Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage. [http://researchspace.auckland.ac.nz/feedback](http://researchspace.auckland.ac.nz/feedback)

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form and Deposit Licence.
BioSignalML: An Abstract Model for Physiological Time-series Data

David Brooks

Supervised by Dr M. R. Titchener and Professor B. H. Smaill.


Auckland Bioengineering Institute
The University of Auckland
New Zealand

May 2013
Abstract

This thesis proposes a new standard framework, BioSignalML, for describing physiological time-series data, in order to address some of the challenges created by the diverse range of formats used for biosignal storage and exchange. I argue that the main cause of difficulties for researchers, wanting to share and exchange biosignal data, is a lack of standardisation as to how metadata is represented and assigned meaning, even for common attributes, and not due to differing data formats. The BioSignalML model seeks to address the lack of metadata standardisation and considers data and metadata to be of equal importance.

Instead of specifying another new storage format, BioSignalML uses open data and metadata standards, in particular those from the Semantic Web, to describe signals in their existing formats. An abstract model and a new ontology, the BioSignalML Ontology, provide a shared framework for biosignals, allowing their information to be extended, unified, and interlinked with other resources on the Web. This approach, of applying Semantic Web technology to biosignals in a general way, is believed to be novel.

The BioSignalML software library allows applications to create and access signal data and metadata in a format independent way. This library forms the basis of a web-accessible biosignal repository that allows standard web applications, including Semantic Web tools, to be used to query, browse, annotate and process data in signal collections. All software is freely available as an open-source resource.

The utility of BioSignalML is demonstrated by its application in three separate areas — to enhance a public repository of physiological signal recordings; to facili-
tate the integration of biosignal data with physiological modelling applications; and to manage polysomnography and respiratory flow data used in product research and development. In all these applications, users benefit from improved query, annotation, and data services.

BioSignalML is designed to facilitate data integration and provide metadata consistency, both within and between research groups, and across a wide range of research domains, in a way that allows for future extension.
Acknowledgments

I am extremely grateful for the generous funding provided by the Auckland Bioengineering Institute in the form of a PhD studentship, which allowed me the time to focus on this work, and to Auckland University’s post-graduate student support funding, which provided the opportunity to present my work internationally and to spend time at the Massachusetts Institute of Technology. A special thank you is due to Roger Marks and George Moodie of PhysioNet, who made me very welcome when I visited their laboratory at MIT.

My supervisors have supported and guided me in their different ways throughout the course of this thesis. I am grateful to Dr Mark Titchener, for keeping me in touch with the real world of experiments and sharing the fun of realtime signal processing, so ensuring I didn’t get too lost in abstract concepts. I am especially grateful to Professor Bruce Smaill, for his dedication to academic excellence, his patience and faith that I could write a thesis, and his skill in coaxing out the writer in me. Thank you.

The Auckland Bioengineering Institute has provided a supportive environment in which to work and share ideas. Thank you to Andre, Denis, Mike, Poul, Richard, and many others, along with a very special thank you to Professor Peter Hunter for his quiet support, confidence and words of encouragement.

My family and friends have had to endure my increasing focus on this thesis for longer than they anticipated, and I’m extremely grateful for their patience and ongoing support. Thank you to my daughter Ilona and her partner Mike for their
support, especially over these last months as this project has neared completion, and to my son Rowan, for his listening ear and sage advice whilst on his travels. Thank you to Stewart and Ra for continuing to make me very welcome in their home, and to Stewart for providing a writing space, mentoring, and ongoing companionship. Thank you to Tina for her support, and to her daughter Rose for keeping me in touch with my two-year old self. To Elly, thank you — for patience, friendship, and ongoing love and support.
# Table of Contents

1 Introduction .................................................. 1

1.1 Digital Signals ............................................. 2

1.1.1 Signal attributes and metadata ......................... 4

1.1.2 Signal processing ........................................ 6

1.1.3 Biosignal characteristics ............................... 7

1.2 Biosignal Storage Formats and Standards .................. 8

1.2.1 Waveform database format ............................... 11

1.2.2 European data format and extensions ................... 12

1.2.3 General data format ..................................... 13

1.2.4 Medical waveform format — encoding rules .............. 14

1.2.5 XML based formats ...................................... 15

1.2.6 Science data formats .................................... 16

1.2.7 Format independence ..................................... 19

1.3 The Evolving Web ........................................... 20

1.3.1 The Semantic Web ..................................... 20

1.3.2 Resources and RDF ..................................... 21

1.3.3 Linking data ............................................. 23

1.3.4 Formalising time ........................................ 24

1.3.5 Provenance and annotation ............................. 25

1.4 Modelling and Simulation ................................... 26

1.4.1 Formalising biology ..................................... 27

1.5 Difficulties .................................................... 28
## TABLE OF CONTENTS

1.5.1 Metadata consistency ........................................... 29  
1.5.2 Ontologies ......................................................... 31  
1.6 Related Work .......................................................... 32  
  1.6.1 ROISES .......................................................... 32  
  1.6.2 Physio-MIMI ..................................................... 32  
  1.6.3 The multi-scale annotation format ......................... 33  
  1.6.4 Web-based biosignal visualisation and annotation .... 33  
1.7 Thesis Overview ...................................................... 34  

2 BioSignalML ............................................................. 37  
  2.1 An Abstract Model .................................................. 38  
    2.1.1 Recording .................................................... 39  
    2.1.2 Signal ....................................................... 39  
    2.1.3 Event ........................................................ 40  
    2.1.4 Annotation ................................................... 40  
    2.1.5 Segment ...................................................... 41  
    2.1.6 Timing ........................................................ 41  
  2.2 The BioSignalML Ontology ....................................... 42  
  2.3 Realising the Model ............................................... 45  
    2.3.1 Identity ..................................................... 45  
    2.3.2 Generic metadata .......................................... 46  
    2.3.3 Specifying units of measure .............................. 46  
    2.3.4 Specifying time ........................................... 48  
    2.3.5 Measurement uncertainty ................................. 50  
    2.3.6 Annotation .................................................. 50  
    2.3.7 Provenance .................................................. 53  
  2.4 An Example .......................................................... 54  
  2.5 Summary ............................................................ 57  

3 BioSignalML Software ................................................. 61  
  3.1 The BioSignalML Repository .................................... 61
# TABLE OF CONTENTS

3.1.1 Naming of resources ........................................... 63  
3.1.2 Controlling access ............................................ 64  
3.1.3 Accessing resources .......................................... 64  
3.1.4 Accessing signal data ....................................... 66  
3.1.5 Web browser endpoint ....................................... 68  
3.1.6 SPARQL endpoint ............................................. 69  
3.1.7 Recording provenance ........................................ 70  
3.2 BioSignalML Library .............................................. 71  
3.2.1 Metadata mapping ............................................ 72  
3.2.2 Abstract Model implementation .............................. 73  
3.2.3 Data abstraction .............................................. 74  
3.2.4 Repository support .......................................... 74  
3.2.5 Accessing remote repositories ............................... 74  
3.3 HDF5 Signal Storage ............................................ 75  
3.3.1 Performance .................................................. 76  
3.4 System Design and Implementation ............................ 77  
3.4.1 Web based .................................................... 77  
3.4.2 Developed in Python ......................................... 78  
3.4.3 Open-source packages used .................................. 79  
3.5 Summary .......................................................... 81  

4 Applying BioSignalML to PhysioBank ............................ 83  
4.1 PhysioBank ....................................................... 83  
4.1.1 Limitations .................................................... 85  
4.2 Software Tools ................................................... 86  
4.2.1 Querying a recording collection ............................. 88  
4.2.2 Browsing a set of recordings ................................. 89  
4.2.3 Signal visualisation .......................................... 90  
4.2.4 Annotation and curation .................................... 94  
4.3 The Conversion Process ......................................... 94  
4.3.1 MIT-BIH Arrhythmia Database ............................... 97
# TABLE OF CONTENTS

4.3.2 MGH/MF Waveform Database ......................... 102
4.3.3 Gait Dynamics in Neuro-Degenerative Disease Database ... 103
4.4 Results and Discussion ........................................... 107
  4.4.1 Metadata inconsistencies .................................... 107
  4.4.2 Other PhysioBank databases ................................ 109
  4.4.3 PhysioBank Web presence .................................... 110
4.5 Summary ............................................................... 110

5 Using BioSignalML for Data ........................................ 113
  5.1 Introduction ....................................................... 113
  5.2 Modelling and Simulation ...................................... 115
    5.2.1 Calcium driven phosphorylation ......................... 116
  5.3 Data Integration .................................................. 120
    5.3.1 Bringing polysomnography data together ............... 121
    5.3.2 Testing CPAP devices ....................................... 123
    5.3.3 BioSignalML stream interface ......................... 124
  5.4 Discussion .......................................................... 126
    5.4.1 Modelling and simulation ................................ 126
    5.4.2 Data integration ............................................. 128
    5.4.3 Streaming signals and events ......................... 128
    5.4.4 Tracking software use ..................................... 129

6 Discussion and Conclusion ......................................... 131
  6.1 Limitations and Extensions .................................... 134
  6.2 Future Work ...................................................... 135
    6.2.1 PhysioBank .................................................. 135
    6.2.2 Repository deployment ..................................... 136
    6.2.3 Annotation frameworks ................................... 137
  6.3 Publications ....................................................... 137

Appendices .............................................................. 141
  A The BioSignalML Ontology ...................................... 141
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Classes</td>
<td>141</td>
</tr>
<tr>
<td>A.2 Properties</td>
<td>147</td>
</tr>
<tr>
<td>B Block Stream Format</td>
<td>153</td>
</tr>
<tr>
<td>B.1 Block Types</td>
<td>154</td>
</tr>
<tr>
<td>B.2 Version</td>
<td>156</td>
</tr>
<tr>
<td>C BioSignalML HDF5 File Layout</td>
<td>157</td>
</tr>
<tr>
<td>C.1 General Structure</td>
<td>157</td>
</tr>
<tr>
<td>C.2 Discontinuous Signals</td>
<td>161</td>
</tr>
<tr>
<td>C.3 Units of Measurement</td>
<td>162</td>
</tr>
<tr>
<td>D Software Tools</td>
<td>163</td>
</tr>
<tr>
<td>E PhysioBank Index for a MGHDB Recording</td>
<td>165</td>
</tr>
<tr>
<td>F Code Modifications for CellML Simulation</td>
<td>167</td>
</tr>
<tr>
<td>F.1 Interface Code for Python CeLED Simulation</td>
<td>168</td>
</tr>
<tr>
<td>G Breath Processing with PERTECS</td>
<td>173</td>
</tr>
<tr>
<td>Bibliography</td>
<td>177</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Typical sampling of a continuous signal. .......................... 3
1.2 Signal processing: some method or algorithm, controlled by parameters, gives rise to a dataset. .......................... 7
1.3 An RDF statement. ........................................... 21

