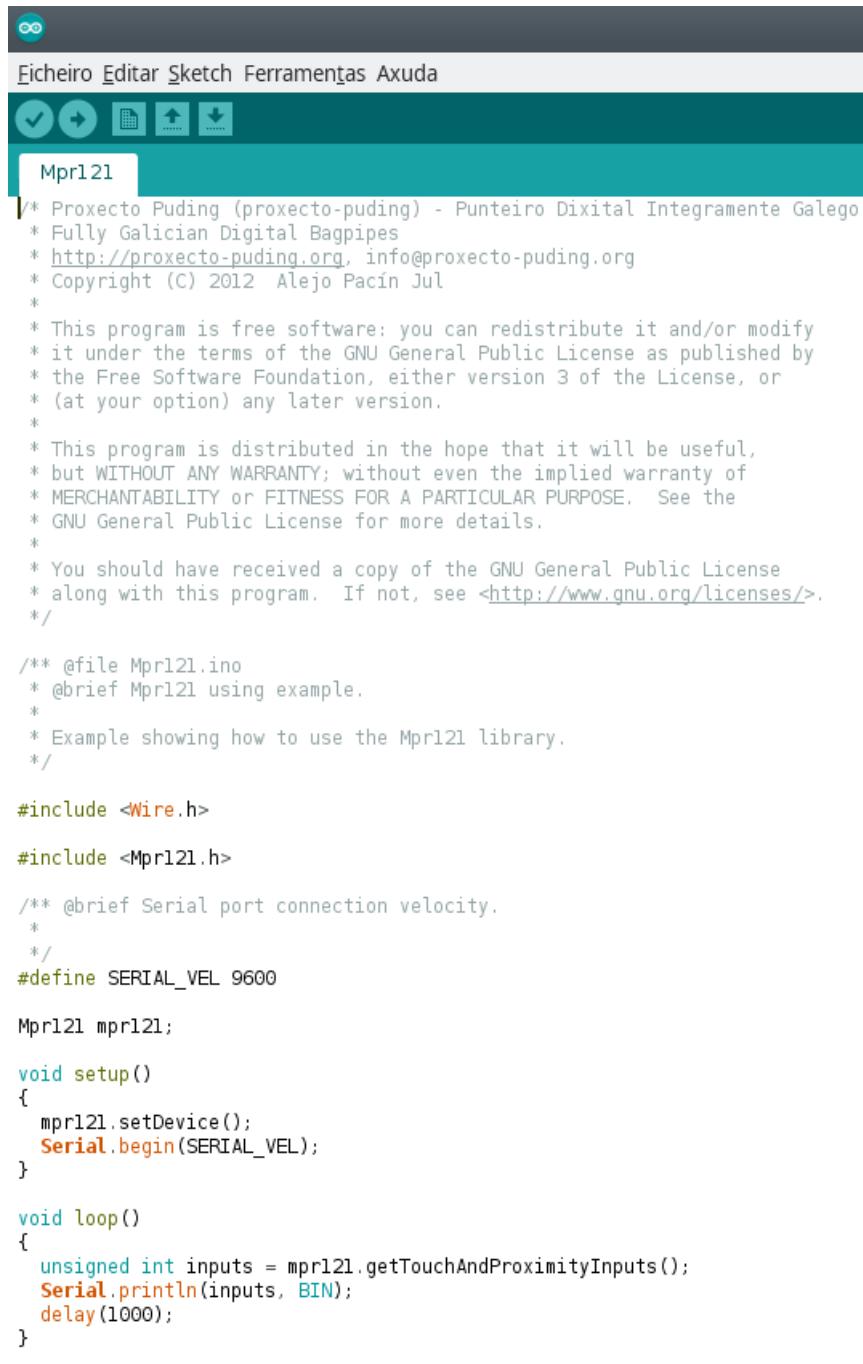


Figura 7.49: Resultado das probas de unidade do sensor de presión.

A screenshot of the Arduino IDE interface. The title bar says "Ficheiro Editar Sketch Ferramentas Axuda". Below the toolbar, the sketch name "Mpr121" is displayed in a teal header bar. The main code area contains the following C++ code:

```
/* Proxecto Puding (proxecto-puding) - Punteiro Dixital Integramente Galego
 * Fully Galician Digital Bagpipes
 * http://proxecto-puding.org, info@proxecto-puding.org
 * Copyright (C) 2012 Alejo Pacín Jul
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
 
/** @file Mpr121.ino
 * @brief Mpr121 using example.
 *
 * Example showing how to use the Mpr121 library.
 */
 
#include <Wire.h>
#include <Mpr121.h>

/** @brief Serial port connection velocity.
 */
#define SERIAL_VEL 9600

Mpr121 mpr121;

void setup()
{
    mpr121.setDevice();
    Serial.begin(SERIAL_VEL);
}

void loop()
{
    unsigned int inputs = mpr121.getTouchAndProximityInputs();
    Serial.println(inputs, BIN);
    delay(1000);
}
```

Figura 7.50: Probas de unidade dos sensores capacitivos.

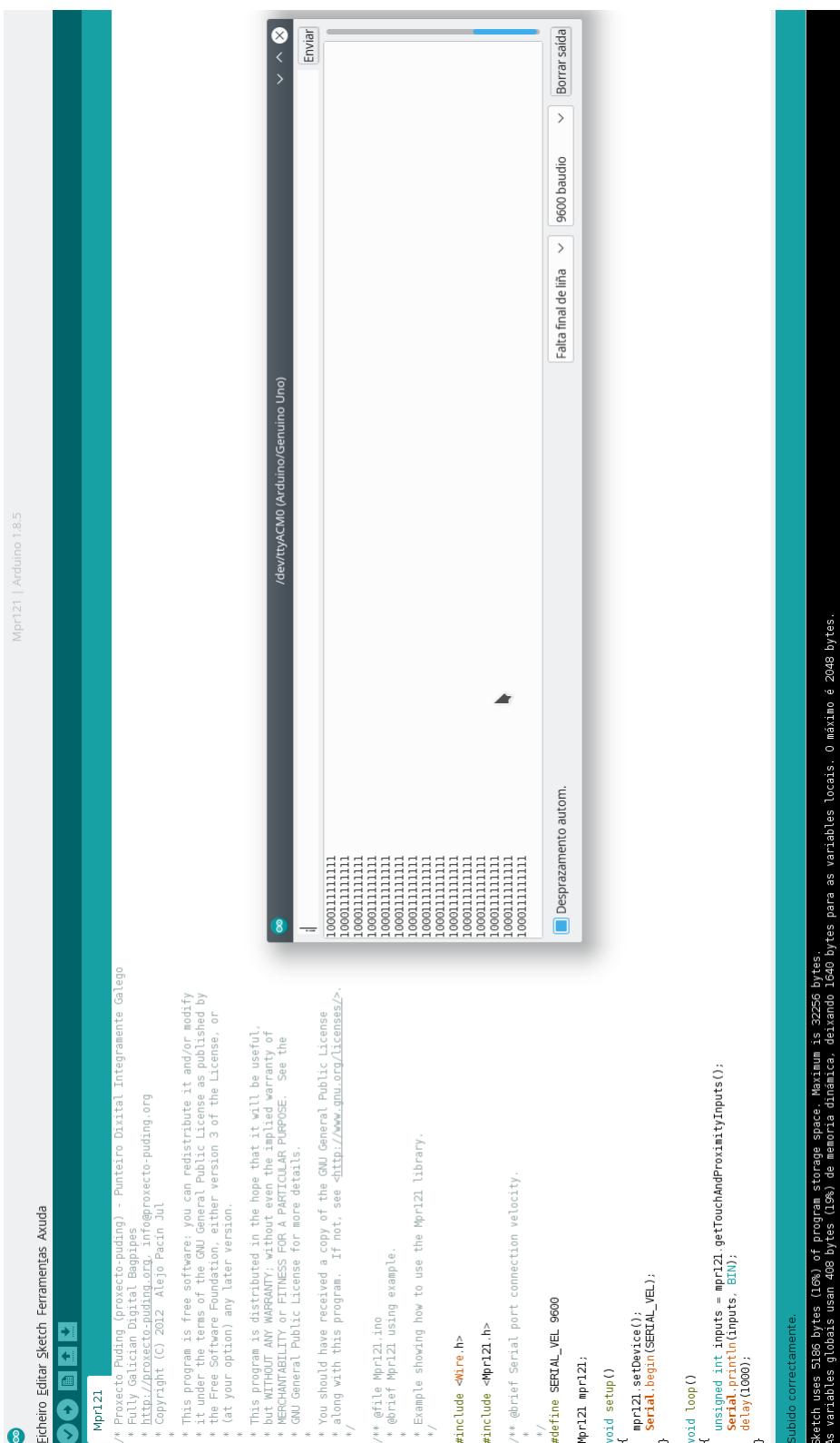


Figura 7.51: Resultado das probas de unidade dos sensores capacitivos.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Ficheiro Editar Sketch Ferramentas Axuda
- Sketch Name:** G1
- Code Content:**

```
/* Proxecto Puding (proxecto-puding) - Punteiro Dixital Integramente Galego
 * Fully Galician Digital Bagpipes
 * http://proxecto-puding.org, info@proxecto-puding.org
 * Copyright (C) 2012 Alejo Pacín Jul
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/** @file G1.ino
 * @brief G1 using example.
 *
 * Example showing how to use the G1 library.
 */

#include <Wire.h>

#include <G1.h>

/** @brief Serial port connection velocity.
 *
 */
#define SERIAL_VEL 9600

G1 g1;

void setup()
{
    g1.setDevice();
    Serial.begin(SERIAL_VEL);
}

void loop()
{
    char *directoryName = ".";
    char *fileName = "test.txt";
    boolean append = false;
    String data = "test";
    char *newFileName = "moved.txt";

    g1.printDeviceVersionInfo();
    g1.listDirectory(directoryName);
    g1.writeFile(fileName, append, data);
    g1.moveFile(fileName, newFileName);
    g1.removeFile(newFileName);
    delay(10000);
}
```

Figura 7.52: Ficheiro de proba do lector de tarxetas.

```

G1 | Arduino 1.8.5
Eixeiro Editar Sketch Ferramentas Axuda
G1
/*
 * Copyright (C) 2012 Alejo Pacin Jul
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>
 */

#include "G1.h"

/* @file G1.ino
 * @brief G1 using example.
 *
 * Example showing how to use the G1 library.
 */

#define SERIAL_9600

G1 g1;

void setup()
{
    g1.setDevice();
}

void loop()
{
    char *directoryName = ".";
    char *filename = "test.txt";
    boolean append = false;
    String data = "Text";
    char newFilename = "moved.txt";

    g1.printDeviceInfo();
    g1.listFiles(directoryName);
    g1.writeFile(filename, append, data);
    g1.moveFile(filename, newFilename);
    g1.removeFile(newFilename);
}

Archiving built core (caching) in /tmp/arduino_cache_20105/core_arduino_0f7246b8028463395b8c146161738a.a
Sketch uses 502 bytes (1%) of program storage space. Maximum is 32256 bytes.
As variables globais usan 417 bytes (20%) de memoria dinámica, deixando 1631 bytes para as variables locais. O máximoo é 2048 bytes.

```

Compilacion satisfactoria.

Figura 7.53: Ficheiro de proba do lector de tarxetas.

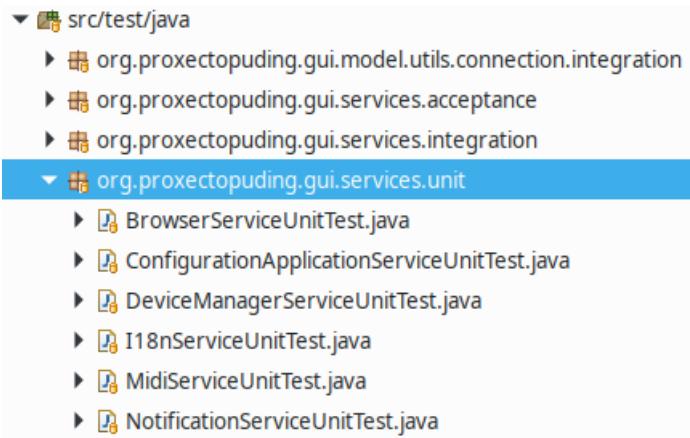


Figura 7.54: Tests unitarios.

```
@Test
public void findBagpipeDevices() {

    // Given
    doNothing().when(connectionManager).writeData(anyString(), eq(false));
    when(connectionManager.readData(false))
        .thenReturn(getDeviceAsJson(), (String) null);
    doNothing().when(deviceManager).addDevice(getDevice());
    when(deviceManager.getDevices()).thenReturn(getDevices());
    doNothing().when(connectionManager).disconnect();

    // When
    Set<BagpipeDevice> devices = deviceManagerService.findBagpipeDevices();

    // Then
    verify(connectionManager, times(2)).writeData(anyString(), eq(false));
    verify(connectionManager, times(1)).readData(false);
    verify(deviceManager, times(1)).addDevice(getDevice());
    verify(deviceManager, times(2)).getDevices();
    verify(connectionManager, times(1)).disconnect();
    assertEquals(getDevices(), devices);
}
```

Figura 7.55: Exemplo de test unitario con Mockito.

configuráse o comportamento do mock. Logo execútase a operación que se quere probar e finalmente compróbanse as poscondicións.

É de salientar que, aínda que máis propio de BDD, intentou empregarse linguaaxe *Gherkin* [72] tanto como foi posible para que os tests fóran o máis sinxelos de ler posible.

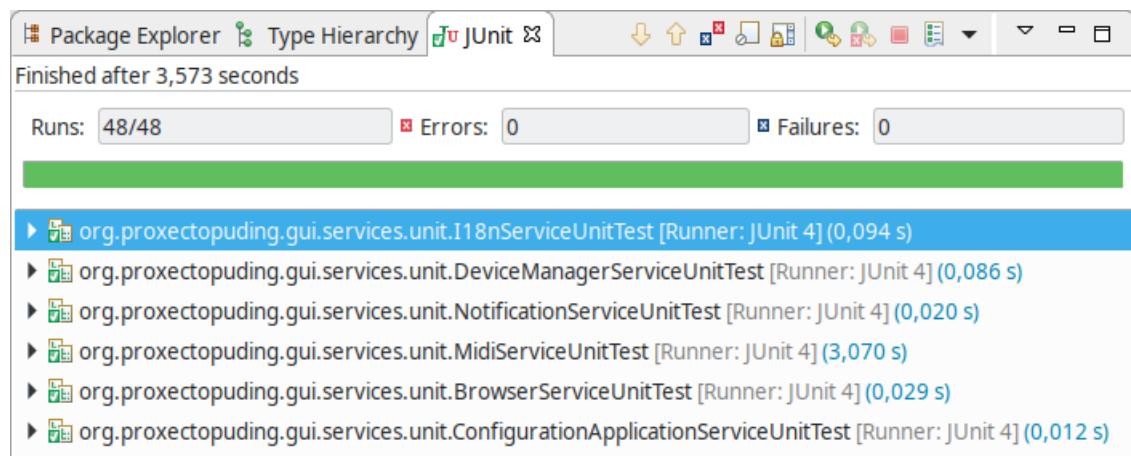


Figura 7.56: Resultados tests unitarios.

Executadas as probas, cun total de 48 tests unitarios, podemos comprobar que todas pasan correctamente (figura 7.56).

7.3.5. Integración e probas

Durante o desenvolvemento do prototipo software operacional creáronse baterías de tests de integración a nivel de servizo.

Creouse unha batería de tests por servizo (figura 7.57 e cubriuse alomenos o *happy path* de cada unha das súas sinaturas.

Vexamos un exemplo. Como se pode apreciar na figura 7.58, créanse as precondicións, logo execútase a operación que se quere probar e finalmente compróbanse as poscondicións.

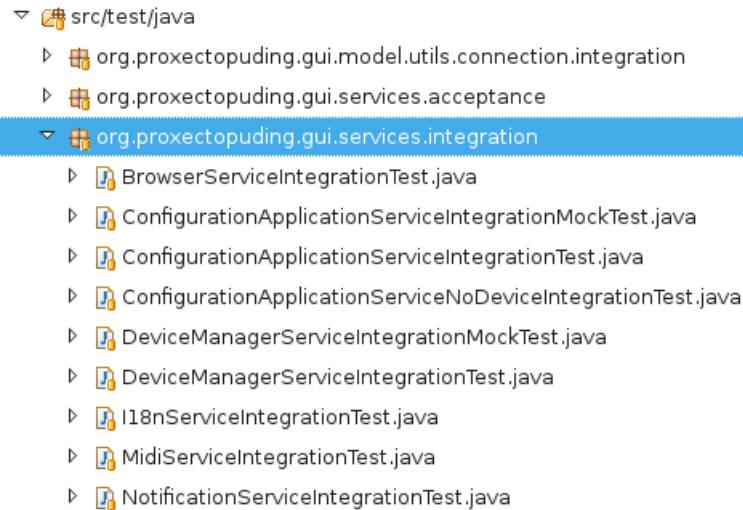


Figura 7.57: Tests de integración.

```
@Test
public void findBagpipeConfigurations() {
    // Given
    Set<BagpipeDevice> devices = deviceManagerService.findBagpipeDevices();
    assertTrue(devices.size() > 0);
    String expectedProductId = devices.iterator().next().getProductId();

    // When
    Set<BagpipeConfiguration> configurations =
        deviceManagerService.findBagpipeConfigurations(expectedProductId);

    // Then
    assertEquals(BagpipeConfigurationType.values().length,
                configurations.size());
    configurations.forEach(configuration ->
        assertEquals(expectedProductId, configuration.getProductId()));
}
```

Figura 7.58: Exemplo de test de integración.

É de salientar que, aínda que máis propio de BDD, intentou empregarse linguaaxe *Gherkin* [72] tanto como foi posible para que os tests fóran o más sinxelos de ler posible.

Aquí a aproximación consistiu en executar os tests tendo o hardware conectado ó pc executando a aplicación.

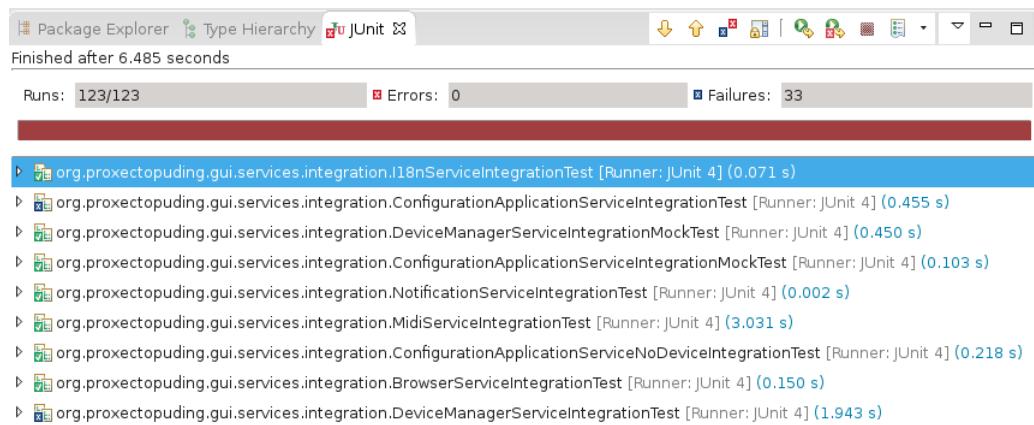


Figura 7.59: Resultados tests integracion.

Executadas as probas, cun total de 133 tests de integración, podemos comprobar que non todas pasan correctamente (figura 7.59).

Concretamente, se nos fixamos en ditos resultados, as que non pasan son as que precisan conectividade co dispositivo. Isto débese a unha incidencia que se deu durante o desenvolvemento e que xa comentamos anteriormente, que nos levou a ter que empregar unha versión simplificada do firmware do dispositivo. Dita incidencia detallarase na sección correspondente.

Para as suites de tests que fallan, para intentar paliar este problema, desenvolvéronse uns tests de integración alternativos (co sufijo Mock), no que se simula o intercambio de información en formato JSON co dispositivo, de maneira que alomenos poidamos verificar e validar a aplicación de configuración incluíndo a serialización e deserialización da información, tal e como se faría cun dispositivo real. É dicir, contamos cun dispositivo simulado en Java que emite e recibe

información en formato JSON, tal coma o identificador ou os distintos tipos de configuración.



Figura 7.60: Tests de integración da conectividade.

Así mesmo, para probar que a conectividade co dispositivo funciona e polo tanto a librería serie en Java cumpría co seu cometido (dado que coa primeira escollida tamén tivemos problemas), desenvolvéronse un par de tests extra de integración da conectividade e os seus sketches asociados (figura 7.60).

Na figura 7.61 podemos apreciar o segundo dos mesmos e o más complexo, que trata de xogar ó ping-pong cunha cadea de texto entre a aplicación e o dispositivo.

Tal e como se pode comprobar na figura 7.62, as probas pasaron correctamente.

Executadas as probas, cun total de 135 tests de integración, puidemos validar a aplicación de configuración e a conectividade co dispositivo hardware.

7.3.6. Probas de aceptación

Durante o desenvolvemento do prototipo software operacional creáronse baterías de tests de aceptación a nivel de servizo.

Creouse unha batería de tests por servizo (figura 7.57 e cubriuse alomenos o *happy path* de cada unha das súas sinaturas).

As probas de aceptación fixéronse en todo momento seguindo os principios de BDD, deseñando os tests en base á interacción do usuario coa aplicación e especificando os mesmos empregando linguaxe Gherkin.

```
/* PingPong.ino

void setup() {
    Serial.begin(9600);
}

void loop() {
    String data = Serial.readString();
    delay(1000);
    Serial.print(data);
}

*/
@Test
public void writeToAndReadFromDevice() {

    try {
        connectionManager.connect();
        String data = null;
        for (int i = 0; i < 10; i++) {
            LOGGER.log(Level.INFO, "Attempt: {0}", i + 1);
            connectionManager.writeData("ping", false);
            Thread.sleep(2000);
            data = connectionManager.readData(false);
            if (data != null) {
                break;
            }
        }
        assertNotNull(data);
    } catch (Exception e) {
        fail(e.getMessage());
    } finally {
        connectionManager.disconnect();
    }
}
```

Figura 7.61: Tests de integración da conectividade.

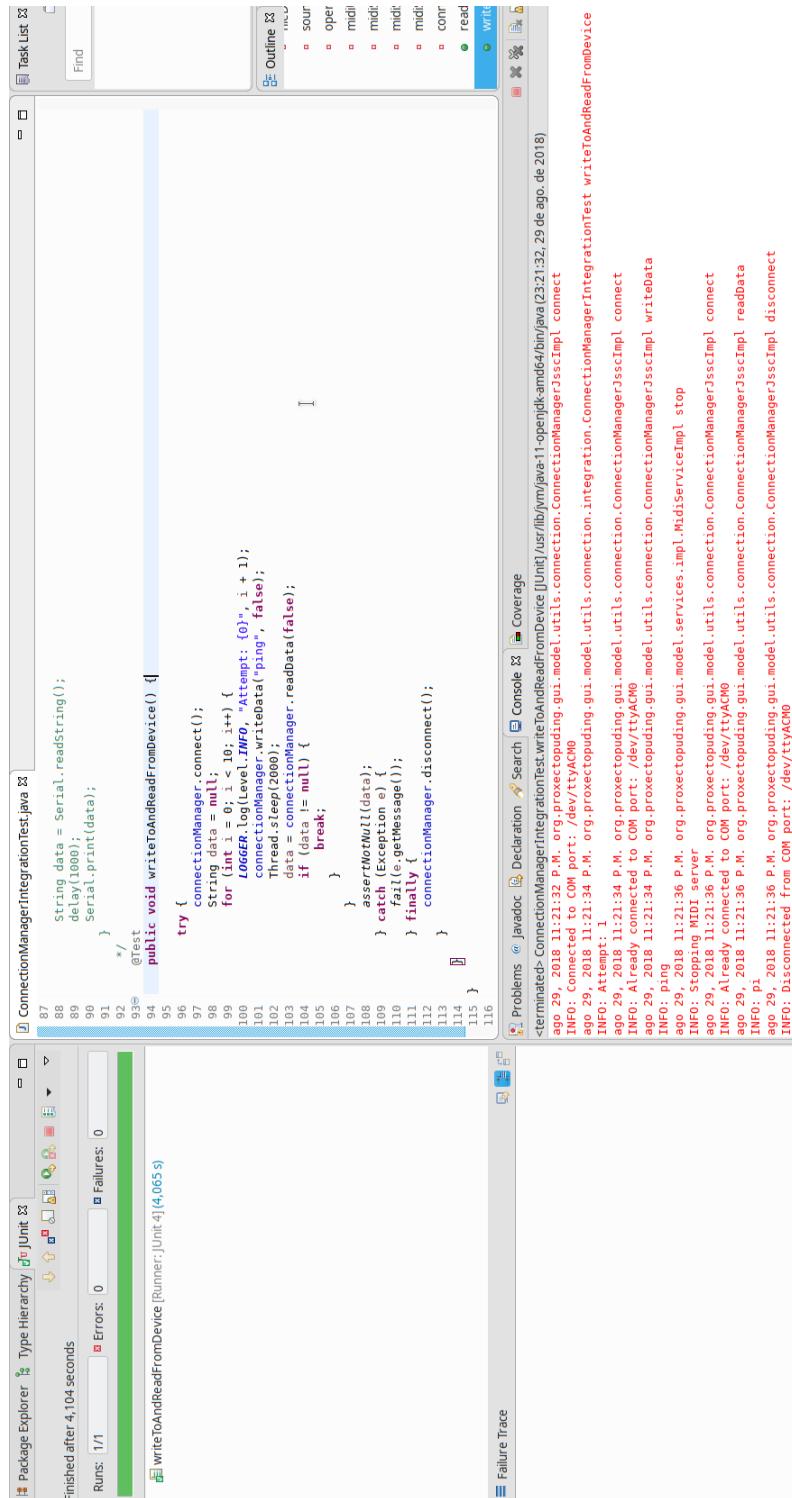


Figura 7.62: Resultados dos tests de integración da conectividade.



Figura 7.63: Tests de aceptación.

Vexamos un exemplo. Como se pode apreciar na figura 7.64, créanse as precondicións, logo execútase a operación que se quere probar e finalmente compróbanse as poscondicións.

Se nos fixamos na documentación da suite e do test, vemos como están especificados en linguaxe Gherkin, onde a suite é proba unha funcionalidade onde se poden dar varios escenarios, que é o que se describe e proba nos tests.

Desta maneira, os propios tests de aceptación poden servir de contrato entre partes, xa que os requisitos se poden especificar en base a funcionalidades con distintos escenarios e, se os tests de aceptación pasan, cúmprese co contrato asinado.

A aproximación dos tests de aceptación foi a mesma que a dos de integración, tendo o hardware conectado ó pc executando a aplicación.

Executadas as probas, cun total de 11 tests de aceptación, podemos comprobar que non todas pasan correctamente (figura 7.65).

Concretamente, se nos fixamos en ditos resultados, as que non pasan son, ó igual que no caso dos tests de integración, as que precisan conectividade co dispositivo, polo mesmo motivo.

Para as suites de tests que fallan, para intentar paliar este problema, desenvolvéronse uns tests de aceptación alternativos (co sufijo Mock), no que se simula

```
DeviceManagerServiceAcceptanceTest.java
```

```
47     private DeviceManager deviceManager = new DeviceManager();
48     private DeviceManagerService deviceManagerService =
49         new DeviceManagerServiceImpl(connectionManager, deviceManager);
50
51     /**
52      * Feature: user configures a device
53      */
54
55     /**
56      * Scenario: user selects a device
57      * Given devices have been found
58      * When user selects a device
59      * Then device's configurations are loaded
60      * And device figures out as selected
61      */
62     @Test
63     public void selectDevice() {
64
65         // Given
66         Set<BagpipeDevice> devices = deviceManagerService.findBagpipeDevices();
67         assertTrue(devices.size() > 0);
68         String expectedProductId = devices.iterator().next().getProductId();
69
69         // When
70         deviceManagerService.setSelectedBagpipeDevice(expectedProductId);
71         Set<BagpipeConfiguration> configurations =
72             deviceManagerService.findBagpipeConfigurations(expectedProductId);
73
74         // Then
75         BagpipeDevice device = deviceManagerService.getSelectedBagpipeDevice();
76         assertEquals(expectedProductId, device.getProductId());
77         assertEquals(BagpipeConfigurationType.values().length,
78             configurations.size());
79         configurations.forEach(configuration ->
80             assertEquals(expectedProductId, configuration.getProductId()));
81     }
82 }
```

Figura 7.64: Exemplo de test de aceptación.

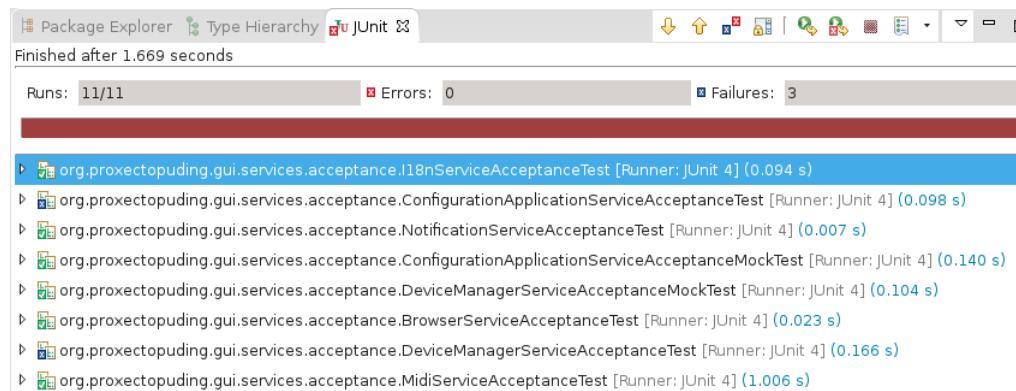


Figura 7.65: Resultados tests aceptación.

o intercambio de información en formato JSON co dispositivo, de maneira que alomenos poidamos verificar e validar a aplicación de configuración incluíndo a serialización e deserialización da información, tal e como se faría cun dispositivo real. É dicir, contamos cun dispositivo simulado en Java que emite e recibe información en formato JSON, tal coma o identificador ou os distintos tipos de configuración.

Ademáis de todo o anterior, examinando a interacción do usuario coa interface para documentar os casos de proba, démonos conta de que no caso de que a busca dos dispositivos fallase, o usuario non tiña maneira de reintentar dita busca máis que pechando e volvendo abrir a aplicación, o que desde o punto de vista de UX non ten sentido, polo que se procedeu a incluír un botón de busca na pantalla de inicio (figura 7.66).

Executadas as probas, cun total de 11 tests de aceptación, puidemos validar os casos de uso da aplicación.

7.3.7. Documentación

Todo proxecto comunitario que se precie, debe contar cunha boa documentación se quere sobrevivir no tempo e crear unha boa comunidade detrás do mesmo.

Ademáis, contar cunha boa documentación dos servizos axuda á hora de determinar o comportamento desexado e escribir as probas.

Por iso, todo o código do presente proxecto foi perfectamente documentado dende un primeiro momento.

Como contamos con parte de código en C e parte en Java e en aras de gardar a máxima similitude entre a documentación das dúas APIs do proxecto, optamos por empregar un software de documentación que soportase ambas linguaxes. Neste caso, o escollido foi *Doxygen* [73] en detrimento doutros software populares como *Javadoc*.

Ambas APIs están dispoñibles para a súa consulta a través dos enderezos:

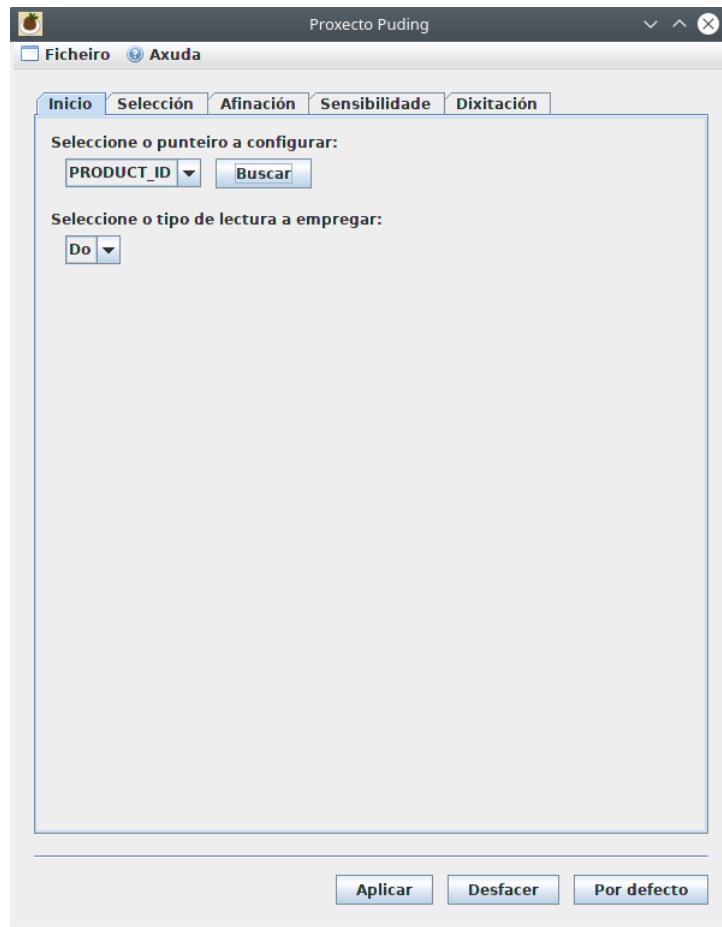


Figura 7.66: Botón de busca.

<http://bagpipeapi.proxecto-puding.org>

<http://confappapi.proxecto-puding.org>

A documentación xerada conta cunha calidade tanto visual coma de contidos moi alta, tal e como pode apreciarse na figura 7.67.

Tampouco nos esquecemos dos manuais de consulta, tanto técnica como de usuario, que se poden consultar a través dos enderezos:

<http://usermanual.proxecto-puding.org>

<http://technicalmanual.proxecto-puding.org>

E para telos sempre á man, incluíuse unha referencia a cada un deles dentro do menú de axuda da aplicación de configuración (figura 7.68).

Ademáis, ó tela disponible en liña e dentro do repositorio, sempre está aliñada á última versión disponible.

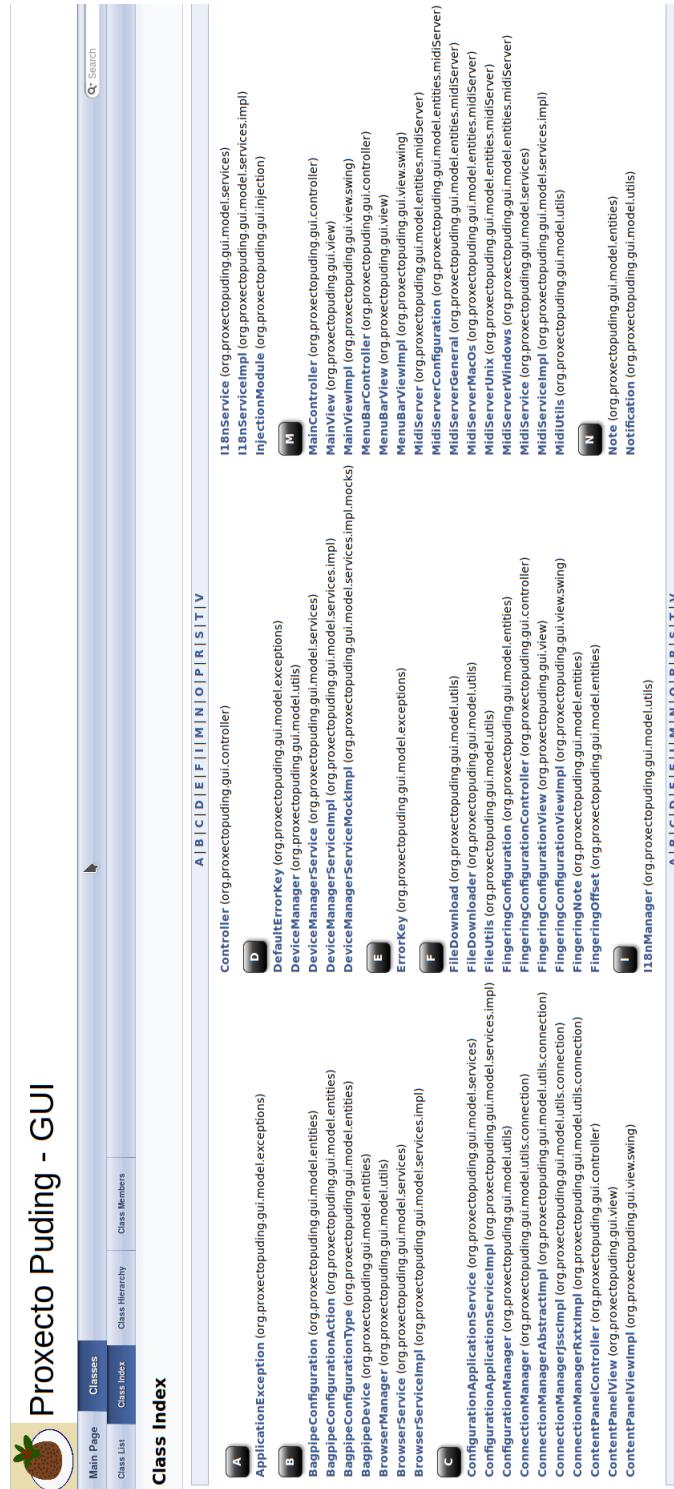


Figura 7.67: Documentación da API da aplicación de configuración.

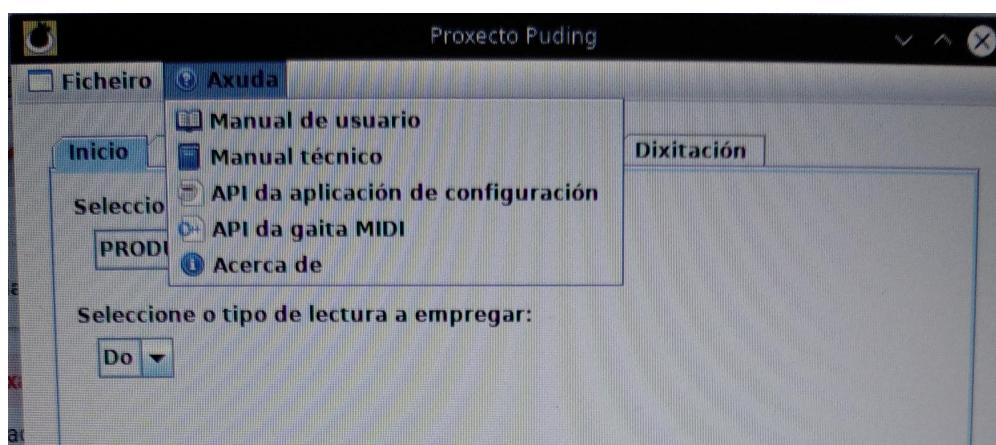


Figura 7.68: Menú de axuda da aplicación.

Capítulo 8

Conclusións

Índice xeral

8.1. Incidencias durante o desenvolvemento do proxecto	215
8.1.1. Planificación	215
8.1.2. Deseño do sistema	218
8.1.3. Implementación	221
8.2. Conclusións finais	227
8.3. Traballo futuro	229

NESTE capítulo exporanse as conclusións do proxecto: desde as múltiples incidencias, as maneiras de resolvelas e as leccións que se aprenderon delas, ata as oportunidades de mellora ou futuras liñas de traballo.

8.1. Incidencias durante o desenvolvemento do proxecto

8.1.1. Planificación

8.1.1.1. Xestión do proxecto

Como se comentou no capítulo 3, nun primeiro momento escolleuse *OpenProj* coma xestor de proxectos, procedendo a meter nel toda a planificación. Todo ía

ben, ata que un día fallou sen explicación aparente.

Por motivos de dispoñibilidade de tempo do proxectando (estudos, traballo, etc.) decidiuse empregar un calendario de media xornada para o proxecto. Pois resulta que *OpenProj*, alomenos na súa última versión liberada (v1.4-2, do 02/10/2008), presenta problemas á hora de manexar calendarios que non son o calendario por defecto (xornada completa).

Concretamente, o erro deuse un día calquera ó abrir o ficheiro do proxecto cando, sen explicación aparante e sen ter cambiado nada, pasou de empregar un calendario de media xornada a un de xornada completa composto por dous de media xornada. É dicir, o que fixo foi meter nun mesmo día dúas medias xornadas e, polo tanto, reducir o intervalo estimado de datas á metade. A partires dese momento, aínda sen ter salvado os cambios, o erro reproducíase sempre.

Dito problema non é salvable polo usuario de ningunha maneira sen distorsionar os datos da planificación, polo que é preciso corrixilo no código da aplicación. Investigando pola rede, parece ser que o erro estaba xa corrixido na seguinte versión, pero dita versión non era libre e non tiña visos de chegar a selo en moito tempo (*OpenProj* tamén cunha versión comercial).

Por dito motivo, tocou tirar coa planificación feita e volver comezar. Do que quedaba na lista e guiándose polas recomendacións, seleccionouse *Planner*.

Planner é un xestor bastante sinxelo e rápido. Perdíase algo de funcionalidade fronte a *OpenProj*, pero dado o atraso acumulado e que as pequenas carencias que se lle vían eran mitigables, optouse por el igualmente.

Houbo que refacer toda a planificación de cero, dado que os formatos non eran compatibles e a xestión da información é totalmente diferente.

Todo perfecto, ata que tocou asignar recursos materiais ás tarefas. Seleccionada a primeira tarefa, que xa tiña asignado un recurso humano, asígnaselle un recurso material e de repente a duración da tarefa redúcese á metade. É dicir, os

recursos materiais consomen esforzo. En resumo, que os recursos materiais traballan. Para entenderse, se nunha tarefa se emprega, por exemplo, un paquete de folios, ese paquete traballa. Inadmisible.

Isto pode ter sentido para recursos máquina, onde pode ser necesario contabilizar as horas de traballo da maquinaria, pero non para recursos materiais en xeral e sen posibilidade de decisión ó respecto.

Notificáronse este e outros erros ós desenvolvedores, que procederon a marcar los todos coma duplicados, cando moitos deles non o eran e o resto tiñan un título moito máis descriptivo que os que xa había (polo que non se atoparon nunha busca previa). Ademais, agrupáronos todos nun único erro “caixón de xastre” sen relación. Por este último motivo e logo de que máis profesionais lles insistiran e non obtiveramos resposta algúnhā, desbotouse tamén o *Planner*.

Na lista restaban por probar *GanttProject* e *LibrePlan*. Analizando un pouco máis en profundidade o *GanttProject* chegouse á conclusión de que era demasiado sinxelo (por non dicir incompleto) polo que se desbotou sen máis.

Chegados a este punto, só quedaba *LibrePlan*: servidor web, base de datos, interface web, multitud de opcións a priori complexas, xestión de información totalmente distinta a todo o visto anteriormente, alta curva de aprendizaxe e un longo etcétera.

Volta outra vez a facer a planificación de cero. Ardua tarefa dada a lentitude da interface web. Tamén se detectou algúñ erro leve (corrixido en versións posteriores) e outro que parecía grave e ó final non era máis que o descoñecemento moi específico que *LibrePlan* marca por defecto e que se atopa bastante agochado.

Con *LibrePlan* deuse tamén unha incidencia grave, froito dun erro do entorno de escritorio do sistema operativo empregado. Nunha das múltiples actualizacións da aplicación, non se reconfigurou a base de datos por quedar oculta a fiestra de xestión da configuración, o que ocasionou que a aplicación se actualizase pero a base de datos non, facendo romper a primeira e corrompendo a segunda.

Gracias ó traballo da xente detrás de *LibrePlan* puido recuperarse a base de datos e con ela a planificación no mesmo estado que antes de corromperse.

8.1.2. Deseño do sistema

8.1.2.1. Prototipo 3

8.1.2.1.1. Prototipo hardware

8.1.2.1.1.1. Receptor XBee

Como se comentou no capítulo 6, empregar un *XBee Explorer USB* [48] (figura 6.11) como base do receptor XBee plantexaba un problema.



Figura 8.1: XBee Explorer USB.

Empregando esta placa, o dispositivo sería recoñecido no equipo coma un dispositivo USB, non coma un dispositivo MIDI, que é o que precisamos en última instancia. Por este motivo, habería que crear un controlador software para este dispositivo que fixese a conversión de USB a MIDI, para que puidese ser recoñecido polo sintetizador. Isto implicaría un retardo considerable (que seguramente dese ó traste co requisito de tempo real) e ademais, implicaría desenvolver un controlador distinto por cada familia de sistemas operativos. Como se pode intuir a simple vista, esta opción non era viable.

Así que houbo que investigar un pouco para ver se había maneira de facer dita conversión por hardware, de tal maneira que ó enchufar o receptor vía USB fose recoñecido coma dispositivo MIDI, aforrando unha capa intermedia, un retardo importante e un esforzo máis ca considerable.

Mirando o esquema do *XBee Explorer USB* pódese ver que leva un chip *FT232RL* (que é o que se encarga de facer a conversión de FTDI a USB) que, segundo pon aquí [74] e aquí [75], non é posible reprogramar para facer unha conversión a MIDI.

Segundo comentan tamén na segunda referencia, ese mesmo chip era o que se empregaba ata a versión *Duemilanove* [76] da placa estándar *Arduino*, polo que tampouco era posible facelo con unha destas placas ata dita versión.

A partires desa versión, cambiouse dito chip polos da familia *Atmel Mega XuY* (8u2, 16u2, 32u4). Comezouse polo *Arduino Uno* (8u2), que na súa terceira revisión o cambiou polo 16u2 [77]. E actualmente, a última placa que acaban de sacar, a *Arduino Leonardo* [78] leva o 32u4.

Dito chip está programado para funcionar exactamente igual que o *FT232RL*, pero coa salvedade de que se pode reprogramar totalmente, polo que se pode emplegar calquera placa *Arduino* con dito chip para facer a conversión.

Aínda que no momento de escribir estas liñas xa está dispoñible unha placa nova [78] que quizáis nos resultase más sinxela de reprogramar, pois xa conta con USB nativo (sen necesidade de facer conversión serie-USB, polo que o novo firmware sería más sinxelo), no momento de realizar o deseño hardware a última placa era a *Arduino Uno r3*, que é un pouco máis completa, pero non conta con USB nativo.