2.1 A Recording’s temporal systems. ........................................... 42
2.2 The BioSignalML Ontology. ........................................... 44
2.3 Describing 1000Hz sampling with fully specified TimeLineMaps. ........ 49
2.4 Describing 1000Hz sampling with implicit TimeLineMaps. ........ 50
2.5 Uncertainty associated with a signal segment. ................. 51
2.6 Using Open Annotation to describe a 10 second segment of a recording. 51
2.7 BioSignalML annotation. ........................................... 52
2.8 BioSignalML markup of an EDF+ file. ........................................... 55

3.1 Repository interfaces ........................................... 62
3.2 Accessing the RDF endpoint. ........................................... 67
3.3 Browsing and annotating recordings. ........................................... 69
3.4 Browsing RDF using SNORQL. ........................................... 69
3.5 Provenance of a BioSignalML recording. ........................................... 71
3.6 Structure of BioSignalML library core. ........................................... 72
3.7 Core BioSignalML library classes. ........................................... 73
3.8 Structure of a BioSignalML HDF5 file. ........................................... 75

4.1 Conversion to BioSignalML. ........................................... 87
4.2 Simple query user interface. ........................................ 88
4.3 Browsing results from a query. .................................... 90
4.4 Browsing a collection of recordings. ............................. 91
4.5 A recording visualisation tool. ..................................... 92
4.6 Controlling what is viewed. ....................................... 93
4.7 General process to convert a PhysioBank database to BioSignalML. ......................... 98
4.8 The <mitdb/102> recording with patient metadata. ............. 100
4.9 Components of the <mitdb/102> recording. ...................... 101
4.10 Part of the <mghdb/mgh137> recording. ......................... 104
4.11 BioSignalML about the <gaitndd/als12> recording. .......... 106

5.1 A general model for using signal data. .......................... 114
5.2 An interface to detect and reconcile data attribute mismatches. ... 114
5.3 Experimentally-obtained calcium responses — reproduction of Figure 1 from Perc et al. ............ 116
5.4 Protein phosphorylation driven by external Ca^{2+} signal. .... 117
5.5 Simulation results stored in BioSignalML repository. .......... 118
5.6 Simulated active phosphorylase using experiment derived Ca^{2+} signals. .................. 119
5.7 A polysomnography recording with event annotation. ......... 122
5.8 CPAP device evaluation using recorded breathing patterns. .... 124
5.9 Interfacing between BioSignalML and data streams. ........... 126

C.1 General structure of a BioSignalML HDF5 file. ............... 158

G.1 Breath feature extraction from a flow signal using PERTECS. ... 173
List of Tables

1.1 Characteristics of common biosignals. ........................................ 7
1.2 Important biosignal formats. ..................................................... 9
1.3 Other biosignal formats, not in common usage. .......................... 10
1.4 Origins of biosignal formats referenced by the BioSig Project. ...... 10

2.1 Internet Media Types specified by BioSignalML. ................. 47
2.2 RDFS and OWL ontologies used by BioSignalML. ................. 58
2.3 Abstract Model mapped to ontologies. ................................. 59

3.1 Time, in seconds, to read data from HDF5 and WFDB formatted recordings. ................................................................. 77

D.1 Important biosignal formats. .................................................. 163
# List of Program Code Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An excerpt from the BioSignalML Ontology.</td>
<td>43</td>
</tr>
<tr>
<td>2.2</td>
<td>User specified units of measurement.</td>
<td>48</td>
</tr>
<tr>
<td>2.3</td>
<td>A recording described in BioSignalML.</td>
<td>54</td>
</tr>
<tr>
<td>2.4</td>
<td>A signal described in BioSignalML.</td>
<td>56</td>
</tr>
<tr>
<td>2.5</td>
<td>An irregular signal described in BioSignalML.</td>
<td>56</td>
</tr>
<tr>
<td>2.6</td>
<td>An event described in BioSignalML.</td>
<td>57</td>
</tr>
<tr>
<td>3.1</td>
<td>Importing and retrieving a recording using curl.</td>
<td>66</td>
</tr>
<tr>
<td>3.2</td>
<td>SPARQL query returning provenance of a recording.</td>
<td>70</td>
</tr>
<tr>
<td>3.3</td>
<td>Mapping table for AbstractObject.</td>
<td>73</td>
</tr>
<tr>
<td>4.1</td>
<td>PhysioBank index entries for mitdb/102.</td>
<td>99</td>
</tr>
<tr>
<td>B.1</td>
<td>Structure of a message block defined using EBNF.</td>
<td>153</td>
</tr>
<tr>
<td>B.2</td>
<td>Block types used by BioSignalML.</td>
<td>154</td>
</tr>
<tr>
<td>E.1</td>
<td>Master index entries for a MGHDB recording.</td>
<td>166</td>
</tr>
<tr>
<td>F.1</td>
<td>Modifications to CeLEDs generated code to use BioSignalML data.</td>
<td>168</td>
</tr>
<tr>
<td>F.2</td>
<td>Python module for CeLEDs simulations.</td>
<td>169</td>
</tr>
<tr>
<td>G.1</td>
<td>An interface specification between BioSignalML and data streams.</td>
<td>174</td>
</tr>
<tr>
<td>G.2</td>
<td>Using \texttt{PERTECS} to extract breath measures.</td>
<td>175</td>
</tr>
</tbody>
</table>
# List of Abbreviations

## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analogue to digital converter</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>BCI</td>
<td>Brain-computer interface</td>
</tr>
<tr>
<td>CeLEDS</td>
<td>CellML language export definition service</td>
</tr>
<tr>
<td>CEN</td>
<td>Comité Européen de Normalisation</td>
</tr>
<tr>
<td>CICR</td>
<td>$\text{Ca}^{2+}$-induced $\text{Ca}^{2+}$ release</td>
</tr>
<tr>
<td>CPAP</td>
<td>Continuous positive airway pressure</td>
</tr>
<tr>
<td>DCMI</td>
<td>Dublin core metadata initiative</td>
</tr>
<tr>
<td>DICOM</td>
<td>Digital imaging and communications in medicine</td>
</tr>
<tr>
<td>EBI</td>
<td>European Bioinformatics Institute</td>
</tr>
<tr>
<td>EBNF</td>
<td>Extended Backus-Naur form</td>
</tr>
<tr>
<td>EDF</td>
<td>European data format</td>
</tr>
<tr>
<td>FDA</td>
<td>Food and Drug Administration</td>
</tr>
<tr>
<td>HDF5</td>
<td>Hierarchical data format five</td>
</tr>
<tr>
<td>HL7</td>
<td>Health level seven</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext markup language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext transfer protocol</td>
</tr>
<tr>
<td>IANA</td>
<td>Internet Assigned Numbers Authority</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronic Engineers</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JSON</td>
<td>Javascript object notation</td>
</tr>
<tr>
<td>MFER</td>
<td>Medical format encoding rules</td>
</tr>
<tr>
<td>OPB</td>
<td>Ontology of physics for biology</td>
</tr>
<tr>
<td>OWL</td>
<td>Web ontology language</td>
</tr>
<tr>
<td>QName</td>
<td>Qualified name</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource description framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>Resource description framework schema</td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>SBML</td>
<td>Systems biology markup language</td>
</tr>
<tr>
<td>SD-ML</td>
<td>Simulation experiment description markup language</td>
</tr>
<tr>
<td>SNOMED</td>
<td>Systematised nomenclature of medicine</td>
</tr>
<tr>
<td>SNOMED CT</td>
<td>Systematised nomenclature of medicine clinical terms</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL protocol and RDF query language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured query language</td>
</tr>
<tr>
<td>SWRL</td>
<td>Semantic web rule language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified modelling language</td>
</tr>
<tr>
<td>UMLS</td>
<td>Unified medical language system</td>
</tr>
<tr>
<td>UOM</td>
<td>Units of measurement</td>
</tr>
<tr>
<td>UOME</td>
<td>Units of measurement expressions</td>
</tr>
<tr>
<td>URI</td>
<td>Universal resource identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Universal resource locator</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated universal time</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WFDB</td>
<td>Waveform database</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible markup language</td>
</tr>
</tbody>
</table>
### Namespace Prefixes

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>bsml</td>
<td><a href="http://www.biosignalml.org/ontologies/2011/04/biosignalml#">http://www.biosignalml.org/ontologies/2011/04/biosignalml#</a></td>
</tr>
<tr>
<td>dct</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
</tr>
<tr>
<td>drugbank</td>
<td><a href="http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/">http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/</a></td>
</tr>
<tr>
<td>oa</td>
<td><a href="http://www.w3.org/ns/openannotation/core/">http://www.w3.org/ns/openannotation/core/</a></td>
</tr>
<tr>
<td>owl</td>
<td><a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a></td>
</tr>
<tr>
<td>pbterms</td>
<td><a href="http://www.biosignalml.org/ontologies/examples/physiobank#">http://www.biosignalml.org/ontologies/examples/physiobank#</a></td>
</tr>
<tr>
<td>prv</td>
<td><a href="http://purl.org/net/provenance/ns#">http://purl.org/net/provenance/ns#</a></td>
</tr>
<tr>
<td>rdf</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
</tr>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
</tr>
<tr>
<td>time</td>
<td><a href="http://www.w3.org/2006/time#">http://www.w3.org/2006/time#</a></td>
</tr>
<tr>
<td>tl</td>
<td><a href="http://purl.org/NET/c4dm/timeline.owl#">http://purl.org/NET/c4dm/timeline.owl#</a></td>
</tr>
<tr>
<td>unit</td>
<td><a href="http://www.biosignalml.org/ontologies/examples/unit#">http://www.biosignalml.org/ontologies/examples/unit#</a></td>
</tr>
<tr>
<td>uome-core</td>
<td><a href="http://www.sbpax.org/uome/core.owl#">http://www.sbpax.org/uome/core.owl#</a></td>
</tr>
<tr>
<td>uome-list</td>
<td><a href="http://www.sbpax.org/uome/list.owl#">http://www.sbpax.org/uome/list.owl#</a></td>
</tr>
<tr>
<td>xmls</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
</tr>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This thesis is concerned with the storage and exchange of biological signals in computing systems, with a particular focus on the representation and handling of metadata.