Sen embargo, se nos fixamos outra vez na referencia [75], vemos que xa a partires da revisión 2 (8u2) existe un firmware que fai este cometido, polo que se aforra moito tempo e esforzo. O esquema de funcionamento é o da figura 8.2.

Ademais, polo que se pode ler ó final do artigo

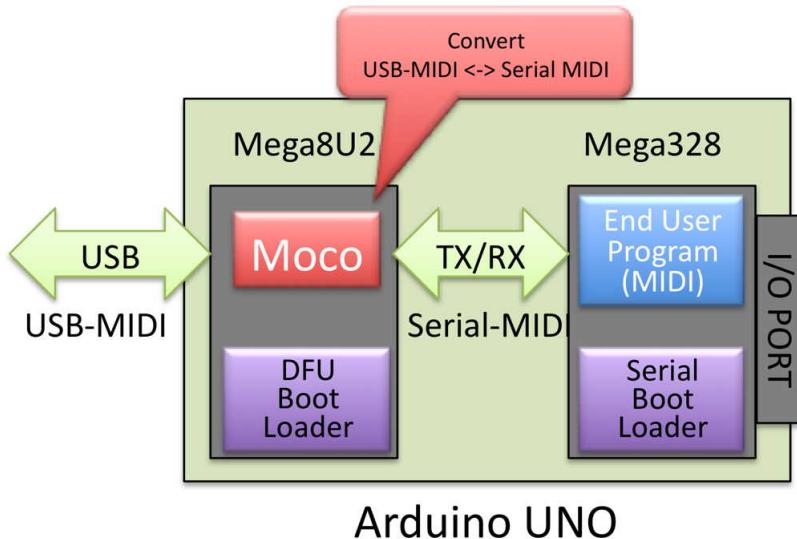


Figura 8.2: Moco.

Moco firmware can use as USB-MIDI <-> USB-Serial bridge with a 8u2 or 32u4 board, like Adafruit's 32u4 breakout board.

vemos que dito firmware tamén funciona co chip 16u2 (que simplemente é un 8u2 con máis memoria) e co 32u4 (que é o más recente), polo que actualmente funciona con tódalas placas *Arduino* lanzadas dende o *Arduino Uno r2*.

Polo que finalmente se optou por empregar como base do receptor un *Arduino Uno* con firmware *Moco*.

8.1.2.1.1.2. Fritzing

Como se comentou no capítulo 6, a instalación de *Fritzing* foi problemática. Instalada a versión disponible no repositorio da distribución en uso, comprobouse que a base de datos non contaba cunha gran cantidade de pezas. Procedese a comprobar a versión instalada (0.6.3) e cotexala coa última disponible nese momento (0.7.7b). Detectado o problema, intentouse compilar a versión disponible na web, pero non foi posible por mor de dependencias non localizables. A solución ó problema da escaseza de pezas da base de datos da versión do repositorio pasou

por hackear a mesma, substituíndo a pola da versión máis recente (que contaba cun número de pezas moito más elevado).

Tamén se deron incidencias coas pezas a usar. A única reseñable por ter dado algo máis de traballo foi a que se deu co módulo de almacenamento, o *MicroDrive G1*.

Como se comentou no capítulo 6, non estaba na base de datos do software por ser dun fabricante menos coñecido. Afortunadamente, estaba dispoñible no apartado de contribucións do repositorio da aplicación [55]. Unha vez descargada, intentouse incorporala ó proxecto, pero non foi posible, pois tiña algúin tipo de erro (e por iso non estaba incluída na base de datos). Investigando a fondo o ficheiro do proxecto (empregando un tutorial [79] como guía), de formato *fzp* (que non deixa de ser un XML) atopouse un erro na ruta ás diferentes vistas da peza (en formato *svg*), pois non coincidían totalmente os nomes dos ficheiros. Solventado o problema (hackeando o XML a man), procedeuse á súa incorporación ó proxecto. Así mesmo, notificouse o erro ós desenvolvedores da aplicación, contribuíndo a versión corrixida ó repositorio do proxecto [80].

8.1.3. Implementación

8.1.3.1. Ensamblado e codificación

Durante a fase de ensamblado e codificación, unha vez tiñamos toda a codificación da parte hardware lista para probar, xurdiu un problema co que non contabamos en absoluto: a falta de memoria RAM da placa para executar o firmware programado.

Todo ía ben e xa tiñamos codificado e compilando os tres módulos dos periféricos, e o software de integración de todos eles e que conforma o simulador do punteiro en si, pero cando fomos compilar todo o conxunto, previo á súa carga na placa Arduino Uno, o IDE díxonos que a compilación estaba ben, pero que o tamaño do executable excedía o máximo da memoria RAM nun 9 % e propúñanos intentar reducir o número de variables globais para intentar mitigalo (figura 8.3).

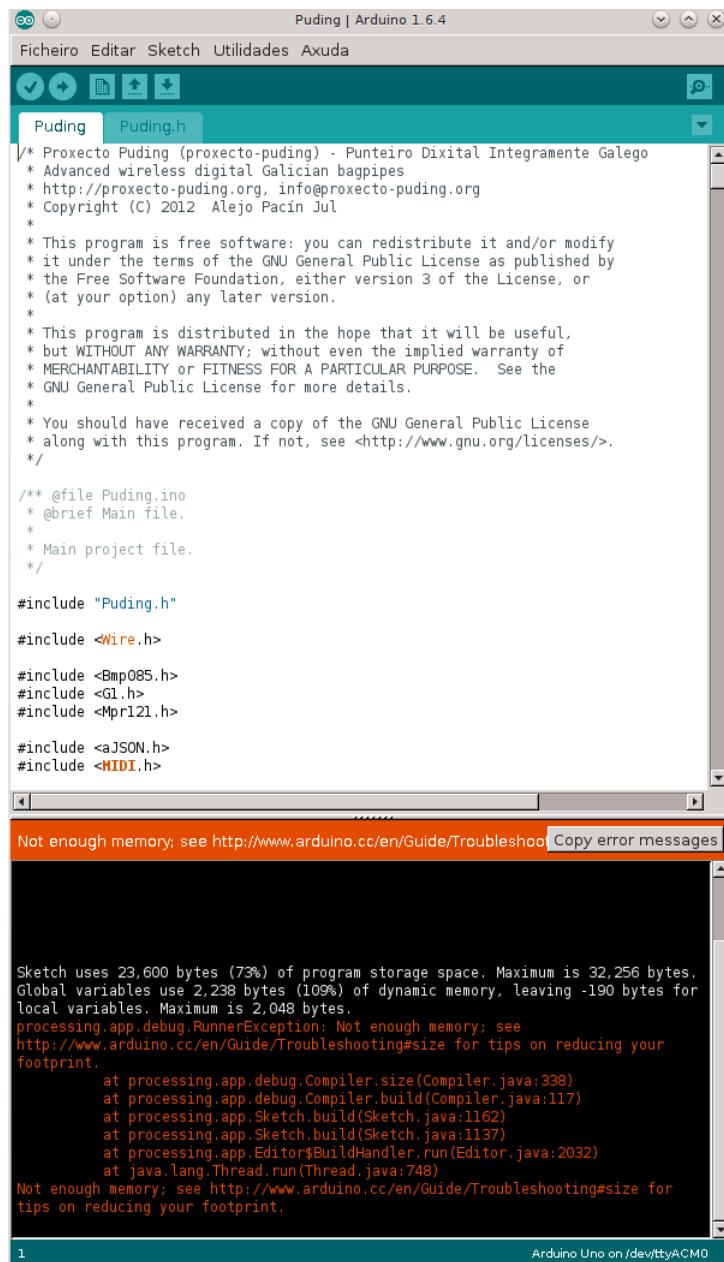


Figura 8.3: Consumo de memoria.

A primeira aproximación foi avaliar as variables globais da aplicación para ver o número delas, os seus tipos e se eran susceptibles de eliminación.

E viuse que eran moi pouquiñas, que os tipos eran básicos e que non eran eliminables. Salvo un par de delas, que son concretamente as matrices constantes que conteñen as relacións predefinidas entre dixitación e desprazamento en semitonos desde a nota base. As coñecidas como dixitación aberta e pechada.

O primeiro que nos chamou a atención foi que, sendo constantes, se almacenaseen en RAM e non en ROM. Buscando a causa, a explicación é que a arquitectura emplegada por Arduino, coas memorias separadas, fai que este tipo de variables constantes vaian parar á RAM.

Vendo que non era posible que acabasen na memoria reservada a datos fixos, analizáronse os valores e vendo que nos chegaban 2 bytes para representalos, cambiáronse de tipo int a short, pero sorprendentemente a nivel compilación non se gañou nada.

O último que probamos foi a movelas como variables locais á única función onde se empregaban, gañando temporalmente un 6 % de espacio de memoria, pero todavía insuficiente.

Cómpre explicar que estas dúas matrices se empregan para cargar noutra matriz de dixitacións globais todas aquelas dixitacións que o usuario queira empregar, puidendo ser aberta, pechada ou personalizada a vontade, polo que hai que ter en conta que a memoria necesaria pode subir ata outro 10 % só tendo en conta esta matriz global. E aínda teríamos que deixar marxe para o resto de variables locais que, se facemos unha similitude con outros sistemas similares, podemos estimar nun 20 % extra. Polo que de maneira estimada e aproximada, estamos nun -36 % de memoria RAM dispoñible.

Compilando o firmware coa última versión do IDE de Arduino instalada no equipo de implantación, consegui reducirse o consumo de memoria nun 3 % (senón por melloras do compilador, probablemente polo cambio de int a short

que poida que si aplique ben nesta nova versión), pero seguía sendo insuficiente. No mellor dos casos, estamos falando dun -28 % de memoria dispoñible.

Aclarar neste punto que, tanto a placa Arduino Uno como a Arduino Fio, contan exactamente coa mesma arquitectura e cantidade de memoria; no caso da RAM, 2KB.

Visto que non había maneira de salvar a restricIÓN de memoria de maneira programática e avaliando as alternativas definidas ó inicio da iteración, decidiuse que a mellor opción era recortar módulos para alomenos ter algo funcional.

Feitas contas e postos a escoller, a reproducción primaba sobre a configuración, polo que se prescindiu do lector de tarxetas, en detrimento tamén da aplicación de configuración.

E posto que tampouco estaba claro como ía evolucionar o consumo de memoria das variables locais no tempo e que o IDE non proporciona dita información de ningunha maneira, por precaución, decidiuse prescindir tamén do sensor de presión e incorporalo posteriormente se había posibilidade.

Polo que se prodeceu ó desenvolvemento dunha versión simplificada paralela que, unha vez rematada, arroxou un consumo de memoria mínimo do 72 % (figura 8.4), que nos deixa exactamente no mínimo de RAM libre que teorizamos que nos faría falta para as variables globais non inicializadas previamente e as variables locais.

8.1.3.2. Integración e probas

Foito do problema comentado no punto inmediantemente anterior, dado que se anularon o sensor de presión e o lector de tarxetas, non foi posible probar e integrar a aplicación de configuración co dispositivo hardware, porque a fin de contas, non había hardware contra o que probar dita funcionalidade.

Para mitigalo, tal e como se comentou durante a fase de probas, o que se fixo



The screenshot shows the Arduino IDE interface with the title bar "SimplePuding | Arduino 1.8.5". The menu bar includes "Ficheiro", "Editar", "Sketch", "Ferramentas", and "Axuda". The toolbar has icons for file operations like Open, Save, and Print. The main window displays the code for "SimplePuding.ino". The code includes a header with copyright information for Projeto Puding, followed by comments about the GNU General Public License. It then defines a class "SimplePuding" with includes for "SimplePuding.h", "Wire.h", "Mpr121.h", and "MIDI.h". The class definition starts with an addtogroup block for "PudingSensors". A variable "Mpr121 mpr121;" is declared. A message "Compilación satisfactoria." is shown in a green bar at the bottom. The serial monitor at the bottom shows the output of the compilation process, including memory usage details.

```
/* Projeto Puding (projeto-puding) - Punteiro Dixital Integramente Galego
 * Advanced wireless digital Galician bagpipes
 * http://projeto-puding.org, info@projeto-puding.org
 * Copyright (C) 2012 Alejo Facin Jul
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
/** @file SimplePuding.ino
 * @brief Main file.
 *
 * Main project file.
 */
#include "SimplePuding.h"
#include <Wire.h>
#include <Mpr121.h>
#include <MIDI.h>
/** @addtogroup PudingSensors
 *
 * Bagpipes sensors and components.
 * @{
 */
Mpr121 mpr121;
/** @}
 */
Compilación satisfactoria.

Archiving built core (caching) in: /tmp/arduino_cache_435449/core/core_arduino_avr_uno_f07246b89284c63395b8c146a161738a.a
Sketch uses 5996 bytes (18%) of program storage space. Maximum is 32256 bytes.
As variables globais usan 1490 bytes (72%) de memoria dinamica, deixando 558 bytes para as variables locais. O máximo é 2048 bytes.

Arduino/Genuino Uno sobre /dev/ttyACM0
```

Figura 8.4: Consumo de memoria da versión simplificada.

foi crear un simulador software o máis fiel posible ó dispositivo hardware a nivel de comportamento, de maneira que pudieramos validar tanto o fluxo de datos coma o seu formato.

8.1.3.3. Implantación

Por se os continuos problemas cos que nos atopamos durante o desenvolvemento do proxecto non fosen suficientes, a día 16 de agosto de 2018 o equipo co que se estaba a desenvolver o proxecto derramouse sen previo aviso, dificultando moito a finalización do mesmo, pois estamos a falar de tres semanas vista ó límite de depósito da memoria, sen tempo case para maniobrar.

Intentando tomar vantaxe do problema e logo de mercar un equipo novo de urxencia e preparalo para o seu uso, decidiuse empregalo tamén como banco de probas para unha implantación limpia.

Durante todo o desenvolvemento do proxecto, logo dos múltiples problemas ocasionados por pequenas actualizacións do entorno de traballo, decidiuse mantelo estable durante a execución do mesmo, facendo que en última instancia contaramos cun sistema operativo, unha máquina virtual de Java e uns entornos de desenvolvemento totalmente desfasados, pero estables.

De aí que, ó facer unha nova implantación de cero e aproveitando para probar coas últimas versións de todo o anterior, moitas das cales ían xa por varias versións maiores posteriores, se presentasen problemas novos.

O proxecto foi incialmente pensado para OpenJDK 6 e posteriormente actualizado a OpenJDK 8 para aproveitar as novas funcionalidades de Java en aras de simplificar o desenvolvemento, pero a versión actual da instalada no equipo é a 11.

A priori non deu problemas, ata que probamos a executar os tests do servizo de navegación web, que non funcionaban. Buscando a causa pola rede, semella que OpenJDK en versións posteriores á 8, rompeu varios enlaces simbólicos e outras partes relacionadas que afectan á carga das librerías de accesibilidade e que fai que a librería nativa do JDK que se emprega habitualmente para interactuar

co escritorio independentemente do sistema operativo rompa en sistemas UNIX, polo que non permitía abrir o navegador web.

Como solución de emerxencia mentres non arranxan a incidencia, que xa está notificada, engadiuse unha implementación alternativa en caso de erro facendo uso dunha librería equivalente da Apache Foundation.

8.2. Conclusións finais

Un profesor desta facultade dixo unha vez que os enxeñeiros somos como navallas suízas, que valemos un pouco para todo aínda que non fagamos nada á perfección, pero que, a pouco que nos esforzamos, brillamos como ferramentas especializadas.

A idea académica subxacente detrás deste proxecto foi a de aplicar un pouquiño de cada unha das ramas da formación xeral adquirida durante a carreira que nos converteu nas navallas suízas que somos hoxe e facelas brillar no seu conxunto na medida do posible.

Planificación de proxectos, metodoloxías de programación, análise de requisitos e viabilidade, deseño hardware e software, deseño de interfaces e UX, deseño de probas, programación de alto e baixo nivel, aplicación de patróns, tratamiento de datos en tempo real, redes e protocolos de comunicación, arquitecturas cliente-servidor, electrónica, etc., son só algunas das materias que se tocaron durante a elaboración do presente proxecto, en moitas das cales só se posuían unhas nocións xerais.

O proxecto tratou de realizar de maneira teórico-práctica un controlador MIDI que simulase o máis fielmente posible unha gaita galega cumplindo cunha serie de requisitos:

- Que non empregase fíos.
 - Que reproducise son en tempo real.
 - E que empregase exclusivamente hardware e software libre.
-

Destes obxectivos, actualmente estamos en condicións de cumplir o primeiro, cumprimos parcialmente o segundo e cumprimos escrupulosamente o terceiro.

Tamén se prantexaron obxectivos secundarios pero non por iso menos importantes:

- Realizar un estudio de viabilidade que amosara o que realmente demanda o mercado.
- Estudar a fondo as tecnoloxías implicadas no proxecto e determinar se son aplicables ó mesmo.
- E demostrar que o uso de hardware/software libre é viable para este tipo de proxectos.

Estes obxectivos cumprímos todos, aínda que o terceiro podería ser matizable.

Debido á extensión do proxecto resulta un pouco complicado ás veces conservar unha visión global do mesmo, polo que a continuación aportamos unhas cifras xerais que sirvan para valoralo de maneira conxunta e cuantitativa.

Código	18000 liñas
Tempo	966 horas
Documentación	400 páxinas

Cadro 8.1: Resumo do proxecto.

Facendo un pouco de retrospectiva, revisitando os obxectivos do proxecto, avaliando o esforzo invertido, a extensión da memoria e os resultados obtidos, quizáis poidamos calificalo como demasiado ambicioso para un Proxecto Fin de Carreira.

Pero a sabedoría popular di que sarna con gusto non pica e consideramos que os resultados obtidos sentan unha boa base coa vista posta nun proxecto de longa duración que, tanto podería acometerse por futuros proxectandos ou doutorandos, como pola comunidade que se poida formar arredor do mesmo.

8.3. Traballo futuro

Sendo un proxecto que toca tantas disciplinas, resulta relativamente sinxelo atopar liñas de traballo futuro polas que poder evolucionalo.

Como máis prioritaria, facer funcionar o lector de tarxetas, sexa empregando este modelo ou outra alternativa máis moderna e axeitada.

A continuación, substituír a placa Arduino Uno/Fio actual por outra da mesma familia que teña unha cantidade de memoria que nos permita executar a versión completa do firmware. Preferentemente con soporte para XBee e integrable no punteiro.

Como seguinte punto a abordar, substituír o hardware que está discontinuado por versións más actuais: o sensor de presión, os sensores capacitivos, o lector de tarxetas ou a placa do receptor están actualmente marcados como discontinuados polos fabricantes, que optaron por renovar os modelos, aínda que a día de hoxe podemos atopar áinda unidades á venda dalgún deles.

Actualizar a interface de Swing a JavaFX debería ser a seguinte prioridade. Cando se comezou co proxecto JavaFX nin sequera existía como primeira versión estable, pero co paso dos anos converteuse no substituto lóxico de Swing, que entre outros, presenta inconvenientes graves como son a falta de unha linguaxe descriptiva estructurada de definición de interfaces e aplicación de estilos, mellora substancial que si aporta JavaFX mediante XML e CSS.

E a partir de aí, as posibilidades son infinitas, polo que deixamos a criterio do lector, futuros proxectandos ou doutorandos e á comunidade a decisión de por onde abordar o futuro do proxecto a longo prazo.

Apéndices

Apéndice A

Planificación completa

NESTE apéndice exponse a planificación completa do proxecto que, dada a súa extensión, non é posible incluír intercalada no correspondente apartado da memoria.

A.1. Planificación inicial

A continuación exponse a planificación inicial con dependencias entre tarefas incluídas e en orde cronolóxica (figuras A.1 a A.8).

Na seguinte ligazón dispone da planificación inicial completa en alta resolución:

<http://initialplanning.proyecto-puding.org>

A.2. Planificación final

A continuación exponse a planificación final comparando as horas estimadas contra as horas reais e seguindo a mesma orde que na planificación inicial (figuras A.9 e A.9).

Na seguinte ligazón dispone da planificación final completa en alta resolución:

A.2. Planificación final

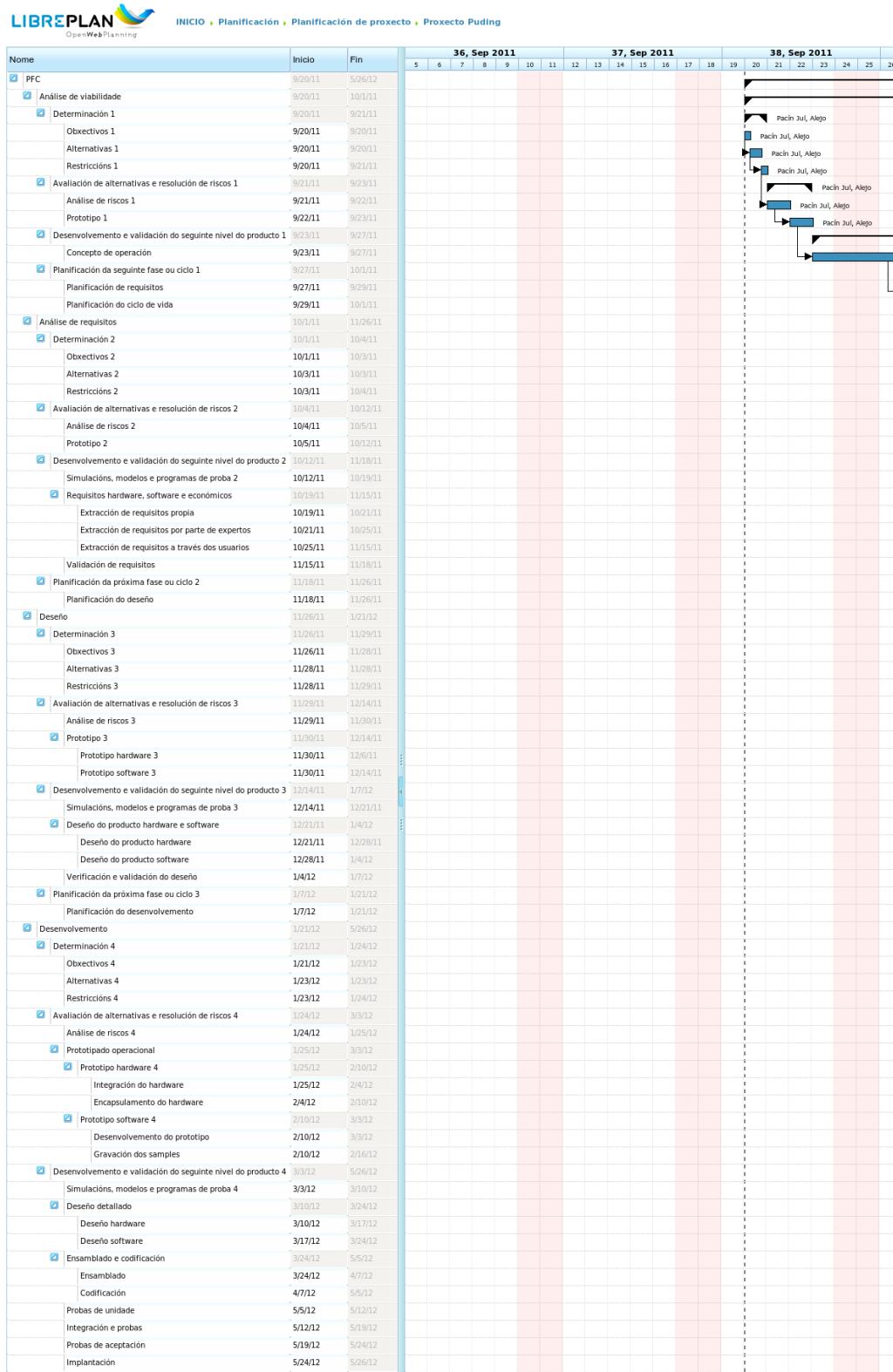


Figura A.1: Planificación inicial (p. 1).