The recording, storage, and exchange of biological signals is fundamental to physiological research. There are however many different formats in which signal data may be held, and this causes difficulties with the sharing of data and with the use of equipment and/or software, particularly if data about data, or metadata, is lost during format conversion or data exchange.

Researchers use terms and concepts from their discipline to annotate and give meaning to data. Unless a storage format has been designed specifically to hold this information, these terms and concepts will be maintained separately, often formatted for human readability. This information may be lost when data is exchanged, or if held with data, lost during format conversion. Even if electronically available, software applications may have difficulty knowing what terms mean, especially those describing datasets sourced from outside of the immediate research group.

These issues are not unique to physiological research — difficulties posed by big data are covered in publications ranging from special issues of Nature [1, 2] and Science [3] to a discussion series on New Zealand’s National Radio [4].
BioSignalML is a new framework for describing physiological time-series data designed to address issues caused by a lack of standardisation, in both biosignal storage formats and in how metadata is represented and assigned meaning.

This chapter first describes the nature of digitally sampled signals. It then reviews how biosignals are stored and exchanged before examining metadata standards developed for the World Wide Web and looking at how biosignals are used in physiological modelling and simulation. Difficulties with the use of biosignal metadata are then discussed, along with a number of challenges which this thesis attempts to resolve. Other projects that seek to address similar issues are then described.

The chapter concludes with an overview of the rest of the thesis.

1.1 Digital Signals

A \textit{signal} is any kind of measurable quantity that varies in space and/or in time, with a \textit{biosignal} generally considered to be a time-varying signal that is the direct result of some biological process, usually in the context of mammalian physiology. Although my work is based on time-varying signals, it is also applicable to other classes of signals, including those that vary in spatial and spatio-temporal dimensions. A signal may also be thought of as a field over its sampling dimension(s) or domain. The terminology in this thesis is that used for sampled digital signals rather than that used for fields over a domain.

A discrete, or digital, signal is the result of sampling some physical quantity along a continuous sampling dimension, typically as shown in Figure 1.1.

The digitisation process begins with sensing the value of some quantity using a sensor, transducer, or electrode to obtain a voltage or current. The resulting raw electrical signal typically has low magnitude, is noisy, and contains components from processes other than that of interest. Front-end processing, such as amplification,
1.1. DIGITAL SIGNALS

threshold and edge detection, and filtering, is used to clean-up the signal before using an analogue-to-digital converter (ADC) to obtain a digitised version of the original signal.

The simplest way to represent a digital signal is as a time-ordered sequence of integers (the analogue-to-digital converter’s output) along with timing and calibration information needed to reconstruct the time-series. When several signals are simultaneously recorded each channel’s sequence of data values may either be kept separately, or multiplexed with other channels’ data often in fixed duration blocks.

Signal timing is usually specified by giving the sampling rate, possibly with an offset or skew; if the sampling interval is non-uniform a separate channel is normally used for storing sample times. Calibration may be expressed in different ways: as a gain and offset; by giving the physical values associated with the converter’s minimum and maximum outputs; or by reference to a separate calibration channel containing known values. Besides timing and calibration, a signal may have other metadata directly associated with it, which may include a channel identifier, a label, its physical units, transducer or electrode characteristics, electrode position, preprocessing and filtering details, and the numerical format of data.

A recording, that is a group of signals that are captured together, may also have associated metadata. This might include the date and time of the recording session, its duration, a description, identifying codes or details (of study, investigator, subject, equipment, etc), and details of how signal channels are stored.
Recording and signal metadata may or may not be directly stored with a recording or signal. Some storage formats use a preamble at the start of the data file for metadata; other formats have a separate file; sometimes an independent configuration file is used; and sometimes no relevant metadata is directly stored and linked with data, being recorded instead in a researcher’s logbook and only used when required for analysis and presentation. Regardless though, for a given format there is agreement amongst users as to the meaning of core metadata values and annotation texts, albeit often restricted to the format’s application domain, or even just to a particular research group, especially for text-based metadata.

1.1.1 Signal attributes and metadata

Some of the attributes and properties that a signal has are of greater importance than others, as they impact the usefulness of the signal for future analysis and processing. Primary attributes include the following.

Identification

Each signal has to be identified so that reference may later be made to it. This may be explicit, using a label, identifier, or description, or implicit, such as using a signal’s position in a dataset as its identifier.

Physical units

Physical units of a measured quantity are essential to ensure correct use and interpretation of a signal. Having appropriate units allows for automated conversion between different scales and ratios for the same physical quality, for example converting a pressure signal from millimetres of mercury to millibars.
1.1. DIGITAL SIGNALS

Data attributes

A digital signal may contain a number of systemic artefacts resulting from the digitisation process, for example due to word size limitations. These might produce errors or bias in later analysis, so it is important that details of pre-processing and digital conversion are stored with a signal, including bit-resolution, signal and ADC range, and sampling rate.

Analogue-to-digital conversion, by its nature, results in timing resolution, quantisation, and linearity errors — only a finite number of bits are available to hold continuous input values. Biosignals are now usually digitised with at least 16-bit resolution, although a significant number of historically significant reference biosignals in public repositories have been recorded with a resolution of 12-bits. The amplitude of a signal with respect to the input range of the ADC is also of significance — too large a signal amplitude will result in overflow; too small an amplitude means that the full bit-resolution of the ADC is not being exploited, resulting in underflow.

Sampling rate, filtering and timing

The Whittaker–Kotelnikov–Shannon Sampling Theorem [5] states that a continuous band-limited signal is completely determined by a discrete sequence of its samples provided that the sampling rate is more than twice the highest frequency in the signal, or in other words, a sampled signal can only be reconstructed if its highest frequency is no more than half the sampling frequency. To ensure this some form of low-pass filtering (anti-alias filter) is applied prior to analogue-to-digital conversion.

However, any filter introduces phase shifts, and unless this is a linear function of frequency, the different frequency components in a signal will be delayed by differing amounts, leading to possible errors when determining the timing of waveform features. Ideally all filters used in recording a set of signals should have identical linear phase delay, as this then simplifies the calculation of timing relationships between features on different signals. In practise, real-world low-pass filters approximate linear phase to about half their cut-off frequency, depending on filter type and
order (Bessel filters have a very good phase response but poor amplitude cut-off; Chebyshev filters have poor phase response but a sharp cut-off; Butterworth filters have both good phase response and amplitude cut-off, and are often the filter of choice).

The absolute value of some biosignals, such as pressure, temperature and position signals, is important and so the analogue front-end must be direct-coupled to allow for this. For other classes of biosignals pre-processing usually includes some form of high-pass filtering to remove very low frequencies and drift.

At a minimum, the bandwidth of a signal’s pre-processing stages is important information that should be kept; preferably all characteristics of filters should be recorded, as this then allows phase delays to be compensated for when making temporal measurements.

**Measurement uncertainty**

The measured value of a physical quantity is never exact but instead is an approximation or estimate [6]. The nature if this uncertainty may be known beforehand, having been obtained from prior measurement and calibration; or it may be derived by analysis of measurements; or if a researcher considers uncertainty, it may be thought of as say three significant figure accuracy. Regardless though, it is important to be able to associate measurement uncertainty with data, and with digital signals.

### 1.1.2 Signal processing

After acquisition, further processing is usually performed to identify and extract information from a signal. This is illustrated in Figure 1.2. This processing may involve calculating statistical measures, filtering, feature recognition, Fourier and wavelet analysis, entropy calculation, and so on. Irrespective, it may be parameterised and the parameters themselves be time varying quantities and thus be considered an associated signal. To allow reproduction, and to verify the result’s credibility, prove-
nance information should be associated with the result — for example, details of input signals, algorithms and parameters used — and ideally, automatically as part of processing.

![Diagram of signal processing](image)

**Figure 1.2** — Signal processing: some method or algorithm, controlled by parameters, gives rise to a dataset.

### 1.1.3 Biosignal characteristics

The characteristics of a few common biosignals are summarised in Table 1.1 — the first part of the table, adapted from [7], shows electrophysiology signals; the second part gives examples of non-electrical biosignals. Modern technology allows the full dynamic range of these signals to be sampled and converted at rates significantly greater than their maximum expected frequency component, which simplifies their processing as digital signals.

<table>
<thead>
<tr>
<th>Biosignal class</th>
<th>Frequency range</th>
<th>Dynamic range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell action potential</td>
<td>100 Hz – 2 kHz</td>
<td>10 µV – 100 mV</td>
</tr>
<tr>
<td>Electroencephalogram (EEG)</td>
<td>0.5 – 100 Hz</td>
<td>2 – 100 µV</td>
</tr>
<tr>
<td>Evoked potentials (EP)</td>
<td>100 Hz – 3 kHz</td>
<td>0.1 – 20 µV</td>
</tr>
<tr>
<td>Single fibre electromyography (SFEMG)</td>
<td>500 Hz – 10 kHz</td>
<td>1 – 10 µV</td>
</tr>
<tr>
<td>Electrocardiogram (ECG)</td>
<td>0.05 – 100 Hz</td>
<td>1 – 10 mV</td>
</tr>
<tr>
<td>High-frequency ECG</td>
<td>100 Hz – 1 kHz</td>
<td>100 µV – 2 mV</td>
</tr>
<tr>
<td>Arterial blood pressure</td>
<td>0.01 mHz – 100 Hz</td>
<td>40 – 200 mmHg</td>
</tr>
<tr>
<td>Normal body temperature</td>
<td>4 µHz – 0.01 mHz</td>
<td>36 – 37.5 °C</td>
</tr>
<tr>
<td>Intracellular calcium concentration</td>
<td>10 mHz – 50 Hz</td>
<td>50 nM – 1.5 µM</td>
</tr>
<tr>
<td>Respiratory flow</td>
<td>50 mHz – 10 Hz</td>
<td>0 – 700 l/min</td>
</tr>
</tbody>
</table>

**Table 1.1** — Characteristics of common biosignals.
1.2 Biosignal Storage Formats and Standards

A number of different formats and standards have been developed in which to store physiological signals [10,11]. Some of the formats in current use are listed in Table 1.2, with other formats proposed in the past decade but not so commonly used listed in Table 1.3.