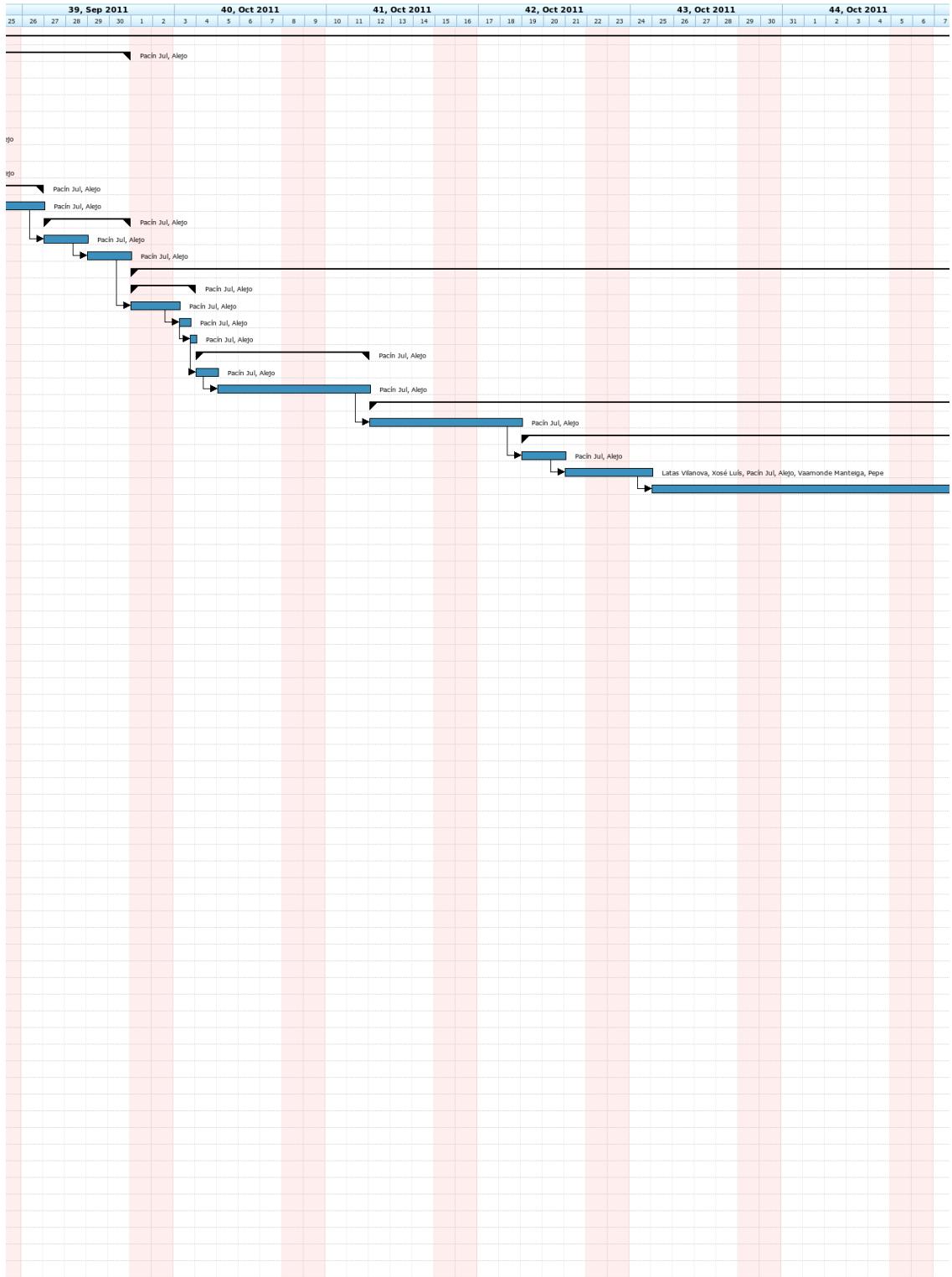


Figura A.2: Planificación inicial (p. 2).

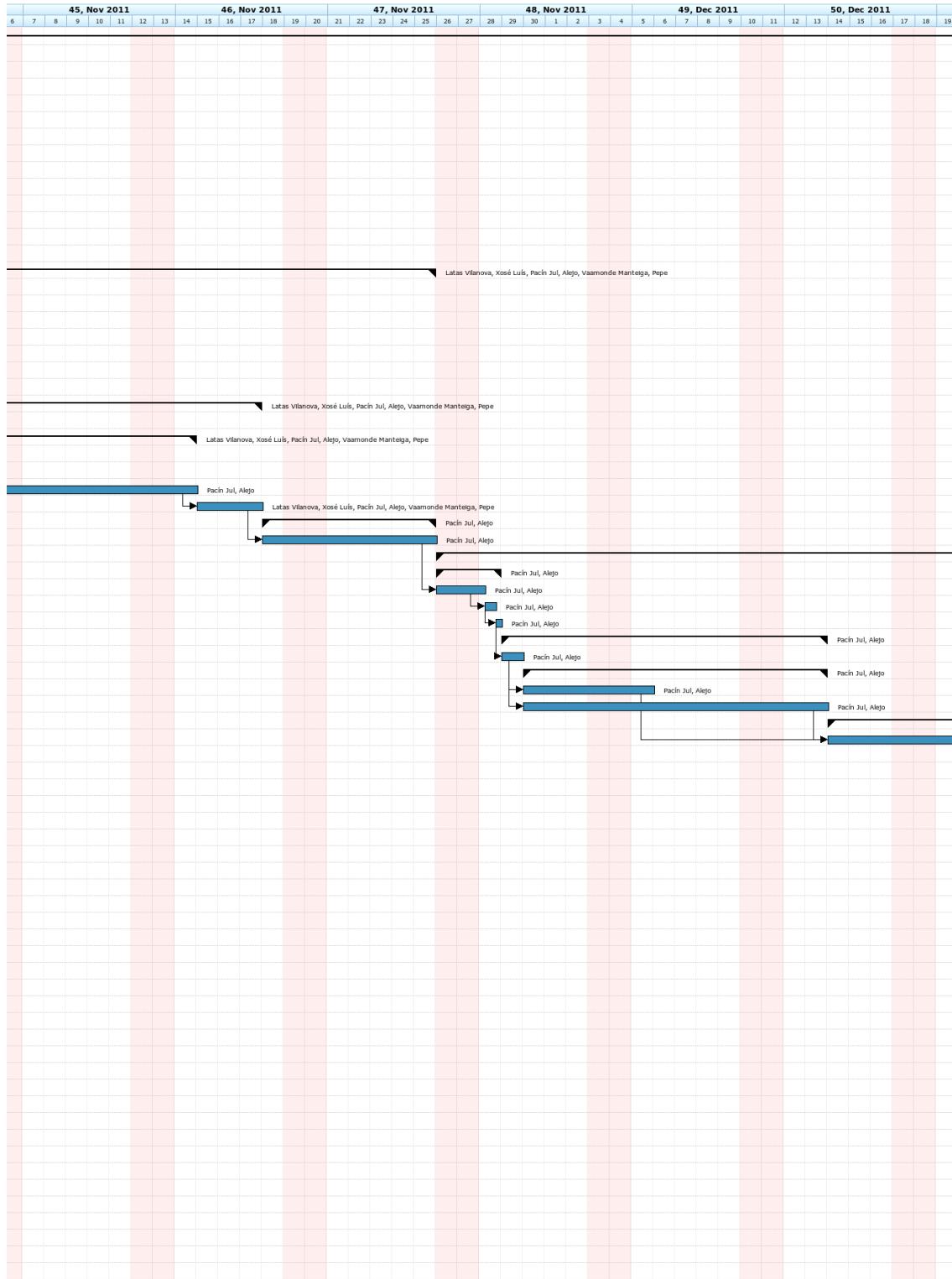


Figura A.3: Planificación inicial (p. 3).

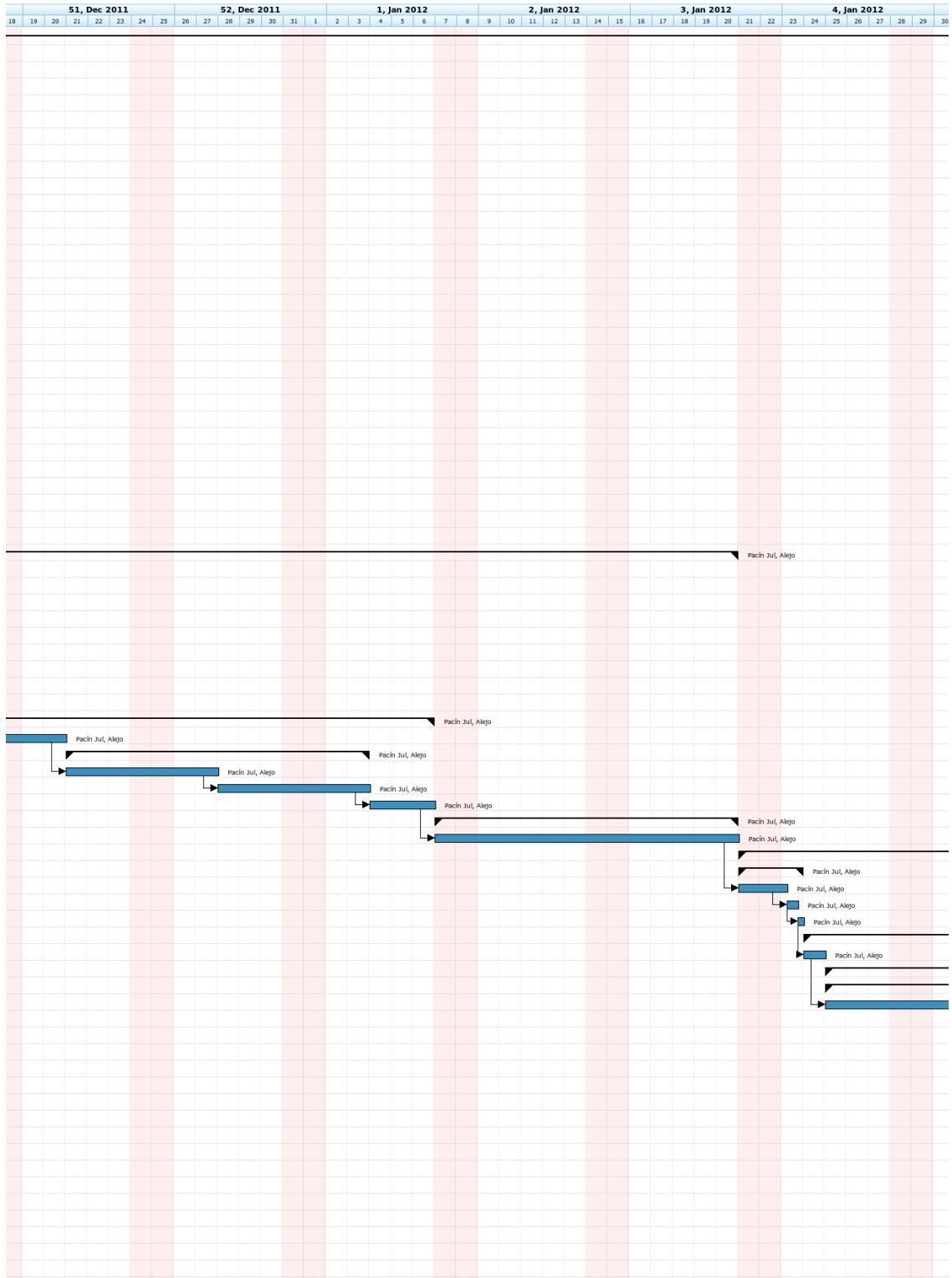


Figura A.4: Planificación inicial (p. 4).

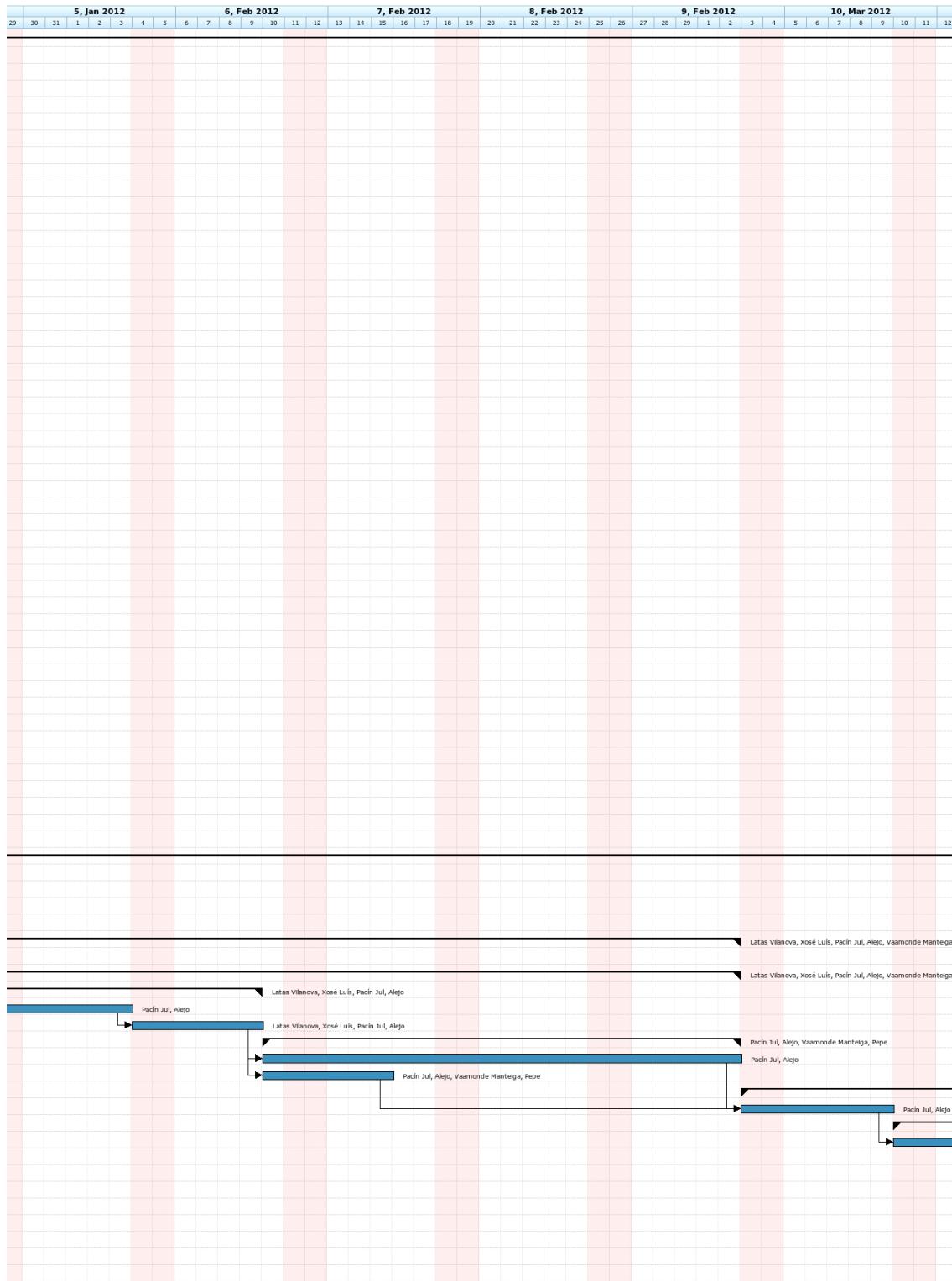


Figura A.5: Planificación inicial (p. 5).

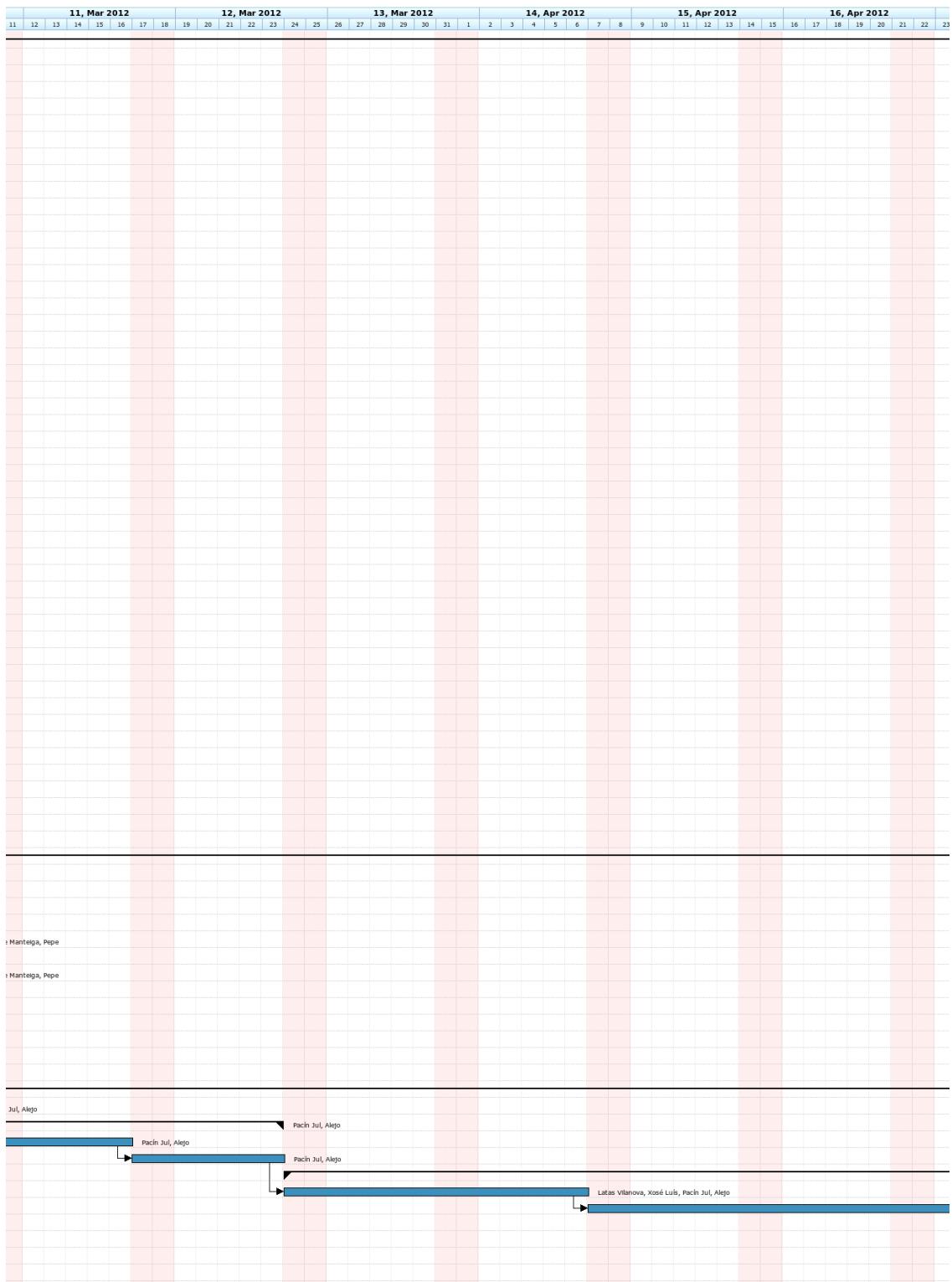


Figura A.6: Planificación inicial (p. 6).

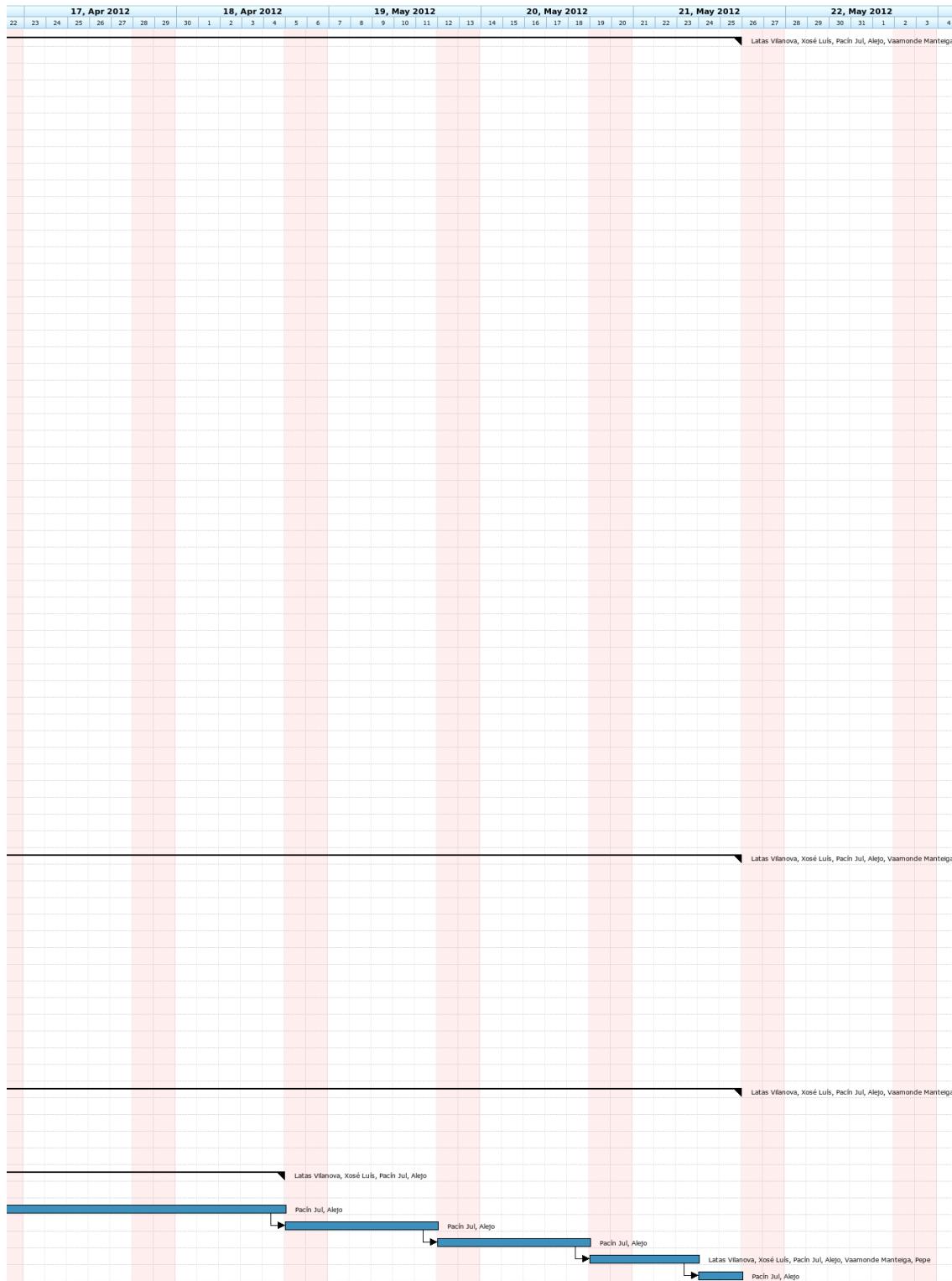


Figura A.7: Planificación inicial (p. 7).