The BioSig Project [12], a well established open source software project, supports more than 50 different biosignal formats used in research. An analysis of the origins of these formats is in Table 1.4, showing that the majority of them are manufacturer specific.

Standardisation of signal formats has been largely driven by a need for researchers to share and exchange data. Later, as the use of digital signals became common in healthcare, came a need for interoperability between biosignal equipment from different vendors (that often used different technologies), and a need for signals to be compatible with medical record systems [21], along with requirements for privacy, accountability and transparency. These areas are considered important for medical signal formats, and regulatory bodies, such as the United States’ Food and Drug Administration (FDA), now have a role in the standardisation process.

Medical and healthcare standards, including standards for biosignal storage and exchange, have been developed by several national and international bodies, including the International Organization for Standardization (ISO), the American National Standards Institute (ANSI), the Institute of Electrical and Electronic Engineers (IEEE), and the Comité Européen de Normalisation (CEN). Medical biosignal standards are only one component of very large and growing sets of specifications, such as the joint CEN ISO/IEEE 11073 standards for medical/health device communication. Format specifications are long and complex, having evolved over several years to meet many diverse requirements — the Standard Communications Protocol for
1.2. BIOSIGNAL STORAGE FORMATS AND STANDARDS

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFDB</td>
<td>Waveform data base [13]</td>
<td>Introduced around 1980 as the default format used by PhysioBank, a major physiological signal repository. A range of open-source software tools are available. An 11 page specification.</td>
</tr>
<tr>
<td>DICOM 30</td>
<td>DICOM supplement 30 waveform standard [16]</td>
<td>Specified in 2000, the supplement describes how to use the DICOM imaging and communications standard to interchange clinical waveforms. A 77 page supplement.</td>
</tr>
<tr>
<td>FDA-ECG</td>
<td>FDA format for ECG signals [18]</td>
<td>Developed in 2002 as an XML-based format for including ECGs in clinical drug trial reports. Now part of the Health Level 7 (HL7) standards. The design document is 27 pages.</td>
</tr>
<tr>
<td>EDF+</td>
<td>European data format plus [19]</td>
<td>A revised version of EDF from 2003, which retains backwards compatibility. Widely used, apart from advanced features which are not supported by readily available software. The specification is 7 pages.</td>
</tr>
</tbody>
</table>

Table 1.2 – Important biosignal formats.

ECG, ISO 11073-91064:2009 [15] is a 159 page document, the end result of a standardisation project that started a decade earlier.

There are many manufacturer specified formats, with vendors developing proprietary formats to capture and keep market share. This has now changed in clinical environments, with government requirements for devices to comply with recognised international standards. For non-clinical applications, some manufacturers now support formats such as EDF, although because research equipment is less
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecgML</td>
<td>A markup language for electrocardiogram data acquisition and analysis</td>
<td>Designed in 2003 as an XML-based format to hold the minimum set of information for the representation and storage of ECG signals.</td>
</tr>
<tr>
<td>MEF</td>
<td>Multiscale electrophysiology format</td>
<td>Designed for continuous, long-term recording of electrophysiological signals from human subjects undergoing evaluation for epilepsy surgery.</td>
</tr>
<tr>
<td>SleepXML</td>
<td>Polysomnographic data storage and exchange</td>
<td>Designed in 2004 as an XML-based format to store raw and processed data in polysomnograms. The format appears to have been abandoned.</td>
</tr>
<tr>
<td>Unisens 2.0</td>
<td>A universal data format for multi sensor data</td>
<td>Designed in 2008 as a universal and generic format for recording and archiving sensor data from various recording systems. XML with separate binary data files.</td>
</tr>
</tbody>
</table>

Table 1.3 – Other biosignal formats, not in common usage.

<table>
<thead>
<tr>
<th>Category</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer specified for their equipment:</td>
<td>75%</td>
</tr>
<tr>
<td>Research developed standards in common use:</td>
<td>10%</td>
</tr>
<tr>
<td>Developed for specific projects and not in general use:</td>
<td>5%</td>
</tr>
<tr>
<td>International standards used in medicine:</td>
<td>5%</td>
</tr>
<tr>
<td>Audio formats used to store biosignals:</td>
<td>5%</td>
</tr>
</tbody>
</table>

Table 1.4 – Origins of biosignal formats referenced by the BioSig Project.

likely to be replaced or upgraded than that used in a clinical setting, older equipment using proprietary formats is often still in use.

In contrast to medical standards, formats developed and used by the research community are much simpler and often designed around specific project requirements and/or researchers’ needs for originality. Sometimes, as a convenient way to exchange data, formats from other fields are used, such as digital audio formats, or even spreadsheet files. Some research developed formats have become de-facto standards within a research domain; others disappear some time after publication.
Most formats have a generic layout, consisting of a header containing metadata followed by binary signal data. Research oriented formats generally keep metadata as text; more formal standards, such as those developed for equipment interoperability, tend to store metadata in binary fields, with well specified values but limited extensibility. As well as holding generic metadata, such as that described in Section 1.1.1, the different formats usually have specific fields for domain specific metadata, which may include patient details such as name, date-of-birth, symptoms, medication, physical measurements, etc.

We now look in more detail at some biosignal formats, general scientific data formats, and ways to achieving format independence, in order to gain an understanding of what might be required for biosignal abstraction.

### 1.2.1 Waveform database format

The WaveForm DataBase format, WFDB [13], had its origins in 9-track magnetic tape recordings of ECG data from more than 30 years ago. A WFDB recording consists of a text header file that has a first line giving the recording’s name, start date and time, number of signals, their common sampling frequency (the frame rate), and an optional number of segments. Following are lines for each signal, giving the name of the signal’s data file; physical units; date encoding; samples per frame; clock skew; offset and gain; data resolution; initial value; an optional checksum; and a free text description. Signals can be grouped by sharing the same data file; in this case their values are multiplexed in the file.

Comment lines, with additional information about the recording, can follow signal specifications. No structure is imposed on these comment lines, although in practice, PhysioBank (the main repository of WFDB files) assigns database specific layouts to these lines to hold patient demographics, medication, and other information.
Each data file contains binary data for all signals that reference it, using the specified numeric format (which range from 8-bit signed differences to 8, 10, 12, 16, 24 and 32-bit integer amplitudes packed in various ways).

Multi-segment recordings are defined by having the first line of the header file specify a number of segments. The remainder of the header then describes segments (instead of signals) by giving the name of each segment’s header file; segment header files then specify signals. All segments must have the same sampling frequency.

Events in a recording are called annotations and are stored in separate binary files. Each annotation is encoded as an integer and stored with its time of occurrence and optionally, a duration. The WFDB User Guide defines standard annotation codes, mainly for properties associated with an ECG waveform; users can also define their own codes.

The WFDB format is primarily used for recordings held in the PhysioBank repository [27] repository, part of the PhysioNet Web resource. PhysioNet also includes PhysioToolkit, a collection of software that works directly with WFDB recordings.

1.2.2 European data format and extensions

The European Data Formats, EDF [14] and EDF+ [19], are widely used by researchers. EDF was designed in 1991 as part of a European wide project to collect and analyse sleep-wake recordings; EDF+ was specified in 2003 to address shortcomings found in using EDF. These formats are capable of holding many different classes of signals, a characteristic of polysomnography recordings.

An EDF file has a text header followed by binary signal data. The header itself consists of a 256-byte long preamble describing the recording, followed by a variable length section describing the signals in the recording. Signal data follows as a temporal sequence of records, where all records are of equal duration, with each containing all data points for the time spanned by the record, in separate data blocks for each signal. Data is stored as little-endian 16-bit signed integers.
Within the header, metadata fields specify study and patient details, the start date and time of the recording, and for each signal, a label, its physical units, transducer or electrode details, and pre-filtering details. Other header fields give the duration of a data record, the number of samples each signal has in a data record (and hence the sampling frequency), and signal scaling and offset quantities used to derive the actual physical value (as a floating point number) from the stored sample.

The structure of an EDF+ file is identical to EDF’s; the differences are a more complete specification of metadata content (including standard texts) and providing a reserved signal channel for time-stamped annotation lists. This allows more timing information, events, discontinuous recordings, and general text to be stored. EDF+ files that do not contain discontinuous recordings can be processed and viewed using software tools designed for EDF.

The BioSemi Data Formats, BDF [28] and BDF+ [29] have been specified as identical to the respective EDF versions except for storing signal data as 24-bit signed integers.

A number of open-source tools and viewers are available for EDF and EDF+, including EDFbrowser\(^1\) and Polyman.\(^2\) The EDF formats have also gained acceptance with equipment providers. EDF+ can carry arbitrary annotation texts, although apart from pre-defined polysomnography texts, users need to define their own structure and content, opening the way for metadata incompatibility.

### 1.2.3 General data format

The General Data Format, GDF [20], was developed for use in brain-computer interface (BCI) research, which like polysomnography, uses many different classes of signals. Although GDF’s design was based on EDF it is not backwards compatible: binary data is allowed in a header, some fields are shortened, and reserved space is valid.

\(^1\)http://www.teuniz.net/edfbrowser/

\(^2\)http://www.edfplus.info/downloads/software/polyman.zip
used. This allows the storage of substantially more metadata, including additional patient data, detailed filter specifications, and electrode positions and impedance.

Following the main header is a second optional header with metadata encoded in a tag-length-value (TLV) format. This can include the patient’s Snomed CT classification, details of recording equipment, orientation of magnetoencephalography (MEG) sensors, descriptions of user specified events, and free format annotation.

Binary records of signal data then follow in fixed duration records as for EDF. Data can be stored in many different numerical formats — 8, 16, 24, 32, 64-bit, signed or unsigned integer; 32, 64, or 128-bit floating point.

An event table follows the signal data block and provides coded values and timing information for events in the recording. Several standard event codes are defined, mainly for clinical neurophysiology. What appears to be lacking is the ability to store arbitrary temporal annotation.

GDF has a range of software available through the BioSig Project and is well represented on the Internet. It is mostly used for BCI research.

1.2.4 Medical waveform format — encoding rules

The Medical waveform Format Encoding Rules, MFER [17], were developed by a consortium of Japanese equipment manufacturers as a general specification for medical waveform data that “may be used with other relevant protocols, such as HL7, DICOM, ISO/IEEE 11073, and database management systems for each purpose.”. It is now part of the ISO/IEEE 11073 family of standards, and intended as a base for domain experts to extend as they develop standards in their fields.

A MFER file is a simple sequence of tag-length-value blocks, encoded as binary. Each block begins with a tag, that gives the block type, then a field with the length of the block’s data, and then actual data. There are four classes of block types, to define basic, supplementary, extended and user values. Each type of block has an initial value, which is used until a block of that type defines a newer value; blocks can also
reset values to their initial state. The attributes of a signal channel are initially set to the current global values and overridden on a per channel basis.

Besides signal data, blocks may specify sample rate, physical units, data type, resolution, layout and compression, waveform classes and attributes, labels and descriptions, dates and times, filtering details, time stamped events, general annotation, system messages, pointers to external resources, and various other values.

Patient demographic information is encoded using extended tags, with a recommendation that local standards such as HL7 or DICOM be used to represent this information. Although waveform attribute codes for medical signals are defined, standard event codes are not, as these are intended to be specified by the application in which the MFER implementation is embedded.

MFER was developed for use in medical equipment and this is where it is being applied. As a general extensible waveform standard it has potential for wider use, however there appears to be no open source software for it, with only preliminary versions of signal viewer executables available for download [30].

1.2.5 XML based formats

The Extensible Markup Language (XML) [31] provides a generic framework in which to represent hierarchical documents. A document is marked up as a nested sequence of elements, beginning with a root element. Each element is delimited by a matching pair of tags enclosed in angle brackets ('<' and '>'); an opening tag has the element name and optional attributes; a closing tag contains a slash character ('/') and the element’s name. An element’s contents is a sequence of other elements, text data, comments, and processing instructions.

XML-based languages are defined by specifying domain specific element and attribute names, and by placing restrictions on content. They are widely used in particular fields of interest. Some such languages are MathML [32], SBML [33], CellML [34]
and later versions of the HyperText Markup Language, HTML [35]. XML languages have been used to store binary data, including biosignal data:

- FDA-XML, the FDA format for ECG signals [18].
- ecgML, a markup language for electrocardiogram data acquisition and analysis [22].
- OpenXDF, the Open eXchange Data Format [24].
- SleepXML, for polysomnographic data storage and exchange [25].

XML content though must be text, which means binary data has to be serialised in some form, leading to processing and storage inefficiencies; additionally the tree-structured and variable-length nature of elements and attributes makes it hard to randomly access data. For these reasons XML-based formats are not commonly used to hold binary data, and when they are, with only relatively small datasets. Large sets of structured numeric data are normally stored in different ways, as described in the following section.

### 1.2.6 Science data formats

Science data sets can be immense. For instance, the Large Synoptic Survey Telescope will produce around 30TB of data each night [36]; other science and engineering disciplines also produce massive quantities of data. Several formats have been developed to work with very large amounts of structured numerical data, although many of them are part of computational-intensive software libraries designed for specific application domains which don’t have any urgent need to exchange data [37]. There has however been a growing recognition of the need to share data between different research communities, and a number of standard formats for storing and exchanging large numerical datasets have been developed:

- Common Data Format, CDF [38].
- Hierarchical Data Format version 5, HDF5 [39].
- Network Common Data Format, NetCDF [40].
All of these formats provide a similar abstraction layer; have a model for structured data; are platform-independent; and work efficiently with data in any dimension or form. HDF5 appears to be the most popular and widely used of these formats; the latest version of NetCDF in fact uses HDF5 for its underlying storage.

Hierarchical data format

The Hierarchical Data Format (HDF) implements a model for managing and storing data — it provides a meta-model with which to define structured storage formats for use in application and problem domains. HDF was originally developed in the 1980s with the original version now known as HDF4 [41]. A completely new format, HDF5 [42], was released in 1998 to address some of the limitations of the original and to meet the requirements of current and future users.

HDF5 is specified by an Abstract Data Model, which defines concepts to use when specifying and describing complex datasets; a Storage Model, for representing objects in the Abstract Model; a Programming Model, to manipulate objects in the Abstract Model; and a Library, which implements the Programming Model, manages data transfers, and exports an Application Program Interface (API) for use by applications.

The Abstract Data Model is not a model of any particular application domain. Instead it allows users to map their data models to concepts and objects defined by the Abstract Model, so that data can be stored as Storage Model representations, or in other words, as HDF5 files. The concepts defined by the Abstract Data Model include those of File, Group, Datatype, Dataspace, Dataset and Attribute.

A File is a hierarchical tree of named Groups and Datasets, beginning with a root Group; each Group can contain other Groups as well as Datasets. A Dataset is a multi-dimensional array of some Datatype, where Datatypes are both the standard numeric types and user defined, and Dataspaces define the array structure of the Dataset.
One or more dimensions of a Dataspace can be declared to be extendible, which enables the underlying Dataset to grow as data elements are added; Dataspace declarations also allow for efficient reading and writing of selected ranges within large Datasets, with dynamic mapping of data between different array shapes. Within a Dataset, numbers are stored using generic types; the library automatically converts between native formats used by an application’s computer platform. A Dataset can also be chunked, which can improve data access times as well as enabling an extendible Dataset to be compressed.

An Attribute is a name/value pair; they can be associated with Groups, Datasets and user Datatypes.

The HDF5 Abstract Data Model is in many ways conceptually similar to the XML Information Set [43] model, with each able to describe a diverse range of structured data and document types by mapping local structure to model concepts. A XML Document corresponds to an HDF5 File with each respectively containing a nested hierarchy of either XML Elements or HDF5 Groups and Datasets. In both cases attributes can be associated with components in the hierarchy. The major difference between the two models is that XML is designed for structured text documents and has additional concepts for this purpose, whereas HDF5 is designed for structured numeric data and so has different additional concepts.

HDF5 is a mature, robust and well supported format that can both describe complex data structures and relationships, and efficiently store and retrieve very large quantities of data across all common computing platforms. Its software library is stable, well documented, has a large installed user base, and is available for modern programming environments.
1.2. BIOSIGNAL STORAGE FORMATS AND STANDARDS

1.2.7 Format independence

As an alternative to defining another standard biosignal format, various projects have developed ways to access and work with biosignal information regardless of the underlying storage format. These include the following.

libRASCH

libRASCH [44] is a programming library designed for biological signals. It provides access to a variety of formats, along with signal processing (mainly ECG related) and viewing. However little, if any, development appears to have occurred since 2005.

BioSig Project

The BioSig Project [45] includes a programming library to access biosignals in different formats; a toolbox of processing and analysis functions for Octave and Matlab; and a signal viewer and scoring system designed for EEG waveforms. The library and tools are oriented towards BCI research and part of the same project for which the GDF format was designed.

PhysioBank

PhysioBank [27] provides format independent access to WFDB and EDF signals held in a public repository and available as Web resource. It has a comprehensive collection of physiological signals organised into more than 50 databases. Text descriptions are provided of databases, recordings and annotators used. Users may find databases of interest using text search; directly browse the repository’s web pages; or use an Automated Teller Machine function which can plot annotated waveforms, display time-series, histograms and summary information, and download recording files. Separate to the Web interface, the \textit{wdflib} software library (part of PhysioToolkit) includes functions that give direct access to recordings via the Internet.
1.3 The Evolving Web

The World Wide Web [46] has irreversibly changed the way we search and access information.

The Web is a large set of interlinked web pages generally designed to allow humans to obtain information by reading content, reasoning about it, and following links to related subjects. The process of extracting information and reasoning about knowledge, as done by users, is difficult to automate, especially if the document was not marked up with machine-readable semantic content. Web documents do not usually provide this, as the HyperText Markup Language, or HTML, was designed to describe document structure and presentation, and not to provide meaning for content.

1.3.1 The Semantic Web

In 2001, Tim Berners-Lee proposed the Semantic Web [47], a future Web in which information is given well-defined meaning so as to allow software agents to process and reason over content. With this proposal the World Wide Web Consortium, W3C [48], began an ongoing program to develop standards and frameworks for sharing and reusing knowledge on the Web. Semantic Web technology is now being used more and more extensively [49] — major examples include BBC websites, eGovernment publishing initiatives, the Gene Ontology,3 and the Bio2RDF project.4

Key to the Semantic Web is making statements and assertions about identifiable resources and their relationships using well-defined vocabularies of terms and concepts. Its core framework is defined by a set of specifications developed by the World Wide Web Consortium — resources are named using Uniform Resource Identifiers (URIs) and have statements made about them in the Resource Description Framework (RDF), using vocabularies and ontologies from domains of interest that have

3http://www.geneontology.org/
4http://bio2rdf.org
1.3. THE EVOLVING WEB

been defined in RDF Schema (RDFS) and the Web Ontology Language (OWL). Interoperability and knowledge sharing is facilitated by the reuse and extension of standard ontologies.

1.3.2 Resources and RDF

A resource is any thing that can be identified.

A resource’s identifier is known as a Uniform Resource Identifier [50], or URI, and can identify both abstract concepts and resources with concrete representations. URIs are sequences of characters which start with the name of a scheme; then a colon (‘:’); and then a scheme-specific part — a familiar URI scheme is http, used for web addressing, such as in http://www.w3c.org. URIs can be, and often are, abbreviated as qualified names (QNames) by defining and using a namespace prefix with a local name part.

Relationships and statements about resources are made using the Resource Description Framework [51], or RDF. An RDF statement is a triple with three components — subject, predicate (or property), and object. The subject and predicate refer to resources (via a URI); the object can either refer to a resource or specify a literal value; additionally, subjects and objects can be blank nodes. A set of statements or triples makes up a directed labelled graph, called a RDF graph; a single statement is depicted graphically in Figure 1.3.

![Figure 1.3 – An RDF statement.](https://example.com/image)

RDF graphs can be serialised in several different ways including as RDF/XML [52], an XML format often mistakenly confused with RDF itself; Notation3 (N3) [53], designed to be human readable; N-Triples, a list of triples and an N3 sub-

---

5These do not have a URI.
set; Turtle [54], a subset of N3 providing a compact and natural text format; and header-dictionary-triples (HDT) [55], a new binary format for efficiently publishing and exchanging very large RDF datasets.

The majority of RDF implementations support named graphs [56], an extension whereby RDF graphs are named by giving them a URI, allowing statements about a graph to be made. These statements can be used to record provenance, control access, track usage and restrict queries to a particular graph.