	23, Jun 2012					24, Jun 2012					25, Jun 2012					26, Jun 2012													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1
Manteiga, Pepe																													
Manteiga, Pepe																													
Manteiga, Pepe																													
ape																													

Figura A.8: Planificación inicial (p. 8).

<http://finalplanning.proyecto-puding.org>

Tal e como se pode apreciar nas imaxes, nas primeiras fases do proxecto aforrouse algo máis dun 50 % das horas.

Foi na última fase, na de implementación e más concretamente na última fase de prototipado e más na codificación final. Outras como o prototipo hardware da fase de deseño ou os tests de integración tamén se foron moi arriba, pero causaron menos impacto polo número de horas en relación ó total.

En xeral, sobreestimamos bastante o tempo necesario para as fases iniciais áínda que non tiñan tanto impacto en horas e subestimamos moito as tarefas de codificación, que si tiveron un impacto moi alto.

Como se pode apreciar, finalmente o total de horas ascendeu a 966 fronte ás 688 horas estimadas inicialmente, cunha desviación total contraria do 40 %. Unha desviación alta se o consideramos un proxecto clásico de enxeñería, pero comprensible polo alto compoñente de incertidume como proxecto de I+D puro.

Tarefa	Horas estimadas	Horas reais	Diferença (h)	Diferença (%)
Análise de viabilidade				
Determinação 1				
Objetivos 1	1	1	0	0,00%
Alternativas 1	2	2	0	0,00%
Restrições 1	1	1	0	0,00%
Avaliação de alternativas e resolução de riscos 1				
Análise de riscos 1	4	4	0	0,00%
Protótipo 1	4	1	3	75,00%
Desenvolvimento e validação do seguinte nível do produto 1				
Planificação da seguinte fase ou ciclo 1	8	2	6	75,00%
Análise de requisitos				
Determinação 2				
Objetivos 2	1	1	0	0,00%
Alternativas 2	2	1	1	50,00%
Restrições 2	1	1	0	0,00%
Avaliação de alternativas e resolução de riscos 2				
Análise de riscos 2	4	1	3	75,00%
Protótipo 2	20	13	7	35,00%
Desenvolvimento e validação do seguinte nível do produto 2				
Simulações, modelos e programas de prova	20	3	17	85,00%
Requisitos hardware, software e econômicos				
Extracção de requisitos propria	8	13	-5	-42,50%
Extracção de requisitos por parte de experts	8	4	4	50,00%
Planificação da próxima fase ou ciclo 2				
Validação de requisitos	60	19	41	68,33%
Planificação do design	12	15	-3	-25,00%
Design				
Determinação 3				
Objetivos 3	1	1	0	0,00%
Alternativas 3	2	1	1	50,00%
Restrições 3	1	1	0	0,00%
Avaliação de alternativas e resolução de riscos 3				
Análise de riscos 3	4	1	3	75,00%
Protótipo 3	8	24	-16	200,00%
Protótipo hardware 3	20	3	17	85,00%
Protótipo software 3	20	3	17	85,00%
Desenvolvimento e validação do seguinte nível do produto 3				
Simulações, modelos e programas de prova 3	20	3	17	85,00%
Desenv. do produto hardware e software	20	1	19	95,00%
Desenv. do produto software	12	12	0	0,00%
Verificação e validação do design	40	6	34	85,00%
Planificação do desenvolvimento				
Desenvolvimento				

Figura A.9: Planificación final (p. 1).

Determinación 4							
Objetivos 4				1	1	0	0.00%
Alternativas 4				2	1	1	50.00%
Restricciones 4				1	1	0	0.00%
Avallación de alternativas e resolución de riescos 4							
Analise de riscos 4							
Prototipado operacional	Prototipo hardware 4						
	Integración do hardware			32	57	25	-78.13%
	Encapsulamento do hardware			16	0	16	100.00%
	Prototipo software 4						
	Desenvolvimento do protótipo			32	223	-191	-596.88%
	Gravación dos samples			16	0	16	100.00%
Desarrollo y validación del siguiente nivel del producto 4	Simulaciones, modelos e programas de prueba 4						
	Diseño detallado			20	4	16	80.00%
	Diseño hardware			20	1	19	95.00%
	Diseño software			20	4	16	80.00%
	Ensamblado y codificación						
	Ensamblado			40	0	40	100.00%
	Codificación			80	395	-315	-393.75%
Pruebas de unidad				20	30	-10	-50.00%
Integración e pruebas				20	71	-51	-255.00%
Pruebas de aceptación				12	11	1	8.33%
Implantación				8	16	-8	-100.00%
	Totals			688	966	-278	-40.41%
	Parciales						
	Análisis de viabilidad			36	19	17	47.22%
	Analise de requisitos			160	75	85	53.13%
	Diseño			56	92	62.16%	
	Desarrollo			344	816	472	-137.41%

Figura A.10: Planificación final (p. 2).

Apéndice B

Enquisa

NESTE apéndice exponse a enquisa mediante a que se levou a cabo o estudio de viabilidade.

B.1. Resultados

A continuación expóñense os resultados obtidos.

10 responses

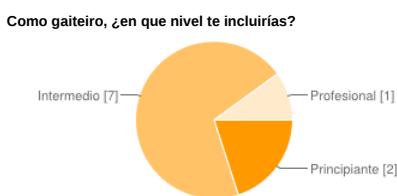
Resumen [Ver las respuestas completas](#)

¿Que anos tés?
14 18 17 41 22 28 23 35 19 24

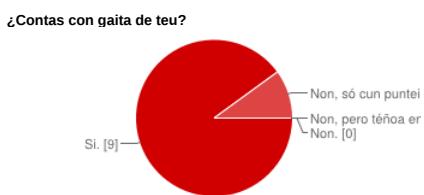
¿En que localidade resides?
Rábade Lugo Meira Neda Oviedo Rodeiro Antas de Ulla Rodeiro Alcalá de Henares Madrid



A Coruña	1	10%
Lugo	4	40%
Ourense	0	0%
Pontevedra	2	20%
Other	3	30%



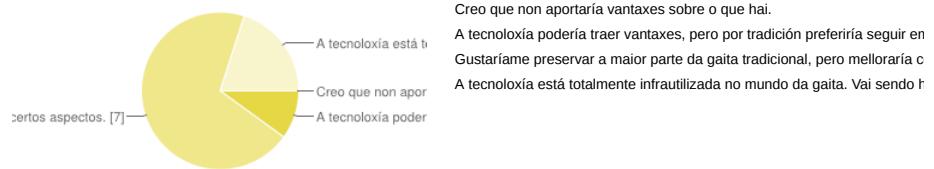
Principiante	2	20%
Intermedio	7	70%
Profesional	1	10%



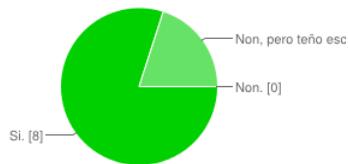
Si.	9	90%
Non, só cun punteiro de iniciación.	1	10%
Non, pero téñoa encargada xa.	0	0%
Non.	0	0%

¿Que pensas da aplicación da tecnoloxía actual ó mundo da gaita?

Figura B.1: Resultados da enquisa (p. 1).

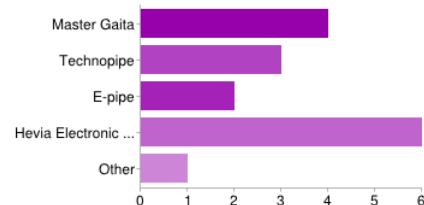


¿Sabes o que é unha gaita ou punteiro MIDI?



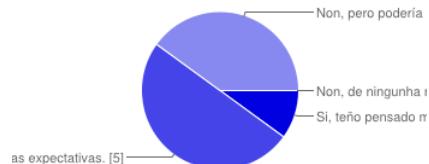
Si.	8	80%
Non, pero teño escido falar delas.	2	20%
Non.	0	0%

En caso afirmativo, ¿coñeces algunha das seguintes?



People may select more than one checkbox, so percentages may add up to more than 100%.

¿Estarías interesado nunha gaita ou punteiro MIDI?



Si, teño pensado mercar unha.	1	10%
Si, pero só se cumple coas miñas expectativas.	5	50%
Non, pero podería chegar a interesarme se xurde algo xeitoso.	4	40%
Non, de ningunha maneira.	0	0%

En caso afirmativo, ¿que uso lle darías?

Figura B.2: Resultados da enquisa (p. 2).

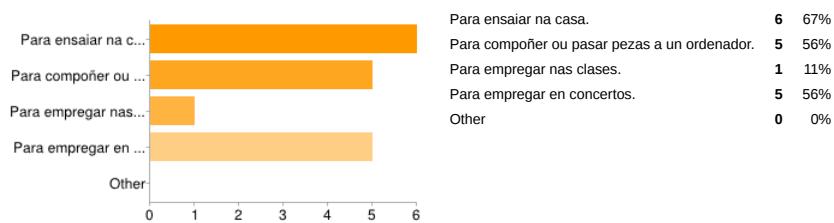
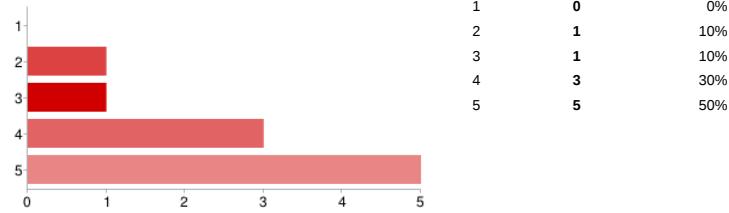
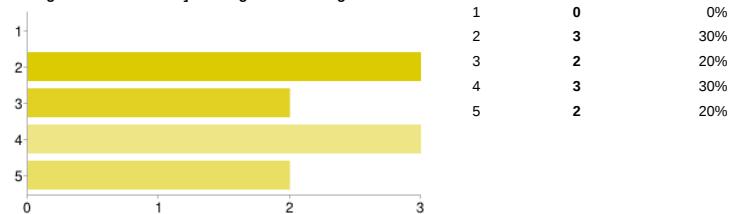


Figura B.3: Resultados da enquisa (p. 3).

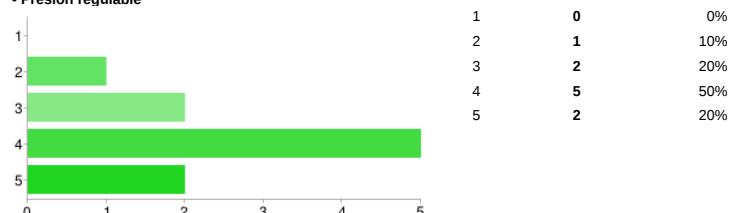
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Ausencia de retardos] - Ausencia de retardos



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Salvagarda da configuración entre usos] - Salvagarda da configuración entre usos

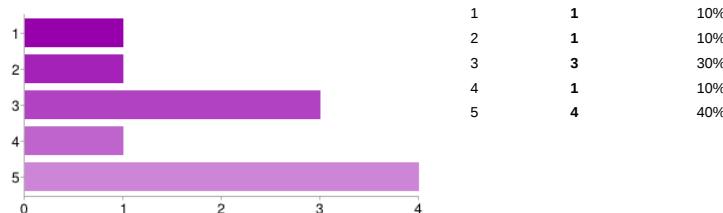


¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Presión regulable] - Presión regulable

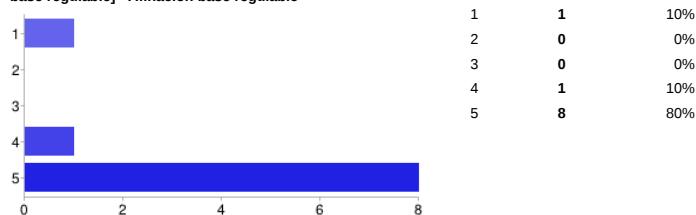


¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Sensores dos dedos regulables independentemente] - Sensores dos dedos regulables independentemente

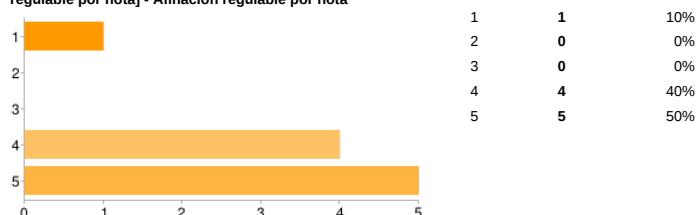
Figura B.4: Resultados da enquisa (p. 4).



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Afinación base regulable] - Afinación base regulable



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Afinación regulable por nota] - Afinación regulable por nota



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Dixitación personalizable] - Dixitación personalizable

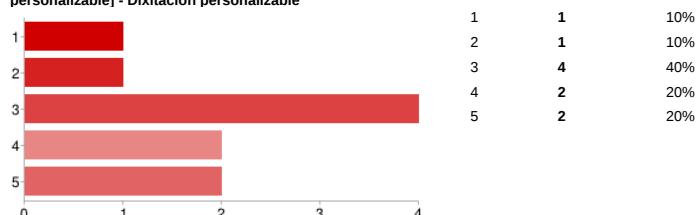
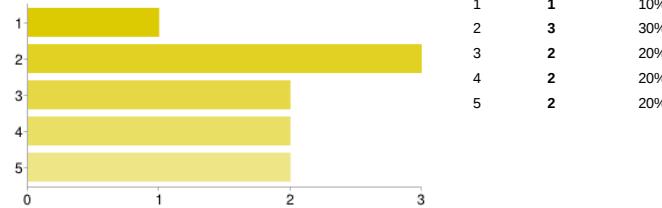
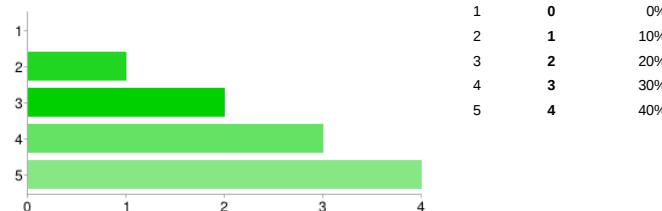


Figura B.5: Resultados da enquisa (p. 5).

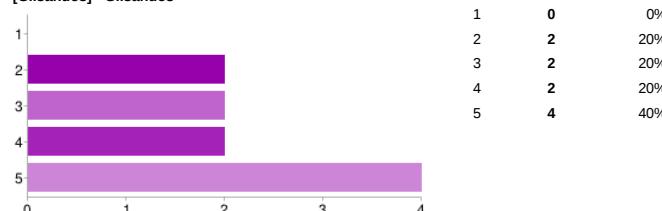
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Sensor da presión do fol deshabilitable] - Sensor da presión do fol deshabilitable



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Vibrato] - Vibrato



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Glisandos] - Glisandos



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade? [Bordóns] - Bordóns

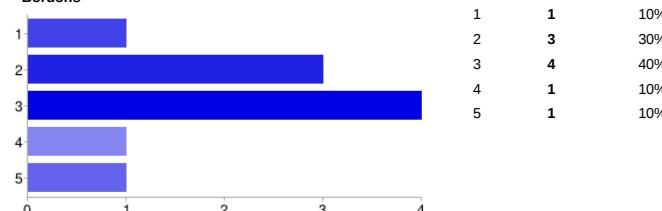
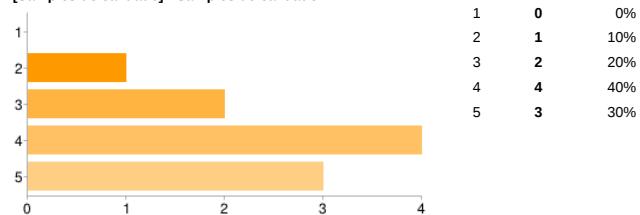
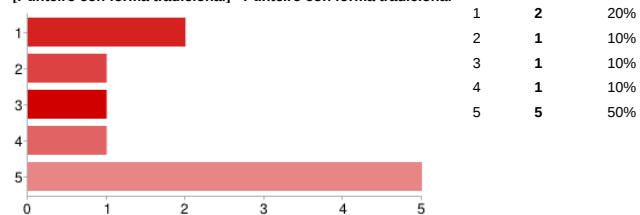


Figura B.6: Resultados da enquisa (p. 6).

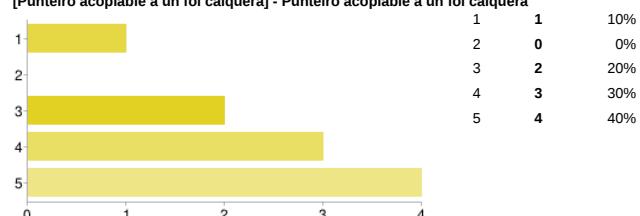
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
[Samples de calidade] - Samples de calidad



¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
[Punteiro con forma tradicional] - Punteiro con forma tradicional



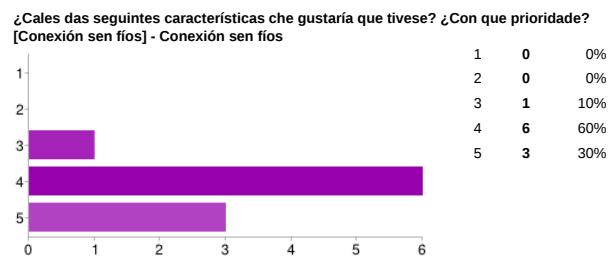
¿Cales das seguintes características che gustaría que tivese? ¿Con que prioridade?
[Punteiro acopiable a un fol calquera] - Punteiro acopiable a un fol calquera



¿Cales das siguientes características che gustaría que tivese? ¿Con que prioridade?
[Posibilidade de conexión a un ordenador] - Posibilidad de conexión a un ordenador



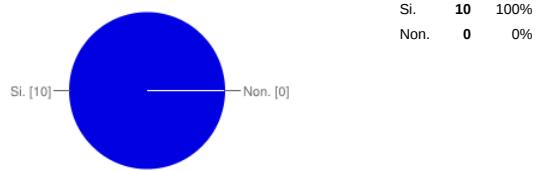
Figura B.7: Resultados da enquisa (p. 7).



Indica outras características que che gustaría que tivese, ordeadas por prioridade

-Cascos ou auriculares de escutar musica que se conecten a gaita para oir ti so, sin que os demais as oian ou que non oian case neda. Tensitura do instrumento de 2 octavas cromáticas completas e a sensible inferior (Si4-Do7) idéntico tamaño y forma que un punteiro normal que sexa cómoda e non moi grande o fol

¿Estarías disposto a mercar unha gaita MIDI coas características anteriores (incluídas as que ti mesmo engadiches)?



De ser así, ¿canto estarías dispuesto a pagar por unha?

1750 €	500	1500	500	1000	1500	500	150	300	500
--------	-----	------	-----	------	------	-----	-----	-----	-----

En caso contrario, ¿por que no?

Sabendo que as gaitas MIDI presentes hoxe en día no mercado oscilan entres os 275 € e os 2800 € e que a maior parte delas non satisfacen as características anteriores ¿canto estarías disposto a pagar por unha que as inclúise todas ou case todas?

Number of daily responses

Figura B.8: Resultados da enquisa (p. 8).

Editar formulario - [Enquisa] - Google Docs

<https://docs.google.com/spreadsheet/gform?key=0AhN...>



Figura B.9: Resultados da enquisa (p. 9).

Apéndice C

Arduino

ARDUINO [81] é unha plataforma de hardware e software libre, baseada nunha placa cun microcontrolador e un entorno de desenvolvemento (IDE), deseñada para facilitar o uso da electrónica en proxectos multidisciplinares.

O hardware consiste nunha placa cun microcontrolador *Atmel AVR* e portos de entrada/saída. Os microcontroladores más usados son o *Atmega168*, *Atmega328*, *Atmega1280*, *ATmega8* pola súa sinxeleza e baixo custo que permiten o desenvolvemento de múltiples deseños. Doutra banda o software consiste nun contorno de desenvolvemento que implementa a linguaxe de programación con *Processing* e *Wiring* e o cargador de arrinque (*boot loader*) que corre na placa.

Arduino pódese utilizar para desenvolver obxectos interactivos autónomos ou pode ser conectado a software do computador (por exemplo: *Macromedia Flash*, *Processing*, *Max/MSP*, *Pure Data*). As placas pódense montar a man ou adquirirse. O contorno de desenvolvemento integrado libre pódese descargar gratuitamente.

Ó ser open-hardware, tanto o seu deseño como a súa distribución é libre. É dicir, pode utilizarse libremente para o desenvolvemento de calquera tipo de proxecto sen adquirir ningunha licenza.

O proxecto Arduino recibiu unha mención honorífica na categoría de *Comunidades Dixitais* no *Prix Ars Electronica* de 2006.

C.1. Historia

En 2005, en Ivrea, Italia no *Interaction Design Institute* (fundado por *Olivetti* e *Telecom Italia*), iniciouse o proxecto para facer un dispositivo controlador programable construído por estudiantes de deseño para proxectos interactivos de baixo custe para sistemas de prototipado. Os fundadores Massimo Banzi e David Cuartielles chamaron ó proxecto despois *Arduin de Ivrea*, tomando o nome dun bar que o tomara a súa vez de *Arduino d'Ivrea*, primeiro rei de Italia en 1002. *Arduino* é tamén un nome de home, que en italiano significa *amigo valente*.

O proxecto Arduino é unha plataforma de código aberto (IDE) para plataformas (GNU/Linux, Mac e Windows). O artista colombiano e programador Hernando Barragán creouno como tese no *Interaction Design Institute Ivrea* baixo a supervisión de Massimo Banzi e Casey Reas. *Wiring* baséase en *Processing* e o seu entorno de desenvolvemento integrado, que fora creado por Casey Reas e Ben Fry.

Grazas á base de software común, dos creadores do proxecto para a comunidade Arduino foi posible desenvolver programas para conectar con este hardware más ou menos calquera obxecto electrónico, ordenador, sensores, pantallas e actuadores. Despois de anos de experimentación, agora é posible beneficiarse dunha ampla base de datos e información.

C.2. Instalación

C.2.1. Windows

Para a instalación da placa Arduino no sistema operativo Windows convén seguir os seguintes pasos.

Coa placa desconectada:

- Descargar e instalar o *Java Runtime Environment* (J2RE).
- Descargar a última versión do *IDE Arduino*.

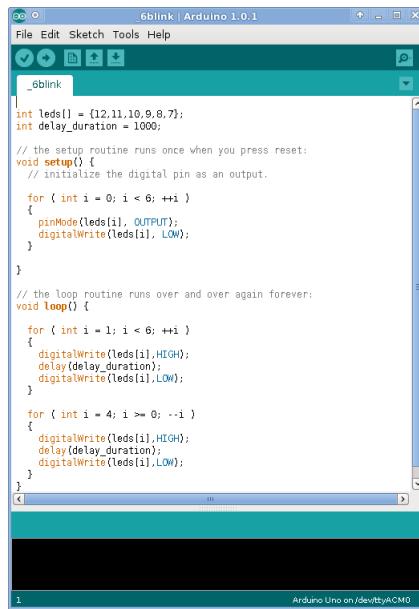


Figura C.1: IDE Arduino.