RDF graphs are queried using the SPARQL Protocol and RDF Query Language [57]. Queries are formed by specifying triples to find, where any component of a query triple may be a variable, and then filtering and combining the resulting sets of matching triples using both union and intersection. Queries may be restricted to particular named graphs or be made over all graphs in a store. The results of a query can be presented as a table of variable/value pairs; as a graph of RDF statements; as a true/false assertion; or as an RDF graph describing the result set. The latest version of SPARQL also allows graphs in a store to be updated and managed [58].

**Formalising meaning**

While RDF is good at describing basic semantic structure and taxonomies it has few constraints and allows arbitrarily complex graphs of triples to be constructed. RDF Schema (RDFS) [59] and the Web Ontology Language (OWL) [60], have been defined to provide structure, specify meaning, and describe types and classes of resources and relationships for the Resource Description Framework.

RDFS defines a vocabulary to use for describing RDF vocabularies by specifying URIs with well-defined meaning. A rdfs:Class is a group of resources; members of a class are called instances; the rdf:type property asserts that a resource is an instance of a class. RDFS includes properties to denote sub-class and sub-property relationships; to specify a property’s domain and range; and to provide labels and descriptive text to resources.
1.3. THE EVOLVING WEB

OWL extends RDFS’s vocabulary while at the same time placing restrictions on the meaning of concepts — not all owl:Classes are rdfs:Classes. Amongst other things, OWL allows the nature of relationships to be specified (e.g. functional, reflexive, asymmetric and transitive); can define cardinality restrictions on properties with respect to a class; can assert class, property and individual equivalence and disjointness; and can use set operations to derive classes from others. A newer version, OWL 2 \[61\], provides increased functionality and expressiveness while retaining backwards compatibility.

The use of RDFS and OWL supports automated derivation of facts as they allow sub-class, sub-property, equivalence, and similar relationships between concepts to be specified, along with property domains and ranges. A number of reasoners and inference engines exist, either as standalone applications (c.f. Pellet \[62\] and HermiT \[63\]) or integrated with RDF frameworks (c.f. as part of Jena\(^6\) \[64\]). Additionally the Semantic Web Rule Language, SWRL \[65\], allows users to write rules using OWL concepts to reason about OWL individuals.

Open knowledge

Knowledge is open on the Semantic Web — we know only what has been stated or can be reasoned from known facts. This is in contrast to closed systems such as databases, where the absence of a fact implies its falsehood; on the Web, the absence of a statement means we don’t know if it is true or false. As a result, we do not have to fully describe a resource using all applicable properties in a given ontology but instead only use properties relevant to our needs.

1.3.3 Linking data

Semantic Web standards are being used to create a Web of Data. By conforming to a set of Linked Data \[66\] guidelines, published data is able to be explored as part of a connected data web:

\(^6\)http://jena.apache.org/documentation/inference/
1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. Provide useful information, using the standards (RDF*, SPARQL), when someone looks up a URI.
4. Include links to other URIs, so that they can discover more things.

The use of these guidelines have resulted in a growing collection of linked data resources spanning many disciplines, and made available through web-sites such as the Data Hub, and BioPortal [67] from the National Centre for Biomedical Ontology.

1.3.4 Formalising time

Time is the critically important dimension in virtually all signals. Allen [68] assumes time is linearly ordered and develops a formal interval-based system of temporal logic, specifying 13 possible relationships between two proper intervals (equals and before, meets, overlaps, during, starts, and finishes, along with their inverses). The Time Ontology in OWL, OWL-Time, [69] provides a vocabulary for describing the relationships of Allen, along with dates, times, durations, and related concepts, including Instant, Interval, and ProperInterval. OWL-Time though can not handle arbitrary temporal systems.

The Timeline Ontology [70] extends OWL-Time to provide concepts for working with different temporal coordinate systems. In this ontology, a TimeLine is a coherent backbone for addressing temporal information, on which Instants and Intervals may be defined; TimeLines may be abstract, continuous or discrete; continuous TimeLines may be physical or relative. Conversion between different temporal systems (that is, timelines) are defined using TimeLineMaps, which allow offsets and rates to be given.

---

7 See http://lod-cloud.net/ for a visualisation of the Linking Open Data cloud.
8 http://datahub.io/
9 http://bioportal.bioontology.org/
1.3.5 Provenance and annotation

The growth of the Semantic Web and Linked Data has resulted in a need for standard frameworks in which to describe resource provenance and annotation, and the W3C is developing standards to ensure interoperability in these areas.

**Provenance**

The Provenance Working Group[^10] has a mission “to support the widespread publication and use of provenance information of Web documents, data, and resources” and has published working drafts of several related specifications. As a starting point, the PROV Ontology[^11], PROV-O, uses three classes as a basis for the rest of the ontology — an Entity is a thing to provide provenance for and may be derived from other entities; an Activity is something that occurs in time, acts upon entities, and may depend on other activities; an Agent is responsible for an activity and its generated entities and may act on behalf of other agents.

The PROV model is intended to be used as a general structural framework for provenance, allowing application ontologies to define domain-specific terms as required. One such ontology is the Provenance Vocabulary[^11][^72], a specialisation designed for describing provenance on the Linked Data web.

**Annotation**

Various RDF models for annotation have been proposed, including the Annotation Ontology, an open ontology for annotating scientific documents on the Web[^73], which is a successor to the Annotea Project[^74], and the Open Annotation Collaboration[^75]. The Open Annotation Community Group, under the auspices of the W3C[^12], have now combined these models and published a community draft standards for both a core data model[^76] and extensions to it. In the core model, an

[^10]: http://www.w3.org/2011/prov/wiki/Main_Page
[^12]: http://www.w3.org/community/openannotation/
Annotation describes one or more Targets using a Body, with provenance information stating how and when the annotation was created — targets may be regions in larger documents; the body, or content, of an annotation is not restricted to text and is a resource in its own right. Non-text content resources are called semantic tags and their use simplifies knowledge discovery by inference tools — tags, and their relationships, can be specified using ontologies.

1.4 Modelling and Simulation

An integral part of research into physiological processes is developing and simulating mathematical models, and specification languages such as the Systems Biology Modelling Language, SBML [33], and CellML [34], have been created to provide standard formats and facilitate model development — CellML enables models to be written as differential algebraic equations, with complex models able to be constructed from simpler components; uses standard ontologies to assign biological meaning; and enforces units and dimensional consistency between all variables, parameters, and components.

A variety of tools can be used for model simulation — for CellML these range from dedicated packages, such as OpenCor;\(^{13}\) to large-scale simulation platforms, such as OpenCMISS;\(^{14}\) and to standalone program code generated via the CellML API. Specifying the model(s) to be simulated, initial values for parameters, the computational algorithm(s) to use, and the generation and presentation of results is usually done in a tool specific way, which may require users to write and modify source code. In response to this, the Simulation Experiment Description Markup Language, SED-ML [77], has been developed to provide a standard format for specifying simulations, with ongoing work to extend SED-ML’s capabilities.

Simulation of a model often includes the linking of models to experiment data, including biosignals. A lack of suitable standards means that simple data formats,

\(^{13}\)http://opencor.ws
\(^{14}\)http://physiomeproject.org/software/opencmiss/
for example comma-separated-value (CSV) files [78], are used to store and exchange signals — as a result any stored metadata in the original format is not available to the simulation.

### 1.4.1 Formalising biology

Model specification languages usually directly represent the mathematics of a model and not the semantics of the specific research domain being modelled. Instead, domain specific meaning is provided through model annotation and formalised by using terms and concepts defined in ontologies, such as in the BioModels Database [79] where specific annotation properties, or qualifiers,\(^\text{15}\) are used to define the relationships between elements of a physiological model and the biological object an element represents.

Several initiatives provide a growing number of reference ontologies to the biological community, including the Open Biological Ontology (OBO) Foundry\(^\text{16}\) [80], the National Centre for Biomedical Ontology (NCBO) [81], and the Ontology Lookup Service\(^\text{17}\) [82] of the European Bioinformatics Institute (EBI). Amongst these ontologies is the Ontology of Units of Measurement, the Foundational Model of Anatomy, and the Ontology of Physics for Biology, OPB\(^\text{18}\) [83], a reference ontology of classical physics designed for annotating the biophysical content of biomedical datasets and analytical models.

The RICORDO Project\(^\text{19}\) [84] is working to improve the sharing of data and models by developing a multi-scale ontological framework and standards for semantic interoperability in biomedical research. This includes the standardisation of metadata, the use of reference ontologies for annotation, and providing tools for metadata management and query.

\(^{15}\)http://biomodels.net/qualifiers/  
\(^{16}\)http://www.obofoundry.org/  
\(^{17}\)http://www.ebi.ac.uk/ontology-lookup/  
\(^{18}\)https://sites.google.com/site/semanticsofbiologicalprocesses/projects/the-ontology-of-physics-for-biology-opb  
\(^{19}\)http://www.ricordo.eu/home
1.5 Difficulties

The diversity of biosignal formats has created difficulties for researchers [10, 85], not only with data exchange between groups but also with equipment and/or software incompatibilities.

A comprehensive review of biosignal data formats written in 2002 [21] describes the development of then current biosignal standards and formats and calls for the ongoing specification and adoption of open standards for interchange and archiving. A challenge raised is how to store and reference biosignal datasets so that the information in them can be linked with other types of information.

A January 2006 workshop discussed the challenges faced by neurophysiologists and sleep-researchers [86] due to the multitude of biosignal formats in use, and called for the development of a new standard format for polysomnography data [87]. There has been little progress since – almost six years later a member of the steering committee, when asked about progress, stated: “Unfortunately, not much has happened since the conference. …no consensus for data sharing has taken root.” [88].

Is inventing another standard biosignal format the correct way to resolve difficulties caused by a multitude of formats? Proposals for new formats are occasionally published and then disappear. Possibly the cost to a researcher of using a different biosignal format to what they currently use is too high — software and tools may need replacing or updating with subsequent familiarisation; equipment may need upgrading; archived recordings may need conversion — all things which consume time and budget. Unless there are very compelling reasons, maintaining the status-quo is the obvious choice. The nature of research though is to produce new things, so when and if the time comes to choose a signal format it may well be one a researcher has played a role in designing, rather than using or extending an existing standard.
1.5. DIFFICULTIES

1.5.1 Metadata consistency

The actual format used to store a biosignal recording is largely irrelevant. Regardless of format, it is important:

- to be able to access both signal data and metadata in a format independent way.
- to have agreement about the meaning of metadata terms and properties.
- to do this in an open and extensible manner.