É recomendable descomprimir o ficheiro no directorio raíz (C:\) mantendo a estrutura orixinal.

Entre todos os cartafoles creados no directorio Arduino convén destacar as seguintes:

c:\arduino-0012\hardware\bootloader

Esta contén o software necesario para cargar o firmware no chip *Atmega168*, para traballar con Arduino. Só se utiliza en caso empregar placas artesanais.

c:\arduino-0012\drivers

Contén os drivers necesarios para o funcionamento da placa Arduino cun PC con S.O. Windows: *FTDI USB Drivers*.

Para instalar estes drivers, agora si, conectar a placa USB. Abrirase automaticamente o asistente de Windows para novo hardware atopado.

1. Seleccionar “Non polo momento” e premer en “Seguinte”.

2. Seleccionar “Instalar desde unha lista ou localización específica (avanzado)” e premer en “Seguinte”.
3. “Buscar o controlador máis adecuado nestas localizacíons”, premer en “Examinar”. Seleccionar o cartafol onde se descomprimira o driver e premer “Seguinte”.

Se non houbo ningún problema o driver da placa estará instalado.

Agora executamos o ficheiro *Arduino.exe* para abrir a interface. Aquí configuraremos o porto USB onde temos conectada a placa para empezar a traballar.

C.2.2. GNU/Linux

Para instalar Arduino nun sistema GNU/Linux necesitamos os seguintes programas para resolver as dependencias:

- Sun Java runtime, *jre*, ou equivalente.
- *avr-gcc*, compilador para a familia de microcontroladores AVR de Atmel.
- *avr-libc*, libc do compilador avr-gcc.

Para instalalos, podemos utilizar o xestor de paquetes ou a terminal de comandos:

```
apt-get install sun-java5-jre gcc-avr avr-libc
```

Nalgunhas distribucións convén desinstalar, se non é necesario, o paquete *brltty*. Este encárgase de permitir o acceso á terminal para persoas cegas a través dun dispositivo especial en Braille.

```
killall brltty
apt-get remove brltty
```

Os dous síntomas deste problema son:

- Non aparece a opción */dev/tty/USB0* no menú *Tools, Serial Port*.

- Se se observa o LED Rx da placa Arduino, e este se ilumina de 3 a 5 veces cada 5 ou 6 segundos.

Por último, descargamos o framework de Arduino. Descomprimímolo no cartafol desexado e executámolo:

```
./arduino
```

Se todo foi ben xa o teremos en funcionamento.

C.3. Hardware

C.3.1. Placas Arduino

Unha placa Arduino consta de un microcontrolador de 8-bit *Atmel AVR* con compoñentes complementarios para facilitar a programación e a incorporación noutrous deseños. Un aspecto importante de Arduino é a forma estándar na que dispónense os conectores, permitindo que a placa da CPU poida ser conectada a unha variedade de módulos intercambiables adicionais coñecidos como *shields*. Algúns comunicanse cos shields da placa Arduino directamente cos diferentes pins, pero moitos shields son individualmente direccionables a través dun bus I2C de serie, o que permite que moitos shields poidan apíllarse e empregarse en paralelo. As placas Arduino oficiais empregan a serie de circuitos integrados *megaAVR*, específicamente o *ATmega8*, *ATmega168*, *ATmega328*, *ATmega1280* e *ATmega2560*. Un feixe doutros procesadores compatibles foron empregados por Arduino. A maioría das placas inclúen un regulador lineal de 5 voltios e un oscilador de 16 MHz de reloxo (ou oscilador de cuarzo nalgunhas variantes), aínda que algúns deseños, tales como o *LilyPad* son a 8 MHz e prescinde do regulador de tensión integrado na placa, debido a restricións determinadas polo deseño. Un microcontrolador Arduino está tamén pre-programado con un xestor de arranque que simplifica a carga de programas na memoria flash do chip, en comparación con outros dispositivos que normalmente necesitan un programador externo ou hardware adicional.

A nivel conceptual, cando se utiliza a pila de software de Arduino, todas as placas se programan a través dunha conexión *RS-232*, pero a forma en que isto se fai varía segundo a versión do hardware. As placas serie Arduino conteñen un circuíto inversor simple para converter entre o nivel *RS-232* e os sinais de nivel TTL. As actuais placas Arduino prográmanse a través de portos USB, implementado a través dun circuíto *FT232 FTDI*. Algunhas variantes, como o *Mini Arduino* e o non oficial *Boarduino*, empregan un cable de datos USB, *Bluetooth* ou outros métodos. Cando se emprega con ferramentas para microcontroladores tradicionais en vez do *IDE Arduino*, emprégase o estándar de programación *AVR ISP*.

A maioría das placa Arduino conectan os pins do microcontrolador como entradas/saídas para o seu uso por otros circuítos. O *Diecimila*, *Due mil nove*, e o actual *Uno* proporcionan 14 pins E/S dixitais, seis dos cales poden producir pulsos de ancho variable, e seis entradas analóxicas (PWM). Estes pins atópanse na parte superior do taboleiro, a través de conectores femia de 0,1 polgadas. Tamén están dispoñibles comercialmente varias shields oficiais de tipo Plug-in.

C.3.2. Linguaxe de programación Arduino

A plataforma Arduino prográmase mediante o uso dunha linguaxe propia baseada na popular linguaxe de programación de alto nível *Processing*. Con todo, é posible utilizar outras linguaxes de programación e aplicacións populares en Arduino. Algúns exemplos son:

- Java
 - Flash (mediante ActionScript)
 - Processing
 - Pure Data
 - MaxMSP (contorna gráfica de programación para aplicacións musicais, de audio e multimedia)
 - VVVV (síntese de vídeo en tempo real)
-

- Adobe Director
 - Python
 - Ruby
 - C
 - C++ (mediante libSerial ou en Windows)
 - C#
 - Cocoa/Objective-C (para Mac OS X)
 - GNU/Linux TTY (terminais de GNU/Linux)
 - 3DVIA Virtools (aplicacións interactivas e de tempo real)
 - SuperCollider (síntese de audio en tempo real)
 - Instant Reality (X3D)
 - Liberlab (software de medición e experimentación)
 - BlitzMax (con acceso restrinxido)
 - Squeak (implementación libre de Smalltalk)
 - Mathematica
 - Matlab
 - Minibloq (Contorna gráfica de programación, corre tamén en OLPC)
 - Isadora (Interactividade audiovisual en tempo real)
 - Perl
 - Visual Basic .NET
 - VBScript
 - Gambas
-

Isto é posible debido a que Arduino se comunica mediante a transmisión de datos en formato serie que é algo que a mayoría das linguaxes anteriormente citadas soportan. Para os que non soportan o formato serie de forma nativa, é posible utilizar software intermediario que traduza as mensaxes enviadas por ambas as partes para permitir unha comunicación fluída. É bastante interesante ter a posibilidade de interactuar Arduino mediante esta gran variedade de sistemas e linguaxes posto que dependendo de cales sexan as necesidades do problema que imos resolver poderemos aproveitarnos da gran compatibilidade de comunicación que ofrece.

C.3.2.1. Funcións básicas e operadores

Arduino está baseado en C e soporta todas as funcións do estándar C e algunhas de C++. A continuación móstrase un resumo con tódalas estruturas da linguaxe Arduino:

C.3.2.1.1. Sintaxe básica

- Delimitadores: ;, {}
- Comentarios: //, /* */
- Cabeceiras: #define, #include
- Operadores aritméticos: +, -, *, /, %
- Asignación: =
- Operadores de comparación: ==, !=, <, >, <=, >=
- Operadores Booleanos: &&, ||, !
- Operadores de acceso a punteiros: *, &
- Operadores de bits: &, —, ^, <<, >>
- Operadores compostos:
 - Incremento/decremento de variables: ++, --
 - Asignación e operación: +=, -=, *=, /=, &=, |=

C.3.2.1.2. Estruturas de control

- Condicionais: if, if...else, switch case
- Bucles: for, while, do... while
- Bifurcaciones e saltos: break, continue, return, goto

C.3.2.1.3. Variables

En canto ao tratamento das variables tamén comparte un gran parecido coa linguaxe C.

C.3.2.1.4. Constantes

- HIGH / LOW: niveis alto e baixo en pines. Os niveis altos son aqueles de 3 voltios ou máis.
- INPUT / OUTPUT: entrada ou saída
- true / false

C.3.2.1.5. Tipos de datos

- void
 - boolean
 - char
 - unsigned char
 - byte
 - int
 - unsigned int
 - word
 - long
-

- unsigned long
- float
- double
- string
- array

C.3.2.1.6. Conversión entre tipos

Estas funcións reciben como argumento unha variable de calquera tipo e devolven unha variable convertida no tipo desexado.

- char()
- byte()
- int()
- word()
- long()
- float()

C.3.2.1.7. Cualificadores e ámbito das variables

- static
- volatile
- const

C.3.2.1.8. Utilidades

- sizeof()

C.3.2.1.9. Funcións básicas

En canto ás funcións básicas da linguaxe atopámonos coas seguintes:

C.3.2.1.9.1. E/S Dixital

- `pinMode(pin, modo)`
- `digitalWrite(pin, valor)`
- `int digitalRead(pin)`

C.3.2.1.9.2. E/S Analólica

- `analogReference(tipo)`
- `int analogRead(pin)`
- `analogWrite(pin, valor)`

C.3.2.1.9.3. E/S Avanzada

- `shiftOut(dataPin, clockPin, bitOrder, valor)`
- `unsigned long pulseIn(pin, valor)`

C.3.2.1.9.4. Tempo

- `unsigned long millis()`
- `unsigned long micros()`
- `delay(ms)`
- `delayMicroseconds(microsegundos)`

C.3.2.1.9.5. Matemáticas

- `min(x, e)`
 - `max(x, e)`
 - `abs(x)`
 - `constrain(x, a, b)`
-

- map(valor, fromLow, fromHigh, toLow, toHigh)
- pow(base, expoñente)
- sqrt(x)

C.3.2.1.9.6. Trigonometría

- sin(rad)
- cos(rad)
- tan(rad)

C.3.2.1.9.7. Números aleatorios

- randomSeed(semente)
- long random(máx)
- long random(mín, máx)

C.3.2.1.9.8. Bits e Bytes

- lowByte()
- highByte()
- bitRead()
- bitWrite()
- bitSet()
- bitClear()
- bit()

C.3.2.1.9.9. Interrupciones externas

- attachInterrupt(interrupción, función, modo)
 - detachInterrupt(interrupción)
-

C.3.2.1.9.10. Interrupcóns

- `interrupts()`
- `noInterrupts()`

C.3.2.1.9.11. Comunicación por porto serie

As funcións de manexo do porto serie deben ir precedidas de “Serial.” aínda que non necesitan ningunha declaración na cabeceira do programa. Por isto se consideran funcións base da linguaxe.

- `begin()`
- `available()`
- `read()`
- `flush()`
- `print()`
- `println()`
- `write()`

C.3.2.1.9.12. Manipulación de portos

Os rexistros de portos permiten a manipulación a máis baixo nivel e de forma mais rápida dos pines de E/S do microcontrolador das placas Arduino. Os pines das placas Arduino están repartidos entre os rexistros B(0-7), C (analóxicos) e D(8-13). Mediante as seguintes variables podemos ver e modificar o seu estado:

- `DDR[B/C/D]`: Data Direction Register (ou dirección do rexistro de datos) do porto B, C ó D. Serve para especificar que pines queremos usar como de entrada e cales de saída. Variable de lectura/escritura.
 - `PORT[B/C/D]`: Data Register (ou rexistro de datos) do porto B, C ó D. Variable de lectura/escritura.
-

- PIN[B/C/D]: Input Pins Register (ou rexistro de pinos de entrada) do porto B, C ó D. Variable de só lectura.

Por exemplo, para especificar que queremos utilizar os pinos 1 a 7 como saídas e o 0 como entrada, bastaría utilizar a seguinte asignación:

```
DDRD = B1111110 ;
```

C.3.2.1.10. A.V.R. Libc Os programas compilados con Arduino enlázanse contra *AVR Libc* polo que teñen acceso a algunhas das súas funcións. *AVR Libc* é un proxecto de software libre co obxectivo de proporcionar unha biblioteca C de alta calidade para utilizarse co compilador GCC sobre microcontroladores *Atmel AVR*. Componse de 3 partes:

- avr-binutils
- avr-gcc
- avr-libc

A maioría da linguaxe de programación Arduino está escrita con constantes e funcións de AVR e certas funcionalidades só se poden obter facendo uso de AVR.

C.3.2.1.10.1. Interrupcións

Para desactivar as interrupcións:

```
cli(); // desactiva as interrupcions globais
```

Para activalas:

```
\item sei(); // activa as interrupcions
```

Isto afectará ao temporizador e á comunicación serie. A función *delayMicroseconds()* desactiva as interrupcións cando se executa.

C.3.2.1.10.2. Temporizadores

A función *delayMicroseconds()* crea o menor retardo posible da linguaxe Arduino que rolda os 2us.

Para retardos mais pequenos débese utilizar a chamada de ensamblador *texitnop* (non operación). Cada sentenza *nop* executarase nun ciclo de máquina (16 Mhz): uns 62.5ns. Faríase da seguinte maneira:

```
--asm--("nop\n\t");
```

C.3.2.1.10.3. Manipulación de portos

A manipulación de portos con código AVR é mais rápida que utilizar a función *digitalWrite()* de Arduino.

C.3.2.1.10.4. Establecer bits en variables

cbi e *sbi* son mecanismos estándar (AVR) para establecer ou limpar bits en PORT e outras variables.

Será necesario utilizar as seguintes cabeceiras para poder utilizalos:

```
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= _BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif
```

Para utilizalas hai que pasaralles como argumento a variable PORT e un pin para establecelo ou limpalo.

Grazas a estes pequenos hacks teremos a posibilidade de mellorar os tempos de execución de certas tarefas críticas ou daquelas que se repitan moitas veces obtendo mellores resultados. Non obstante o código fonte que escribamos resultará probablemente menos legible se os utilizamos polo que haberá que sopesalo en función das nosas necesidades.

C.3.2.1.11. Primeiro contacto: Ola Mundo en Arduino

O primeiro paso antes de comprobar que a instalación é correcta e empezar a traballar con Arduino é abrir algúns exemplos prácticos que veñen dispoñibles co dispositivo. É recomendable abrir o exemplo *led_blink* que atoparemos no menú *File, Sketchbook, Examples, led_blink*. Este código crea unha intermitencia por segundo nun led conectado no pin 13. É cuestión de comprobar que o código é correcto, para iso, prememos o botón que é un triángulo (en forma de *play*) e seguidamente faremos un *upload* (que é a frecha cara á dereita) para cargar o programa á placa. Se o led empeza a pestanexar, todo estará correcto.

Vexamos o código necesario para conseguilo:

```
# define LED_PIN 13

void setup () {

    // Activamos o pin 13 para saída digital
    pinMode (LED_PIN, OUTPUT);

}

// Bucle infinito
void loop () {

    // Acendemos o led enviando un sinal alto
    digitalWrite (LED_PIN, HIGH);
```

```
// Esperamos un segundo (1000 ms)
delay (1000);

// Apagamos o led enviando un sinal baixo
digitalWrite (LED_PIN, LOW);

// Esperamos un segundo (1000 ms)
delay (1000);

}
```

A orde de execución será: primeiro faise unha chamada á función *init()* que inicializa o programa, despois execútase a función *setup()* que configura diversos parámetros, e por último execútase un bucle *while(1)* que chama repetidamente á función *loop*. Todo iso execútase dentro de *main()* e podería indicarse explícitamente (no caso anterior encárgase o IDE de engadir o código que omitiu).

C.4. Bibliotecas en Arduino

Para facer uso dunha biblioteca en *Sketch* (o IDE de Arduino), basta con facer clic sobre *Import Library* no menú, escoller unha biblioteca e engadirase o *#include* correspondente.

- Serial: lectura e escritura polo porto serie.
- EEPROM: lectura e escritura no almacenamento permanente.
 - read()
 - write()
- Ethernet: conexión a Internet mediante *Arduino Ethernet Shield*. Pode funcionar como servidor que acepta peticóns remotas ou como cliente. Permiteñase até catro conexións simultaneamente.
 - Servidor
 - Server()

- begin()
- available()
- write()
- print()
- println()
- Cliente
 - Client()
 - connected()
 - connect()
 - write()
 - print()
 - println()
 - available()
 - read()
 - flush()
 - stop()
- Firmata: comunicación con aplicáns de computador utilizando o protocolo estándar do porto serie.
- LiquidCrystal: control de LCDs con chipset *Hitachi HD44780* ou compatibles. A biblioteca soporta os modos de 4 e 8 bits.
- Servo: control de servo motores. A partir da versión 0017 de Arduino a biblioteca soporta até 12 motores na mayoría de placas *Arduino* e 48 na *Arduino Mega*.
 - attach()
 - write()
 - writeMicroseconds()
 - read()
 - attached()
 - detach()

- O manexo da biblioteca é bastante sinxelo. Mediante *attach(número de pin)* engadimos un servo e mediante *write* podemos indicar os graos que queremos que teña o motor (habitualmente de 0 a 180).
- SoftwareSerial: comunicación serie en pines dixitais. Por defecto Arduino inclúe comunicación só nos pines 0 e 1 pero grazas a esta biblioteca podemos realizar esta comunicación co resto de pines.
- Stepper: control de motores paso a paso unipolares ou bipolares.
 - Stepper(steps, pin1, pin2)
 - Stepper(steps, pin1, pin2, pin3, pin4)
 - setSpeed(rpm)
 - step(steps)

O manexo é sinxelo. Basta con iniciar o motor mediante *Stepper* indicando os pasos que ten e os pines aos que esta asociado. Indícase a velocidade á que queiramos que vire en revolucóns por minuto con *setSpeed(rpm)* e indícanse os pasos que queremos que avance con *step(pasos)*.

- Wire: envío e recepción de datos sobre unha rede de dispositivos ou sensores mediante *Two Wire Interface* (TWI/I2C). Ademais as bibliotecas *Matrix* e *Sprite* de *Wiring* son totalmente compatibles con Arduino e serven para manexo de matrices de leds. Tamén se ofrece información sobre diversas bibliotecas desenvolvidas por contribuidores diversos que permiten realizar moitas tarefas.

C.4.1. Creación de bibliotecas

Ademais das bibliotecas base, as que son compatibles e as que achegaron outras persoas temos a posibilidade de escribir nosa propia biblioteca. Isto é moi interesante por varias razóns: permite dispor de código que pode reutilizarse noutras proxectos de forma cómoda; permítenos manter o código fonte principal separado das bibliotecas de forma que sexan mantenibles de forma separada; e a organización dos programas construídos é máis clara e elegante.

Vexamos un exemplo da creación dunha biblioteca que envía código *Morse*.

Creamos o ficheiro *Morse.h* que inclúe a definición da clase *Morse* que ten 3 funcións: un construtor (*Morse()*), unha función para enviar 1 punto (*dot()*) e unha función para enviar unha raia (*dash()*). A variable *_pin* permite indicar o pin que imos utilizar.

```
/*
Morse.h - Library for flashing Morse code.

Created by David A. Mellis, November 2, 2007.

Released into the public domain.

*/
#ifndef Morse_h
#define Morse_h
#include "WProgram.h"

class Morse {
public:
    Morse(int pin);
    void dot();
    void dash();
private:
    int _pin;
}
```

```
};  
# endif
```

Ademais necesitaremos un ficheiro *Morse.cpp* co código das funcións declaradas. A continuación móstrase o código:

```
/*  
  
Morse.cpp – Library for flashing Morse code.  
  
Created by David A. Mellis, November 2, 2007.  
  
Released into the public domain.  
  
*/  
  
# include "WProgram.h"  
# include "Morse.h"  
  
Morse :: Morse( int pin ) {  
  
    pinMode( pin , OUTPUT );  
  
    _pin = pin ;  
  
}  
  
void Morse :: dot() {  
  
    digitalWrite( _pin , HIGH );  
  
    delay( 250 );
```

```
    digitalWrite( _pin , LOW );  
  
    delay( 250 );  
  
}  
  
void Morse :: dash() {  
  
    digitalWrite( _pin , HIGH );  
  
    delay( 1000 );  
  
    digitalWrite( _pin , LOW );  
  
    delay( 250 );  
  
}
```

E con isto xa poderiamos utilizar a biblioteca mediante o correspondente `#include`. Se quixeramos enviar un SOS polo pin 13 bastaría con chamar a `Morse(13)` e executar:

```
morse . dot();  morse . dot();  morse . dot();
```

```
morse . dash();  morse . dash();  morse . dash();
```

```
morse . dot();  morse . dot();  morse . dot();
```

Apéndice D

XBee

NESTE apéndice exponse todo o relacionado coa tecnoloxía *XBee*.

XBee [1] é o nome comercial de *Digi International* para unha familia de módulos de radio con formato compatible. As primeiras radios *XBee* comercializáronse baixo a marca *MaxStream* en 2005 e estaban baseados no estándar IEEE 802.15.4-2003, deseñado para comunicacóns punto a punto e en estrela e velocidades de transmisión sen fíos de 250 Kbps.

Inicialmente comercializáronse dous modelos, o *XBee* de 1 mW e baixo custe e o *XBee-PRO* de 100 mW. Desde a comercialización inicial, introduciuse no mercado unha nova serie de radios XBee e agora todas se venden baixo a marca Digi.

As radios XBee precisan dun mínimo de catro conexións para ser usadas: alimentación (3,3 V), terra e entrada e saída de datos (UART), sendo recomendado tamén o uso de liñas Reset e Sleep. Ademais, a maioría das familias XBee contan con algunas liñas de control de fluxo, I/O, A/D e de sinalización integradas. A versión programable do XBee conta cun procesador adicional para código de usuario.

A familia de radios XBee compónse de (figura D.1):

Product	Specifications*					Networking Features		
	Frequency	Power Output	Maximum Range	RF Data Rate	Protocol	Multipoint	Mesh	
<u>XBee ZB</u>	2.4 GHz	1.25/2 mW	120 m	250 Kbps	ZigBee		✓	
<u>XBee-PRO ZB</u>	2.4 GHz	63 mW*	3.2 km	250 Kbps	ZigBee		✓	
<u>XBee ZB SMT</u>	2.4 GHz	3.1/6.3 mW	1200 m	250 Kbps	ZigBee		✓	
<u>XBee-PRO ZB SMT</u>	2.4 GHz	63 mW	3.2 km	250 Kbps	ZigBee		✓	
<u>XBee 802.15.4</u>	2.4 GHz	1 mW	90 m	250 Kbps	802.15.4	✓		
<u>XBee-PRO 802.15.4</u>	2.4 GHz	63 mW*	1.6 km	250 Kbps	802.15.4	✓		
<u>XBee-PRO 900HP</u>	900 MHz	250 mW	45 km**	200 Kbps	Proprietary (Multipoint and DigiMesh)	✓	✓	
<u>XBee-PRO XSC</u>	900 MHz	100 mW	24 km**	9.6 Kbps	Proprietary	✓		
<u>XBee-PRO 900</u>	900 MHz	50 mW	10 km**	156 Kbps	Proprietary	✓		
<u>XBee-PRO DigiMesh 900</u>	900 MHz	50 mW	10 km**	156 Kbps	DigiMesh		✓	
<u>XBee DigiMesh 2.4</u>	2.4 GHz	1 mW	90 m	250 Kbps	DigiMesh		✓	
<u>XBee-PRO DigiMesh 2.4</u>	2.4 GHz	63 mW*	1.6 km	250 Kbps	DigiMesh		✓	
<u>XBee-PRO 868</u>	868 MHz	350 mW	80 km**	24 Kbps	Proprietary	✓		
<u>XBee Wi-Fi</u>	2.4 GHz	16 dBm	300 m	65 Mbps	802.11bg	✓		
<u>XBee 865LP</u>	865 MHz	12dBm	4 km	80 kbps	Proprietary (Multipoint and DigiMesh)	✓	✓	
<u>XBee 868LP</u>	868 MHz	12dBm	4 km	80 kbps	Proprietary (Multipoint and DigiMesh)	✓	✓	

Figura D.1: Familia XBee [1].