Working with metadata in a consistent way is the greater challenge — a regularly sampled signal, being a sequence of numbers and a sampling rate, has only a limited number of practicable representations.

Metadata is data about data, and for scientific datasets, may be classified into the following general categories [89]:

1. Syntactic metadata is metadata describing how data is structured and stored, such as file layout and how numbers are represented. This information is often implicit, defined by say a format specification.
2. Content or search metadata describes the intrinsic nature of data, such as the physical parameters and variables measured, and when and how it was collected.
3. Semantic or use metadata provides meaning by relating content to some known context; what a dataset signifies in terms of the domain of interest it is part of.

In terms of biosignal datasets, syntactic metadata is important for each particular dataset format but is otherwise irrelevant — it says nothing about what is in a recording nor what the recording is about. As an example, EDF stores signal data as 16-bit integers along with scaling factors; a general biosignal model is interested in the value of a physical quantity at some time instant, not how it has been encoded, even though encoding may be part of a specific file format’s data model.

Following are two examples of the use of metadata — the first describing content; the second, describing semantics.
Content metadata: units of measurement

Members of a research group usually have a tacit understanding about the units they use to measure quantities. But what happens when a different group uses another’s data and makes their own assumptions? Misunderstandings about units have lead to costly and embarrassing failures including those of NASA [90] and the Strategic Defence Initiative [91].

How are units specified in different biosignal formats? Many biosignals are measured in microvolts; as an example, how is this unit specified?

- EDF has no requirement for its units field apart from it being a text field. The specification document has an example that uses ‘µV’ for microvolts, where ‘µ’ is the Unicode symbol for the micro- prefix.
- EDF+ stipulates that the letter ‘u’ be used for micro- along with a prescribed set of abbreviations for common units — microvolts is represented as ‘uV’.
- GDF has a fully specified numerical encoding scheme for units, based on ISO 11073-10101:2004. Microvolts would be encoded as the number 4275.
- MFER uses a numerical encoding scheme and includes a table of values for common medical waveform units, but does not have any apparent means of extension. Microvolts would be encoded as the number 0.
- WFDB specifies a character string without embedded whitespace for physical units. No predefined values are given.
- Files produced by a sleep analysis software package were found to be in the Stanford Data Format [92] and using ‘µV’ for microvolts.

The use of the ‘µ’ symbol may also be problematic. Unicode [93] has two different characters that look the same in some fonts – a micro sign, which is encoded as U+00B5, and the Greek letter μ, which is encoded as U+03BC. Any check based on string comparison will not identify the different character strings as representations of the same thing.
Although some storage formats provide a controlled set of values in which to express units, others do not. There is certainly no de-facto standard shared by the different formats.

**Semantic metadata: PVC — contraction or complex?**

A *premature ventricular contraction* is a common form of cardiac arrhythmia in which the heartbeat is initiated by the ventricles instead of the sinoatrial node,\(^\text{20}\) causing the ventricles to contract before they have been filled with blood by the atria. Cardiac electrophysiologists use the abbreviation PVC to refer to such events, which they take to mean *premature ventricular complex*; medical classification systems use other descriptions, for instance in the 10\(^\text{th}\) Revision of the International Statistical Classification of Diseases and Related Health Problems (ICD-10), a PVC event has a code of *I49.3* and is described as a *ventricular premature depolarization*.\(^\text{21}\)

### 1.5.2 Ontologies

One way of standardising terms and concepts is by using ontologies to define a simplified, abstract view of the world we are interested in. Within the field of Information Science, an ontology has been described as “an explicit specialisation of a conceptualisation” [94], where a conceptualisation is the objects, concepts and other entities of some domain of interest together with the relationships between them and axioms restricting how terms are to be interpreted and used. Ontologies formalise meaning and so facilitate information sharing — if the same things are referred to in the same way, knowledge can be computationally processed in a similar manner to numeric data [95].

Differences in terminology and convention are usually insignificant to experts in their respective field; they are however significant to software agents and tools, and this is where it is important that cross-referenced classification codes and ontologies

\(^{20}\)http://en.wikipedia.org/wiki/Premature_ventricular_contraction

\(^{21}\)http://apps.who.int/classifications/icd10/browse/2010/en#/I49.3
are in a machine-readable format, so to provide applications with a shared semantic meaning for terms and concepts. Unless programmed to reason over known facts, computers in general cannot know that similar but different terms may in fact be equivalent. This can cause difficulties — with data exchange, long-term archiving, data reuse, collaborative research, and so forth.

### 1.6 Related Work

A number of other research projects seek to provide general frameworks for describing and working with physiological time-series data, including the following.

#### 1.6.1 ROISES

The Research Oriented Integration System for ECG Signals (ROISES) [96] is designed to provide unified access to biosignal databases and accompanying metadata and allow complex, content based queries against diverse biosignal sources. Its initial application is with cardiology data. Text classification techniques are used to dynamically create mappings between a global ontology and local ontologies describing data sources. The global ontology includes concepts from ecgML and the Unified Medical Language System [97]. ECG data sources supported are EDF+, SCP-ECG, and a proprietary relational database.

Amongst other things, the lack of a standardised structure for capturing event information is seen as impeding greater integration of ECG data.

#### 1.6.2 Physio-MIMI

Physio-MIMI (Multi-Modality, Multi-Resource Information Integration environment)\(^\text{22}\) [98,99] is a data integration project being developed for clinical researchers, http://physiomimi.case.edu/physiomimi/index.php/Main_Page
in which biosignal data from several institutions, not sharing a common data model, is combined and queried. Its initial application is for sleep research data, but the system is able to be generalised. The project includes the development of a Sleep Domain Ontology [100] and mappings from SQL databases to obtain terms and concepts to use when constructing queries. It provides a federated view of institutional databases.

1.6.3 The multi-scale annotation format

The Multi-scale Annotation Format (MAF) [101] has been developed as an integrated metadata and annotation framework for working with large, multi-scale, electrophysiological datasets, using a new file format, the Multi-scale Electrophysiology Format (MEF) [23] for storing signal data.

The challenges MAF seeks to address are those of flexibly describing data to allow meaning to be communicated and maintained over time. MAF provides a solution by using a new XML language, XREDE (XML-based Research Data Exchange), in which to describe metadata and annotations (with a XML Schema to define standard terms), along with a software implementation that uses SQL for storage, and Java for incorporation with interactive data-analysis environments, such as Matlab.\(^\text{23}\)

1.6.4 Web-based biosignal visualisation and annotation

A web-based platform for biosignal visualisation and annotation is described in [102]. The project aims to improve the quality of metadata available to biomedical applications by developing tools to allow non-technical users to retrieve, visualise, and annotate biosignals. An extendable and scalable data model is presented as part of the project.

\(^{23}\text{http://www.mathworks.com}\)
Underlying this project is the use of HDF5 (for signal storage); JSON\textsuperscript{24} (to describe metadata); and NoSQL\textsuperscript{25} (as a document store in which to hold and query metadata). No mention is made of using Semantic Web standards for metadata; instead the data model defines fixed field names for metadata and represents values using numbers or strings “which can be of any type or size” — this would compound general issues with metadata standardisation.

1.7 Thesis Overview

This thesis aims to addresses the principal difficulties, caused by a lack of standardisation in how biosignal metadata is represented and assigned meaning, by developing a new, extensible framework in which to describe and work with physiological time-series data. The new framework is called BioSignalML. The remaining chapters describe BioSignalML and illustrate its use.

Chapter 2 develops an abstract model to encapsulate common features of existing biosignal formats and then uses the model as the basis of a new ontology, the BioSignalML Ontology. This ontology, along with existing standard ontologies, is used to specify an RDF model in which to describe biosignal metadata.

Chapter 3 describes a software implementation of the BioSignalML RDF model. This consists of a software library and a web-based biosignal repository, and includes a standard interface to common biosignal storage formats. A new HDF5 file layout for biosignals is also described.

The use of BioSignalML is demonstrated first in Chapter 4, which describes how BioSignalML might be applied to PhysioBank to provide a number of benefits over its current implementation, including making its resources accessible to the Semantic Web.

\textsuperscript{24}http://json.org/
\textsuperscript{25}http://en.wikipedia.org/wiki/Nosql
Chapter 5 then describes how BioSignalML can be used to resolve common interface issues with the use of signal data, whether as part of the modelling and simulation of physiological processes, or when used for data analysis and signal processing.

Chapter 6 concludes the thesis with a discussion and some suggestions for further research.

To improve readability throughout the thesis, URIs have been abbreviated using the namespace prefixes defined in the List of Abbreviations.
Chapter 2

BioSignalML

BioSignalML is the name given to the new framework I have developed for working with biosignals. It is based on an abstract model that encapsulates common features of biosignals and their storage formats, and on a new ontology, the BioSignalML Ontology, which formally defines the model’s concepts. This framework allows biosignal information to be described and exchanged, regardless of the underlying signal format. Sections 2.1 and 2.2 of this chapter describe the model and ontology.¹

BioSignalML uses Semantic Web standards. This includes the use of existing ontologies, some well-established, others evolving with the development of bio-ontologies. Why particular ontologies were chosen and how they are used within BioSignalML is discussed in Section 2.3.

The chapter concludes with a summary of the ontological classes and properties that form the basis of BioSignalML.

¹My terminology assumes signals are time-varying, although the concepts discussed are equally applicable to signals with spatial and spatio-temporal sampling dimensions.
2.1 An Abstract Model

The process of building an abstract model for a particular problem domain is one of finding high-level concepts in which to describe the common characteristics of things in the domain. Applying this process to the biosignal formats of Section 1.2 finds that:

- Storage formats keep several individual signals together as a named entity, usually as a single file, although some formats use multiple files. This collection of signals is captured in a single recording session and pertain to a single subject. The signals may be of a similar type and from a single piece of equipment (e.g. a multi-lead ECG or EEG recording) or be of different types and from different devices (e.g. a polysomnography recording).

A wide range of attributes can be associated with this collection of signals. Minimal information includes subject and investigation identifiers, when signal capture began, its duration, the actual format used to store data, and where it is stored.

- Signals are time-varying and generally sampled at a uniform rate. In addition to actual data values and timing, other information stored usually includes a label, physical units, and details of sensors or electrodes, and any pre-filtering.

- Descriptions of things occurring at points in time and possibly with a duration, are stored.

- Notes or comments describing aspects of the recording session, signals and events may be stored, possibly with details of who made the comment and when it was made.