D.1. Formatos, antenas e modos de datos

Os módulos XBee están disponibles en dous formatos de montaxe: *Through-Hole* e *Surface Mount*.

Todos os modelos de XBee (a excepción do *XBee 868LP*) no popular formato Through-Hole de 20 pins. Certos módulos están tamén disponibles en formato Surface Mount de 37 pads, formato popular en produtos de alta demanda debido ós reducidos custos da tecnoloxía SMT.

Os módulos XBee veñen normalmente con varias opcións de antena, incluíndo *U.FL*, *PCB Embedded*, *Wire* e *RPSMA*.

Os módulos XBee poden traballar en modo de datos transparente ou en modo API baseado en paquetes. No modo transparente, os datos disponibles no pin Data IN (DIN) transmítense directamente e sen fíos ós módulos receptores sen modificación ningunha. Os paquetes entrantes poden ser direccionados a un obxectivo (punto a punto) ou repetilos a múltiples obxectivos (estrela). Este modo emprégase principalmente en casos no que o protocolo existente non tolera cambios no formato de datos. Para configurar un módulo empréganse comandos AT. En modo API os datos son empaquetados permitindo o direccionamento, a configuración de parámetros e o control de recepción de paquetes, incluíndo a teledetección e o control dos pins dixitais de E/S e os pins de entrada analóxicos.

D.2. XBee ZB

Os módulos Zigbee, XBee e XBee-PRO ZB (figura D.2, proporcionan conectividade sen fíos de baixo custo a dispositivos de redes ZigBee en malla (mesh).

D.3. ZigBee

Zigbee é o nome da especificación dun conxunto de protocolos de comunicación sen fíos de alto nivel para a súa utilización en radiodifusión dixital de baixo consumo, baseada no estándar *IEEE 802.15.4* de redes sen fíos de área persoal



Figura D.2: XBee ZB.

(WPAN). O seu obxectivo son as aplicacións que requiren de comunicacións seguras con baixa taxa de envío de datos e maximización da vida útil das súas baterías.

A tecnoloxía definida pola especificación de ZigBee está deseñada para ser máis simple e menos custosa que outras WPAN, coma Bluetooth ou Wi-Fi.

As redes ZigBee están securizadas a través do uso de claves de cifrado simétrico de 128 bits. As distancias de transmisión van desde os 10 ós 100 metros, dependendo da potencia de saída e das características ambientais.

ZigBee foi concebido en 1998, estandarizado en 2003 e revisado en 2006. O nome refírese á danza das abellas.

D.3.1. Visión xeral

ZigBee é un estándar para redes en malla sen fíos de baixo consumo e baixo custe. O baixo custe permite que a tecnoloxía sexa empregada de forma masiva en aplicacións de control e monotorización sen fíos. O baixo consumo permite máis autonomía con baterías más pequenas. As redes mesh proporcionan alta fiabilidade e un rango más extenso.

ZigBee opera nas bandas de radio industriais, científicas e médicas (ISM): 868 MHz en Europa, 915 MHz en USA e Australia e 2,4 GHz na maior parte do mundo. A velocidade de transmisión varía desde os 20 Kbps na banda de frecuencia dos 868 MHz ata os 250 Kbps na banda de frecuencia dos 2,4 GHz.

A capa de rede de ZigBee soporta nativamente tanto redes en árbore como en estrela, ademais de redes en malla xenéricas. Cada rede debe ter un dispositivo coordinador, encargado da súa creación, o control dos seus parámetros e o mantemento básico. En redes en estrela, o coordinador debe ser o nodo central. As redes en árbore e en malla permiten o uso de routers ZigBee para extender a comunicación a nivel de rede.

ZigBee está construído sobre a capa física e a de control de acceso ó medio do estándar IEEE 802.15.4-2003 para redes WPAN de baixo taxa de transmisión. A especificación pretende completar o estándar engadindo catro compoñentes principais: a capa de rede, a capa de aplicación, os denominados *ZigBee device objects* (ZDOs) e os *manufacturer-defined application objects* que permiten a personalización en favor da integración total.

Ademais de engadir dúas capas de rede de alto nivel para a estrutura subxacente, a mellora máis significativa é a introdución de ZDOs. Estes son os responsables dunha serie de tarefas, que inclúen o mantemento dos roles dos dispositivos, a xestión das solicitudes para participar nunha rede, a detección de dispositivos e a seguridade.

ZigBee non pretende soportar *powerline networking* pero si interactuar con el para propósitos de polo menos de medición intelixente e aplicacións intelixentes.

Porque os nodos ZigBee poden pasar de estado de reposo a estado activo en 30 ms ou menos, a latencia pode ser baixa e os dispositivos poden chegar a responder sen problemas. Se o comparamos cos retardos de Bluetooth desde estado de reposo a activo, que normalmente andan en torno ós 3 segundos, vemos unha gran diferencia. Ademais, os nodos ZigBee poden estar en modo de reposo a maior parte do tempo, polo que a media de consumo pode ser baixa, resultando nunha longa autonomía.

D.3.2. Historia

As redes de tipo ZigBee empezaron a ser concebidas sobre 1999, cando moitos instaladores se deron conta de que tanto Wi-Fi coma Bluetooth ían ser inusables

para moitas aplicacións. Máis concretamente, moitos enxeñeiros viron a necesidade de redes de radio dixital ad-hoc auto-organizadas. A necesidade real das mallas leva sendo posta en dúbida desde aquelas, dada a súa longa ausencia no mercado.

D.3.2.1. ZigBee 2004

O estándar IEEE 802.15.4-2003 completouse en maio de 2003.

No verán de 2003, *Philips Semiconductors*, un grande contribuidor ás redes malladas, cortou a súa inversión. Sen embargo, *Philips Lighting* continuou a participación de *Philips*, polo que este continúa sendo membro promotor no *ZigBee Alliance Board of Directors*.

A *ZigBee Alliance* anunciou en outubro de 2004 que a cantidade dos seus membros se dobraran e que eran xa máis de 100 compañías en 22 países. En abril de 2005 chegaron ás 150 compañías e en decembro de 2005 pasaron das 200.

A especificación de ZigBee foi ratificada o 14 de decembro de 2004. A *ZigBee Alliance* anunciou a dispoñibilidade da especificación 1.0 o 13 de xuño de 2005, coñecida como *ZigBee 2004*.

D.3.2.2. ZigBee 2006

En setembro de 2006, anunciouse a especificación *ZigBee 2006*. A pila do protocolo de 2004 está agora máis ou menos obsoleta.

A versión de 2006 reempraza principalmente a estructura *Message/Key Value Pair* usada na de 2004 cunha *cluster library*.

D.3.2.3. ZigBee PRO

En 2007 publicouse unha nova versión, de nome *ZigBee PRO*, tamén chamada *ZigBee 2007* por veces.

ZigBee PRO é totalmente compatible cos dispositivos ZigBee 2006. Un dispositivo ZigBee 2007 debe ser capaz de conectarse e operar nunha rede ZigBee 2006 e viceversa. Debido a diferencias no enrutado:

- Os dispositivos ZigBee PRO deben converterse en *ZigBee End-Devices* (ZEDs) non enrutadores, nunha rede ZigBee 2006.
- Os dispositivos ZigBee 2006 deben converterse en ZEDs nunha rede ZigBee PRO.

As aplicacións que se executan nestes dispositivos funcionan da mesma maneira, independentemente do perfil de pila que teñan por debaixo.

O primeiro *ZigBee Application Profile*, o *Home Automation*, foi publicado o 2 de novembro de 2007.

D.3.3. Usos

O protocolo ZigBee está pensado para aplicacións integradas que requieren baixo envío de datos e baixo consumo. A rede resultante usará moi pouca cantidade de enerxía. Os dispositivos individuais deben ter unha autonomía de polo menos dous anos para pasar a certificación ZigBee.

As áreas típicas de aplicación inclúen:

- Entretenimento no fogar e domótica: automatización, iluminación intelixente, control avanzado de temperatura, seguridade, audio e vídeo.
- Redes de sensores sen fíos.
- Control industrial.
- Sensores embebidos.
- Recolección de datos médicos.
- Alarmas de fumes e seguridade.
- Inmótica.

D.4. Estándar e perfís

A *ZigBee Alliance* é un grupo de compañías que manteñen e publican o estándar ZigBee. O termo ZigBee é unha marca rexistrada deste grupo, non só un estándar técnico. A Alliance publica perfís de aplicación que permiten a moitos vendedores OEM crear produtos interoperables. A relación entre IEEE 802.15.4 e ZigBee é similar á que hai entre IEEE 802.11 e a *Wi-Fi Alliance*.

D.4.1. Licencia

Para propósitos non comerciais, a especificación ZigBee é gratuíta para todos os públicos. A membresía de entrada á *ZigBee Alliance*, chamada *Adopter*, ofrece o acceso ás especificacións todavía non publicadas e o permiso para crear produtos comerciais empregando as especificacións.

D.4.2. Perfís de aplicación

- Especificacións publicadas:
 - ZigBee Home Automation 1.2
 - ZigBee Smart Energy 1.1b
 - ZigBee Telecommunication Services 1.0
 - ZigBee Health Care 1.0
 - ZigBee RF4CE – Remote Control 1.0
 - ZigBee RF4CE – Input Device 1.0
 - ZigBee Light Link 1.0
 - ZigBee IP 1.0
 - ZigBee Building Automation 1.0
 - ZigBee Gateway 1.0
 - ZigBee Green Power 1.0 as optional feature of ZigBee 2012
- Especificacións en desenvolvemento:

- ZigBee Smart Energy 2.0
- ZigBee Retail Services
- ZigBee Smart Energy 1.2/1.3
- ZigBee Light Link 1.1
- igBee Home Automation 1.3

D.5. Hardware de comunicacóns RF

O deseño do radio usado por ZigBee foi cuidadosamente optimizado para o baixo custo en produción a gran escala. Ten só unhas poucas fases analóxicas e emprega circuitos dixitais onde é posible.

Aínda que os propios radios son baratos, o *ZigBee Qualification Process* implica unha validación completa dos requisitos da capa física. Toda radio derivado do mesmo *validated semiconductor mask set* gozará das mesmas características RF. Unha capa física non certificada que funcione mal podería arruinar a autonomía doutros dispositivos nunha rede ZigBee. Os radios ZigBee teñen restriccións moi axustadas sobre consumo e ancho de banda.

Este estándar especifica a operación nas bandas ISM dos 2,4 GHz (todo o mundo), 915 MHz (América e Australia) e 868 MHz (Europa). Na banda dos 2,4 GHz hai 16 canles asignadas, cada unha con 5 MHz de ancho de banda. Os radios usan codificación de espectro ensanchado por secuencia directa, manexada polo fluxo dixital dentro do modulador. Nas bandas de 868 e 915 MHz úsase modulación por desprazamento de fase binaria (BPSK) e na banda dos 2,4 GHz úsase *offset quadrature phase-shift keying* (OQPSK) que transmite dous bits por símbolo.

A velocidade de transmisión de datos en bruto (RAW) por canle é de 250 Kbps na banda dos 2,4 GHz, 40 Kbps na de 915 MHz e 20 Kbps na de 868 MHz. A transferencia de datos real será inferior á taxa de bits máxima especificada debido á sobrecarga introducida pola paquetería e ós retardos de procesamento. Para transmisións en interiores na banda dos 2,4 GHz o alcance pode ser de 10

a 20 metros, dependendo dos materiais de construcción, o número de paredes a traspasar e da potencia legal de saída permitida. En exteriores e con visión directa, o alcance pode ser de ata 1500 metros dependendo da potencia de saída e das condicións ambientais. A potencia de saída dos radios é normalmente de 0-20 dBm (1-100 mW).

D.6. Tipos de dispositivos e modos de operación

Os dispositivos ZigBee poden ser de tres tipos:

- *ZigBee Coordinator (ZC)*: o dispositivo más completo, o *Coordinator*, conforma a raíz da árbore de rede, puidendo facer de ponte a outras redes. Hai exactamente un ZC en cada rede que é o que creou a rede orixinalmente. Almacena información sobre a rede, incluíndo o actuar coma *Trust Center* e repositorio de claves de seguridade.
- *ZigBee Router (ZR)*: ademais de executar funcións de aplicacións, un ZR pode actuar como intermediario pasando datos entre dispositivos.
- *ZigBee End Device (ZED)*: ten a funcionalidade xusta para comunicarse co nodo pai (a ZC or a ZR), pero non pode transmitir datos doutros dispositivos. Esta relación permite que o nodo esté durmido unha cantidade significativa do tempo, alongando a autonomía do dispositivo. Un ZED require a menor cantidade de memoria e, polo tanto é más barato de fabricar ca un ZR ou un ZC.

O protocolo ZigBee soporta tanto redes con ou sen baliza. Nas redes sen baliza, emprégase un mecanismo de acceso á canle de tipo CSMA/CA sen slot. Neste tipo de redes, os ZR teñen tipicamente os seus receptores sempre activos, o que require dunha fonte de alimentación máis potente. Sen embargo, isto permite misturar nunha rede dispositivos heteroxéneos, onde uns reciben continuamente mentres outros transmiten só cando detectan un estímulo externo.

Nas redes con baliza, os nodos especiais da rede (os ZR) transmiten balizas periodicamente para confirmar a presencia doutros nodos. Os nodos poden durmir entre balizas, extendendo a súa autonomía. O intervalo entre balizas depende

da velocidade de transmisión. Estará entre 15,36 ms e 251,65824 segundos a 250 Kbps, entre 24 ms e 393,216 segundos a 40 Kbps e entre 48 ms e 786,432 a 20 Kbps. Sen embargo, intervalos moi longos requiren alta precisión á hora de medilos, o que pode entrar en conflicto cos baixos custes de produción.

En xeral, o protocolo ZigBee minimizan o tempo que o radio está encendido e, por tanto, o consumo. En redes con balizas, os nodos só necesitan estar activos cando se transmite unha baliza. En redes sen balizas, o consumo é asimétrico: algúns dispositivos están sempre activos mentres outros están a maior parte do tempo durmindo.

Excepto o *Smart Energy Profile 2.0*, os dispositivos ZigBee teñen que cumplir co protocolo estándar 802.15.4-2003 *Low-Rate Wireless Personal Area Network* (LR-WPAN). O estándar especifica as capas máis baixas do protocolo: a capa física (PHY) e a parte do control de acceso ó medio da capa de enlace de datos (DLL). O modo básico de acceso á canle é o *carrier sense, multiple access/collision avoidance* (CSMA/CA). É dicir, comproban que ninguén esté transmitindo antes de comenzar a transmitir, excepto en tres casos:

- As balizas envíanse en intervalos fixos, polo que non usan CSMA.
- Os ACKs tampouco usan CSMA.
- As redes con balizas con requisitos de tempo real de baixa latencia usan *Guaranteed Time Slots* (GTS), o cal, por definición, non usa CSMA.

D.7. Software

O software está deseñado para ser desenvolvido en procesadores pequenos e baratos.

D.7.1. Capa de rede

As funcións principais da capa de rede son habilitar o correcto uso da subcapa MAC e proporcionar unha interface adecuada para ser usada pola capa superior, a capa de aplicación. As súas capacidades e estructura son as típicas asociadas a

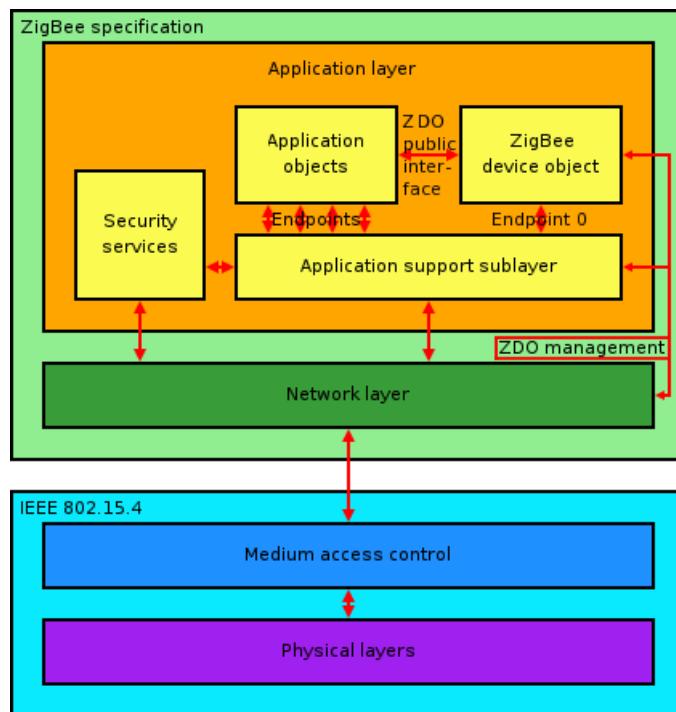


Figura D.3: ZigBee: pila do protocolo [82].

outras capas de rede, incluíndo o enrutamento.

Por unha banda, a entidade de datos crea e xestiona unidades de datos da capa de rede da carga útil da capa de aplicación e realiza o enrutamento de acordo coa topoloxía actual.

Por outra, está o control de capa, que se usa para manexar a configuración de novos dispositivos e estableces novas redes: pode determinar se un dispositivo pertence á rede e detectar novos veciños e routers. O control tamén pode detectar a presenza dun receptor, o que permite comunicación directa e sincronización da MAC.

O protocolo de enrutamento usado pola capa de rede é o *AODV*. Coa fin de atopar o dispositivo de destino, transmite unha petición de ruta a todos os seus veciños e estes ós seus ata que chega ó seu destino. Unha vez que alcanza o destino, envía unha resposta de ruta a través dunha transmisión unicast seguindo a ruta con menor custo ata a fonte. Unha vez que a fonte recibe a resposta, esta

actualiza a súa táboa de enrutamento para a dirección de destino co seguinte salto na ruta de acceso e o custe da ruta.

D.7.2. Capa de aplicación

A capa de aplicación é a capa de máis alto nivel definida pola especificación e é a interface do sistema ZigBee que empregarán os seus usuarios finais. Comprende a maioría dos compoñentes engadidos pola especificación ZigBee: tanto os ZDO coma os seus procedementos de xestión, xunto cos obxectos de aplicación definidos polo fabricante, son considerados parte desta capa.

D.7.3. Compoñentes principais

O ZDO é o responsable de definir o rol dun dispositivo coma coordinador ou dispositivo final, pero tamén de detectar os novos dispositivos da rede a distancia dun salto e identificar os servizos que ofrecen. É entón cando pode pasar a establecer enlaces seguros con dispositivos externos e responder adequadamente ás peticións de enlace.

A subcapa de soporte de aplicación (APS) é o outro compoñente estándar principal da capa e, como tal, ofrece unha interface ben definida e uns servizos de control. Funciona coma ponte entre a capa de rede e os outros compoñentes da capa de aplicación: mantén actualizadas as táboas de enlace en forma de base de datos, que poden ser usadas para atopar dispositivos apropiados dependendo dos servizos que sexan necesarios e do que ofrecen os diferentes dispositivos. Como a unión entre as dúas capas especificadas, tamén enruta a través das capas da pila do protocolo.

D.7.4. Modelos de comunicación

Unha aplicación pode consistir en comunicar obxectos que cooperan para realizar as tarefas desexadas. O foco de ZigBee é distribuír o traballo entre varios dispositivos diferentes que forman parte dunha rede ZigBee.

A colección de obxectos que forman a rede comunicanse empregando as facilidades que ofrece APS, supervisados por interfaces ZDO. O servizo de datos

da capa de aplicación segue unha estructura típica de *request-confirm/indication-response*. Dentro dun único dispositivo poden existir ata 240 obxectos de aplicación, numerados dentro do rango 1-240. O 0 está reservado para interface de datos do ZDO e o 255 para o broadcast; o rango 241-254 está reservado para usos futuros.

Hai dous servizos dispoñibles para ser usados polos obxectos de aplicación:

- O servizo *key-value pair* (KVP) está destinado a propósitos de configuración. Habilita a descripción, petición e modificación de atributos de obxecto a través dunha simple interface baseada en funcións getters/setters e de eventos, algunas das cales permiten unha petición por resposta. A configuración emprega XML comprimido (pódese usar tamén XML sen compresión) para proporcionar unha solución adaptable e elegante.
- O servizo de mensaxes está deseñado para ofrecer unha aproximación xeral para o tratamiento da información, evitando a necesidade de adaptar os protocolos de aplicación e a potencial sobrecarga na que incorre o KVP. Permite que payloads arbitrarios sexan transmitidos en frames APS.

O direccionamento é tamén parte da capa de aplicación. Un nodo de rede componse dun radio transceptor conforme á especificación 802.15.4 e unha ou máis descripcións de dispositivo (basicamente coleccións de atributos que poden ser consultados ou configurados, ou monitorizados a través de eventos). O transceptor é a base para o direccionamento, e os dispositivos dentro dun nodo especificánsen mediante un identificador de punto final no rango 1-240.

D.7.5. Comunicación e detección de dispositivos

Para que as aplicacións poidan comunicarse, os dispositivos que comprenden deben usar un protocolo de aplicación común (tipos de mensaxes, formatos e demais); estes conxuntos de convencións agrúpanse en perfís. Ademais, o enlazado decídese emparellando os identificadores de clúster de entrada e saída, únicos dentro do contexto dun perfil dado e asociados a un fluxo de datos entrante ou saínte dun dispositivo. As táboas de enlazado conteñen pares de orixe e destino.

En función da información dispoñible, a detección de dispositivos pode seguir diferentes métodos. Cando se coñece a dirección de rede, a dirección IEEE pode ser solicitada mediante comunicación unicast. En caso contrario, as peticións faranse mediante broadcast (a dirección IEEE será parte do payload de resposta). Os dispositivos finais responderán simplemente coa dirección solicitada, mentres un coordinador de rede ou router enviará tamén as direccións de todos os dispositivos asociados con el.

Este protocolo de detección extendida permite os dispositivos externos separan sobre os dispositivos dunha rede e os servizos que ofrecen, facilitando que os dispositivos finais dessa rede poidan informar cando son solicitados polo dispositivo detector (o cal obtivo previamente as súas direccións). Os servizos que coincidan poden ser usados tamén.

O uso de identificadores de clúster reforza o enlace de entidades complementarias por medio de táboas de enlace, que son mantidas polos coordinadores ZigBee, dado que as táboas deben estar sempre dispoñibles dentro dunha rede e os coordinadores son os más propensos a ter unha fonte de alimentación permanente. As copias de seguridade, xestionadas por capas de alto nivel, poden ser necesarias para algunha aplicación.

A comunicación pode ocorrer logo da asociación. O direccionamento directo usa tanto a dirección do radio coma o identificador do punto final, mentres que o direccionamento indirecto usa todo campo relevante (dirección, punto final, clúster e atributo) que precisan de ser enviados ó coordinador da rede, o cal mantén asociacións e peticións de traducción para a comunicación. O direccionamento indirecto é particularmente útil para manter algúns dispositivos moi sinxelos e minimizar a súa necesidade de almacenamento. Ademais destes dous métodos, está permitido facer broadcast a todos os puntos finais dun dispositivo e o direccionamento en grupo úsase para comunicarse con grupos de puntos finais pertencentes a un conxunto de dispositivos.

D.8. Servizos de seguridade

Como unha das súas características definidoras, ZigBee ofrece facilidades para a realización de comunicacións seguras, protexendo o establecemento e transporte de claves criptográficas, frames de cifrado e dispositivos de control. Baséase no framework de seguridade básico definido en IEEE 802.15.4. Esta parte da arquitectura depende da correcta xestión de claves simétricas e da correcta implementación dos métodos e políticas de seguridade.

D.8.1. Modelo de seguridade básico

O mecanismo básico para asegurar a confidencialidade é a adecuada protección de todo o material relacionado coas claves. A confianza debe ser asumida na instalación inicial das claves, así coma no procesamento da información de seguridade. Para que unha implementación funcione a nivel global, asúmese que o seu comportamento é conforme ó especificado.

As claves son a pedra angular da arquitectura de seguridade; como tal a súa protección é de suma importancia, e as claves nunca deben ser transportadas a través dunha canle insegura. Unha excepción momentánea a esta regra ocorre durante a fase inicial de adición á rede dun dispositivo non configurado anteriormente. O modelo de rede de ZigBee debe ter especial coidado das consideracións de seguridade, como rede ad hoc debe ser fisicamente accesible a dispositivos externos pero o ambiente de traballo particular non pode ser predito; de igual xeito, as distintas aplicacións correndo simultaneamente e utilizando o mesmo transceptor para comunicarse deben ser mutuamente confiables: por razóns de custo, o modelo asume que non hai un firewall entre entidades de nivel de aplicación.

Dentro da pila do protocolo, as diferentes capas de rede non están criptográficamente separadas, polo que son necesarias políticas de acceso e se asume un deseño correcto. O modelo de confianza aberto dentro dun dispositivo permite compartir claves, o que reduce os custos notablemente. Con todo, a capa que crea un frame é responsable da súa seguridade. Se puidesen existir dispositivos maliciosos, todo payload da capa de rede debe ser cifrado, de maneira que todo tráfico non autorizado poida ser inmediatamente rexeitado. A excepción, unha

vez máis, é a transmisión da clave de rede (que confire unha capa de seguridade unificada á rede) a un novo dispositivo de conexión.

D.8.2. Arquitectura de seguridade

ZigBee usa claves de 128 bits para implementar os seus mecanismos de seguridade. Unha clave pode ser asociada a unha rede (sendo usable por ambas capas ZigBee e a subcapa MAC) ou a un enlace, adquirida a través dunha pre-instalación, acordo ou transporte. O establecemento de claves de enlace baséase nunha clave mestra que controla a correspondencia entre claves de enlace. En último caso, alomenos a clave mestra de inicio debe ser obtida a través dun medio seguro (transporte ou pre-instalación), dado que a seguridade de toda a rede depende dela. As claves mestras e de enlace só son visibles pola capa de aplicación. Diferentes servizos usan diferentes variacións de sentido único da clave de enlace a fin de evitar perdas e riscos de seguridade.

A distribución de claves é unha das funcións más importantes da rede. Unha rede segura designará un dispositivo especial no que os demás dispositivos confiarán para a distribución de claves seguras: o *Trust Center*. Idealmente, os dispositivos terán a dirección do Trust Center e a clave mestra inicial precargadas; se se permite un momento de vulnerabilidade, será enviada como se describiu anteriormente. As aplicacións típicas sen necesidades especiais de seguridade usarán unha clave de rede proporcionada polo Trust Center (a través da canle insegura inicial) para comunicarse.

Así, o Trust Center mantén a clave de rede e proporciona seguridade punto a punto. Os dispositivos só aceptarán comunicacións orixinadas a partir dunha clave proporcionada polo Trust Center, excepto para a clave mestra inicial. A arquitectura de seguridade está distribuída entre as capas de rede como segue:

- A subcapa MAC é capaz de facer comunicacións fiables dun só salto. Como regra xeral, o nivel de seguridade a empregar será especificado polas capas superiores.
 - A capa de rede controla o enrutamento, o procesamento das mensaxes recibidas e o ser capaz de facer broadcast das peticións. Os frames de saída
-

usarán a clave adecuada acorde á ruta, se está dispoñible; noutro caso, usarase a clave de rede para protexer o payload de dispositivos externos.

- A capa de aplicación ofrece servizos de establecemento de claves e de transporte tanto ó ZDO como ás aplicacíons. É tamén responsable da propagación a través da rede dos cambios nos dispositivos que a componen, que poden ter orixe nos propios dispositivos (por exemplo, un simple cambio de estado) ou no Trust Manager (que pode informar á rede de que un certo dispositivo vai ser eliminado da mesma). Tamén enruta as peticións desde os dispositivos ó Trust Center e as renovacións da clave de rede desde o Trust Center a tódolos dispositivos. Ademais disto, o ZDO mantén as políticas de seguridade do dispositivo.

A infrastructura de niveles de seguridade está baseada en CCM*, que engade funcionalidades de cifrado e integridade a CCM.

Apéndice E

Fritzing

NESTE apéndice exponse todo o relacionado coa ferramenta de deseño hardware *Fritzing*.

Fritzing [54] é unha iniciativa software open source que busca axudar a deseñadores e artistas a pasar de prototipos a productos finais. Foi desenvolvido pola *University of Applied Sciences of Potsdam*.

E.1. Obxectivos

O software creouse seguindo o espírito de Processing e Arduino e permite que un deseñador, artista, invertigador ou afeccionado documente os seus prototipos baseados en Arduino e cree un esquema dun PCB para a súa fabricación. O sitio web [54] permite compartir e discutir borradores e experiencias así coma reducir os costes de fabricación.

Fritzing pode ser visto como unha ferramenta de automatizado de deseño electrónico (EDA) para non enxeñeiros.

As imaxes dos componentes electrónicos distribúense baixo licencia CC-BY-SA, que será tamén a licencia de calquera vista do resultado xerado.

E.2. Interface de usuario

Fritzing presenta unha interface fácil de usar para un fluxo de traballo rápido e fácil.

E.2.1. Seccións

Consta principalmente de catro seccións (figura E.1):

- Área de traballo: é a zona onde se montará o circuíto.
- Partes ou pezas: amosa a base de datos de pezas dispoñibles.
- Inspector de peza: amosa os datos concretos dunha peza.
- Navegador de vistas: permite cambiar entre as tres vistas principais.

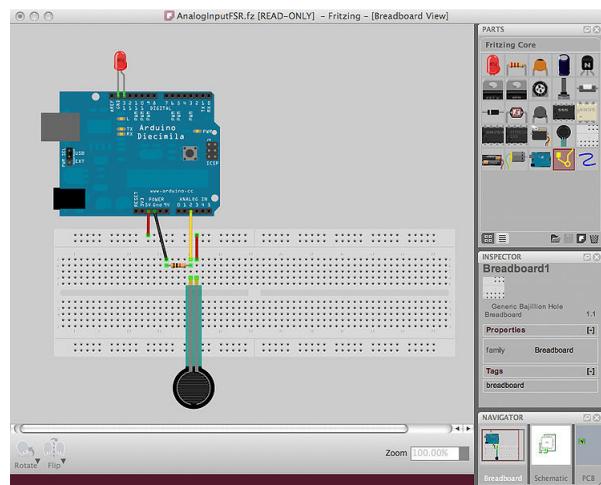


Figura E.1: Fritzing: vista de prototipado.

E.2.2. Vistas

Consta de tres vistas principais:

- Vista de prototipado: amosa a vista real dos componentes.

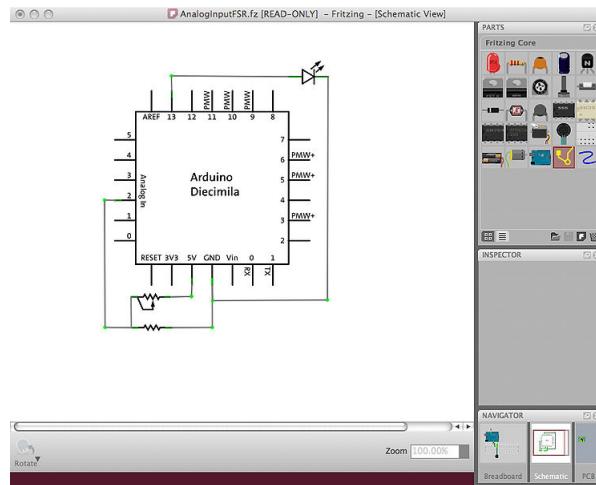


Figura E.2: Fritzing: vista esquemática.

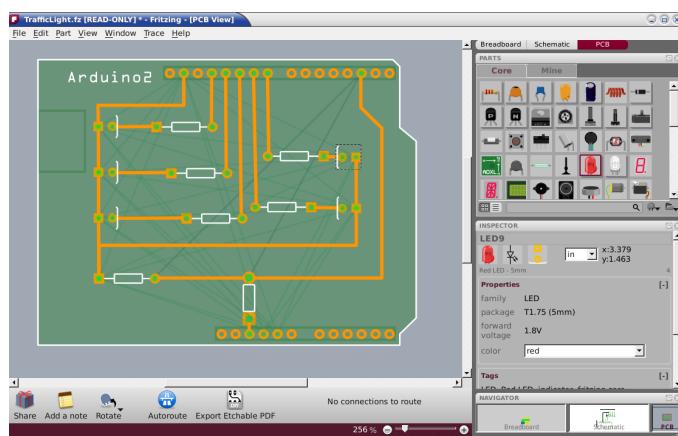


Figura E.3: Fritzing: vista de PCB.

- Vista esquemática: amosa o esquema de conexión.
- Vista de PCB: amosa cómo quedaría o PCB.

Apéndice F

Glosario de acrónimos

BDD *Behavior-Driven Development.*

BLE *Bluetooth Low Energy.*

DIN *Deutsches Institut für Normung.*

FTDI *Future Technology Devices International.*

GUI *Graphical User Interface.*

I2C *Inter-Integrated Circuit.*

IDE *Integrated Development Environment.*

ISM *Industrial, Scientific and Medical.*

JSON *JavaScript Object Notation.*

JST *Japan Solderless Terminal.*

LSB *Least Significant Bit.*

MIDI *Musical Instrument Digital Interface.*

OSC *Open Sound Control.*

PCB *Printed Circuit Board.*

PUDING *Punteiro Dixital Integramente Galego.*

SPI *Serial Peripheral Interface.*

TDD *Test-driven development.*

UART *Universal Asynchronous Receiver/Transmitter.*

USB *Universal Serial Bus.*

UX *User eXperience.*

XML *eXtensible Markup Language.*

WPAN *Wireless Personal Area Network.*

Apéndice G

Glosario de termos

Bordón Tubo sonoro que produce unha nota pedal.

Chillón Tubo sonoro que sae do fol paralelo á ronqueta, que soa na dominante da tonalidade do punteiro e na mesma oitava.

Controlador MIDI Xerador de mensaxes MIDI.

Dixitación Maneira concreta de colocación dos dedos sobre o punteiro.

Nota pedal Na música tonal, unha nota pedal é unha nota sostida ou continua que se mantén mentres o resto da melodía segue avanzando.

Fol Bolsa onde se almacena o aire que fai soar a gaita.

Gaita Instrumento musical de vento que consta dun fol e dun ou máis tubos sonoros.

Punteiro Tubo cónico que sae polo frontal do fol e que é o que produce a melodía principal en base á colocación dos dedos.

Ronco Tubo sonoro que se apoia sobre o ombro do gaiteiro, que soa na mesma tonalidade que o punteiro, pero dúas oitavas más grave.

Ronqueta Tubo sonoro que se apoia sobre o antebrazo derecho do gaiteiro, que soa na mesma tonalidade que o punteiro, pero unha oitava más grave.

Secuenciador MIDI Gravador de mensaxes MIDI.

Sintetizador MIDI Xerador de son a partir de mensaxes MIDI.

Bibliografía

- [1] Digi.
Xbee.
<http://www.digi.com/xbee>.
- [2] Wikipedia.
Gaita galega.
http://gl.wikipedia.org/wiki/Gaita_galega, 2013.
- [3] Afonso X O Sabio.
Cantigas de santa maría.
http://gl.wikipedia.org/wiki/Cantigas_de_Santa_Mar%C3%ADa, s.
XIII.
- [4] Asociación de Gaiteiros Galegos.
A casa da gaita.
http://www.gaiteirosgalegos.com/index.php?option=com_content&view=article&id=1835, 2013.
- [5] Bruno Villamor Gay.
Cadernos de Gaita, Caderno 15: Temperando.
Dos Acordes, 2009.
- [6] Pablo Carpintero Arias.
Os instrumentos musicais na tradición galega.
<http://www.consellodacultura.org/asg/instrumentos/clasificacion/os-aerofonos/aerofonos-de-lingueta/aerofonos-de-lingueta-dupla-dotados-de-fol-as-gaitas-de-fol>,
2010.
- [7] Bruno Villamor Gay.

- Cadernos de Gaita, Caderno 18: Temperando.*
Dos Acordes, 2010.
- [8] Wikipedia.
Midi - musical instrument digital interface.
http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface, 2013.
- [9] Wikipedia.
Osc - open sound control.
http://en.wikipedia.org/wiki/Open_Sound_Control, 2013.
- [10] Ask Audio.
Midi 2.0? next steps and midi alternatives for musicians.
<https://ask.audio/articles/midi-20-next-steps-and-midi-alternatives-for-musicians>
2013.
- [11] Nik Reiman.
A viable alternative to midi?
<https://stackoverflow.com/a/8962034>, 2012.
- [12] José J. Presedo.
Master gaita.
<http://www.arrakis.es/~jpresedo/midig/mggal.html>.
- [13] Anders Fagerstrom.
Technopipes.
<http://www.fagerstrom.com/technopipes/index.html>.
- [14] Rui Cardoso.
epipe.
<http://digitalelectronica.com.sapo.pt/ePipe.htm>.
- [15] José Ángel Hevia Velasco.
Hevia electronic bagpipes.
<http://heviaelectronicbagpipes.com>.
- [16] Santiago J. Barro Torres.
Diseño e implementación de un sistema autónomo basado en controlador midi para la interpretación de gaita gallega.
http://kmelot.biblioteca.udc.es/record=b1408588~S1*gag.
-

- [17] Dr. Javier Andrade Garda.
Temario da materia Enxeñería do Software.
Facultade de Informática - Universidade da Coruña, 2010.
 - [18] OpenProj.
Openproj.
<http://sourceforge.net/projects/openproj>, 2008.
 - [19] GNOME.
Planner.
<https://wiki.gnome.org/action/show/Apps/Planner?action=show&redirect=Planner>, 2011.
 - [20] GanttProject.
Ganttpoint.
<http://www.ganttproject.biz>.
 - [21] Igalia S.L.
Libreplan.
<http://www.libreplan.com>.
 - [22] Microsoft.
Project.
<http://office.microsoft.com/es-es/project>.
 - [23] Proxecto Puding.
Estudio de viabilidade.
<http://enquisa.proxecto-puding.org>.
 - [24] Wikipedia.
Problema de fermi.
http://es.wikipedia.org/wiki/Problema_de_Fermi.
 - [25] Asociación de Gaiteiros Galegos.
A asociación de tódolos músicos tradicionais de galicia.
<http://www.gaiteirosgalegos.com>.
 - [26] Pedro Morales Vallejo.
Estadística aplicada a las ciencias sociales. tamaño necesario de la muestra:
¿cuántos sujetos necesitamos?
-

<http://www.upcomillas.es/personal/peter/investigacion/Tama%C3%B3nMuestra.pdf>, 2012.

[27] Lis Latas.

Obradoiro de gaitas lis latas.
<http://www.lislatas.com>.

[28] Pepe Vaamonde.

Pepe vaamonde grupo.
<http://www.pepevaamondegrupo.com>.

[29] Arduino.

Arduino.
<http://arduino.cc>.

[30] BeagleBoard Foundation.

Beaglebone.
<http://beagleboard.org>.

[31] PandaBoard.

Pandaboard.
<http://pandaboard.org>.

[32] Atmel.

Qtouch.
<http://www.atmel.com/products/touchsolutions/touchsoftware/default.aspx>.

[33] Raspberry Pi Foundation.

Raspberry pi.
<http://www.raspberrypi.org>.

[34] SparkFun Electronics.

Sparkfun.
<http://www.sparkfun.com>.

[35] BricoGeek.

Bricogeek.
<http://www.bricogeek.com>.

[36] Libelium.

Cooking hacks.

- [http://www.cooking-hacks.com.](http://www.cooking-hacks.com)
- [37] SparkFun.
Bmp085.
[https://www.sparkfun.com/products/9694.](https://www.sparkfun.com/products/9694)
- [38] SparkFun.
Mpr121.
[https://www.sparkfun.com/products/9695.](https://www.sparkfun.com/products/9695)
- [39] Arduino.
Arduino fio.
[http://arduino.cc/en/Main/ArduinoBoardFio.](http://arduino.cc/en/Main/ArduinoBoardFio)
- [40] Digi.
Xbee zb.
[http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-zb-module.](http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-zb-module)
- [41] Wikipedia.
Zigbee.
[http://es.wikipedia.org/wiki/ZigBee.](http://es.wikipedia.org/wiki/ZigBee)
- [42] Arduino.
Arduino bluetooth.
[http://arduino.cc/en/Main/ArduinoBoardBluetooth.](http://arduino.cc/en/Main/ArduinoBoardBluetooth)
- [43] Bluetooth.
Bluetooth.
[http://en.wikipedia.org/wiki/Bluetooth.](http://en.wikipedia.org/wiki/Bluetooth)
- [44] EMFWISE.
Wifi.
[http://emfwise.com/distance.php#smartmeter.](http://emfwise.com/distance.php#smartmeter)
- [45] STG.
Comparativa zigbee.
[http://www.stg.com/wireless/ZigBee_comp.html.](http://www.stg.com/wireless/ZigBee_comp.html)
- [46] BricoGeek.
Comunicación xbee.
-

<http://www.bricogEEK.com/shop/modulos-radiofrecuencia/411-xbee-2mw-con-antena-chip-serie-2-zb.html>.

[47] SparkFun.

Midi breakout.

<https://www.sparkfun.com/products/retired/9598>.

[48] SparkFun.

Xbee explorer usb.

<https://www.sparkfun.com/products/8687>.

[49] Arduino.

Arduino wireless sd shield.

<http://arduino.cc/en/Main/ArduinoWirelessShield>.

[50] Arduino.

Arduino wireless proto shield.

<http://arduino.cc/en/Main/ArduinoWirelessProtoShield>.

[51] 4D Systems.

Microdrive g1.

<http://old.4dsystems.com.au/prod.php?id=22>.

[52] BricoGeek.

Lipo 850 mah.

<http://www.bricogEEK.com/shop/baterias-lipo/342-bateria-lipo-850mah.html>.

[53] CadSoft.

Eagle.

<http://www.cadsoftusa.com>.

[54] Friends of Fritzing Foundation.

Fritzing.

<http://fritzing.org>.

[55] Friends of Fritzing Foundation.

Fío de contribucións a fritzing.

<http://code.google.com/p/fritzing/issues/detail?id=875#c208>.

[56] Wikipedia.

Bdd.

- [https://en.wikipedia.org/wiki/Behavior-driven_development.](https://en.wikipedia.org/wiki/Behavior-driven_development)
- [57] Wikipedia.
Tdd.
[https://en.wikipedia.org/wiki/Test-driven_development.](https://en.wikipedia.org/wiki/Test-driven_development)
- [58] FortySevenEffects.
Midi for arduino.
<https://playground.arduino.cc/Main/MIDILibrary>.
- [59] Interactive Matter.
ajson.
<https://github.com/interactive-matter/aJson>.
- [60] Benoît Blanchon.
Arduino json.
<https://github.com/bblanchon/ArduinoJson>.
- [61] Google.
Guice.
<https://github.com/google/guice>.
- [62] Google.
Guava.
<https://github.com/google/guava>.
- [63] GNU.
Rxtx.
<http://rxtx.qbang.org>.
- [64] Alexey Sokolov.
Java simple serial connector.
<http://rxtx.qbang.org>.
- [65] Google.
Gson.
<https://github.com/google/gson>.
- [66] Wikipedia.
Timidity.
<https://en.wikipedia.org/wiki/Timidity%2B%2B>.
-

- [67] FluidSynth.
Fluidsynth.
<http://www.fluidsynth.org>.
- [68] Wikipedia.
Afinación temperada.
https://en.wikipedia.org/wiki/Equal_temperament.
- [69] Wikipedia.
Afinación natural.
https://en.wikipedia.org/wiki/Just_intonation.
- [70] Escuela Universitaria de Música Universidad de la República de Uruguay.
Escalas.
<http://www.eumus.edu.uy/eme/ensenanza//acustica/apuntes/afyesc2/escalas.html>.
- [71] Mockito.
Mockito.
<https://site.mockito.org>.
- [72] Wikipedia.
Gherkin.
[https://en.wikipedia.org/wiki/Cucumber_\(software\)#Gherkin_language](https://en.wikipedia.org/wiki/Cucumber_(software)#Gherkin_language).
- [73] Doxygen.
Doxygen.
<http://www.doxygen.nl>.
- [74] Hispasonic.
Ft232rl.
<http://www.hispasonic.com/foros/usa-arduino-para-comunicarse-ft232/324759>.
- [75] MorecatLab.
Moco - midi firmware for arduino.
<http://morecatlab.akiba.coocan.jp/lab/index.php/arduino/midi-firmware-for-arduino-uno-moco/?lang=en>.
- [76] Arduino.
-

- Arduino duemilanove.
[http://arduino.cc/en/Main/ArduinoBoardDuemilanove.](http://arduino.cc/en/Main/ArduinoBoardDuemilanove)
- [77] Arduino.
Arduino uno.
[http://arduino.cc/en/Main/ArduinoBoardUno.](http://arduino.cc/en/Main/ArduinoBoardUno)
- [78] Arduino.
Arduino leonardo.
[http://arduino.cc/en/Main/ArduinoBoardLeonardo.](http://arduino.cc/en/Main/ArduinoBoardLeonardo)
- [79] TLAANA.
Titorial de creación de componentes para fritzing.
<http://tlaana.com/recursos/crea-tus-propios-componentes-para-fritzing-fzp-1>
- [80] Friends of Fritzing Foundation.
Contribución ó proxecto fritzing.
<http://code.google.com/p/fritzing/issues/detail?id=875#c371>.
- [81] Wikipedia.
Arduino.
<http://gl.wikipedia.org/wiki/Arduino>, 2013.
- [82] Wikipedia.
Zigbee protocol stack.
[http://es.wikipedia.org/wiki/ZigBee_\(especificaci%C3%B3n\)](http://es.wikipedia.org/wiki/ZigBee_(especificaci%C3%B3n)).
-