- Timing within the collection of signals is either measured by sample number (for particular signals) or by elapsed time from the start of the recording session. The start time of the recording session locates it in physical time. Both points and durations in time are used to describe temporal information.
These concepts form the core of the BioSignalML model, which uses the names **Recording**, **Signal**, **Event**, and **Annotation** for the types, or classes, of information stored in a biosignal file.

### 2.1.1 Recording

A **Recording** is a collection of Signals held together as a named entity, all pertaining to one thing (the subject) and which have been recorded in the same session. Core attributes of a Recording are:

- **subject** — Who or what is the subject of investigation.
- **investigator** — Who made the recording.
- **starttime** — When the recording began.
- **duration** — The length of the recording.
- **format** — How actual data values are stored.
- **dataset** — Where data values are stored.

### 2.1.2 Signal

In very general terms, a **Signal** is a sequence of measurements of some varying quantity, ordered by a sampling dimension, normally time. A Signal is always associated with some Recording.

Underlying a Signal is a **Time-series** — a sequence of sample times and data points. The increasing sequence of sample times can be thought of as **Clock Times**. Clock Times may be either uniformly or irregularly spaced.

A Signal has attributes that give meaning to the raw data in a time-series. Core attributes are:

- **recording** — The recording the signal is part of.
- **label** — A descriptive label.
- **units** — The physical units of the quantity measured.
sensor — The transducer or electrode used to capture the signal.
filter — Details of filtering applied prior to sampling.
source — What produced the signal.
rate — The sampling frequency of a uniform signal (or period).
times — An increasing sequence of sampling times of a non-uniform signal.
resolution — The numeric resolution at which data was sampled.

The actual way in which numeric values are represented is considered to be implementation detail and not part of an abstract model.

2.1.3 Event

An Event is something that happens in time (i.e. a recording’s sampling dimension) and may have a duration. Events could be captured as part of a recording (e.g. trigger events) or derived from other signals (e.g. breath events in a hypnogram; ECG beat annotations). The value associated with an event is often from a small, discrete set of values (e.g. the presence or absence of a breath; the type of an ECG beat); the time of an event is when it occurred and is not usually periodic. A sequence of Events all describing aspects of the same quality can be considered to be a Signal, usually with irregular clock times.

Events are either associated with a Signal or with a Recording. Core attributes are:

kind — What the event represents.
time — When the event occurred.
duration — The length of the event.

2.1.4 Annotation

An Annotation is a general note or comment about the whole, or some portion, of a Recording, Signal, Event or possibly another Annotation. Annotations are not re-
restricted to textual comments, and could include one or more keywords, or semantic
tags, taken from controlled vocabularies.

Some biosignal file formats (e.g. WFDB) use the name annotation to refer to se-
quences of time-stamped qualities derived from a Signal or Recording — exactly
what is defined above as an Event. BioSignalML considers Annotations to be dif-
f erent from Events, with the Annotation of a temporal region being a more general
concept than that of some Event occurring over the region.

Core attributes of an Annotation are:

- **target** — What the annotation describes.
- **comment** — The text of the annotation.
- **tag** — A semantic tag about the target.
- **creator** — Who made the comment.
- **created** — When it was made.

### 2.1.5 Segment

Annotations are frequently made about temporal regions, or segments, of signals
and recordings. To support this, BioSignalML has the concept of **Segment** — some
portion of a sampled entity, defined by an interval in the sampling dimension. This
simplifies associating other properties, besides annotation, with temporal regions.

Core attributes of a Segment are:

- **source** — The signal or recording which the segment is part of.
- **time** — The temporal interval over which the segment is defined.

### 2.1.6 Timing

Recordings, signals, events, segments and data points all have timing information
associated with them, either as **Instants** and/or **Intervals**. These temporal entities are
likely to be from related, but different coordinate systems. In general, the temporal
systems and their relationships are as shown in Figure 2.1 — a recording is located in physical time, either Coordinated Universal Time (UTC) or local time; a signal may start some time after the recording, specified say as an offset in seconds, and represent times using an integer index and a sampling rate; timing of events might be relative to the recording, a signal, or a sample point (e.g. 0.3 milliseconds before the 12th sample of signal 1).

![Figure 2.1 – A Recording’s temporal systems.](image)

### 2.2 The BioSignalML Ontology

The BioSignalML Ontology is a formal specification of the Abstract Model. Using the Web Ontology Language (OWL), each concept from the model is given:

- a URI to identify it;
- an ontological type — either owl:Class, owl:ObjectProperty, owl:DatatypeProperty, or owl:NamedIndividual;
- a description and possibly a secondary comment.

Equivalence, sub-class, and sub-property relationships are used for concepts that are either equivalent to, or specialisations of, concepts in other ontologies. Properties may also have assertions made about their domain and/or range, and whether the property is functional.
A diagrammatic overview of the BioSignalML ontology is presented in Figure 2.2, with the full version available as RDF at www.biosignalml.org/ontologies/2011/04/biosignalml. A text version of the ontology is in Appendix A.

An brief excerpt from the ontology, serialised as Turtle, is shown in Listing 2.1, in which the concept of Signal is defined (in lines 1-7) by giving it a URI, bsml:Signal, and stating that it represents a class of resources. A human-readable description is also provided, together with the assertion that a bsml:Signal is part of the more general concept, opb:OPB01015 (a Process trajectory, defined in the Ontology of Physics for Biology).

Subsequent lines define bsml:UniformSignal as a sub-class of a bsml:Signal and the bsml:recording property as a function over the class bsml:Signal into the class bsml:Recording.

Listing 2.1 – An excerpt from the BioSignalML Ontology.

```turtle
1  bsml:Signal
2    rdf:type owl:Class ;
3    dct:description
4      """A sequence of periodic measurements of some physical quantity, ordered
5      by some sampling dimension, normally time. A Signal is part of some
6      Recording."""" ;
7    rdfs:subClassOf opb:OPB01015 .

9  bsml:UniformSignal
10   rdfs:subClassOf bsml:Signal ;
11   dct:description "A signal that has been sampled at a constant rate." ;
12   rdf:type owl:Class .

14 bsml:recording
15   rdf:type owl:FunctionalProperty, owl:ObjectProperty ;
16   dct:description "The Recording a Signal is part of." ;
17   rdfs:domain bsml:Signal ;
18   rdfs:range bsml:Recording .
```

---

2 After namespace expansion this is http://www.biosignalml.org/ontologies/2011/04/biosignalml#Signal

3 http://bioportal.bioontology.org/ontologies/47869/?p=terms&conceptid=OPB01015
Figure 2.2 – The BiosignalML Ontology.
The next section looks at how concepts from the BioSignalML Ontology can be used with other ontologies and vocabularies to describe biosignals in terms of the Abstract Model.

## 2.3  Realising the Model

Semantic Web standards and technologies are used in defining the BioSignalML Model. This allows biosignal information to be part of a very general framework, irrespective of storage format, software application, or computing platform.

Several readily available RDFS/OWL ontologies are used by BioSignalML. Any particular ontology is, in many ways, no better or worse than any other that defines similar or equivalent concepts — what is more important is the use of formal, web-accessible definitions. Where possible, I have chosen well-established general ontologies and vocabularies; if not, emerging standards from the biomedical and bioengineering research communities are used.

### 2.3.1  Identity

URIs are used to identify biosignal recordings and their components. In keeping with Linked Data guidelines (Section 1.3.3), URIs should be in the `http` scheme.\(^4\)

BioSignalML implementations should make no assumptions about the structure of any URI — a URI is considered to be opaque, with no information encoded in its path or other components. For readability software may generate URIs following some local convention (e.g. by creating URIs for signals by appending `/signal/N` to a recording’s URI, where `N` is the signal’s index), however no such encoding convention should be assumed nor used to provide information about a resource.

---

\(^4\) An exception is within a repository implementation, with the use of `file:` for resources in the local filesystem.
2.3.2 Generic metadata

For general metadata, BioSignalML uses terms from RDF Schema (RDFS) and the Dublin Core Metadata Initiative (DCMI):

- From RDFS, two properties that can be freely applied to resources are:
  
  \textit{rdfs:comment} — a human-readable description of a resource.
  
  \textit{rdfs:label} — a human-readable name for a resource.

  In particular, \textit{rdfs:label} is used to give labels to Signals.

- From DCMI, the following properties are used:

  \textit{dct:created} — the date and time a Recording was started.
  
  \textit{dct:creator} — the person or group who made a Recording.
  
  \textit{dct:description} — a free-text description of a resource.
  
  \textit{dct:extent} — the duration of a Recording or Signal.
  
  \textit{dct:format} — the format of a Recording.
  
  \textit{dct:source} — the source of a Recording or Signal.
  
  \textit{dct:subject} — the subject of a Recording.

Identifying storage formats

The \textit{dct:format} property is used to specify the format of a Recording. The DCMI Metadata Terms standard recommends values come from a controlled vocabulary, such as the list of Internet Media Types \[103\]. BioSignalML uses the types shown in Table 2.1, which are in the non-standard\footnote{A standardisation process for BioSignalML could register definitive media types with the Internet Assigned Numbers Authority (IANA).} \textit{application/x-} namespace.

2.3.3 Specifying units of measure

A unified biosignal model needs to represent units of measurement in a way that allows new units to be defined in terms of existing ones, and provide sufficient in-
2.3. REALISING THE MODEL

<table>
<thead>
<tr>
<th>Media Type</th>
<th>Recording Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-bsml+hdf5</td>
<td>BioSignalML HDF5 format</td>
</tr>
<tr>
<td>application/x-bsml+edf</td>
<td>European data format (EDF)</td>
</tr>
<tr>
<td>application/x-bsml+wfdb</td>
<td>Waveform database format (WFDB)</td>
</tr>
</tbody>
</table>

Table 2.1 – Internet Media Types specified by BioSignalML.

formation to enable automated conversion. This means the representation must be more than a simple list or controlled vocabulary.

The Ontology for Engineering Mathematics [104] provides a conceptual framework for sharing and reusing models in engineering, including units of measurement. It was an early formalisation of these concepts, and still is an important framework when designing ontologies about physical dimensions and units. However the ontology was implemented in Lisp and has not been converted to OWL, so is not able to be directly used with the Semantic Web.

A number of OWL-based ontologies about units of measurement do exist, including the following. As a measure of their utility, examples are given of their representation of microvolts:

**OBO Unit Ontology** — The Ontology of Units of Measurement [105] is part of the OBO Foundry. It defines both Volt and microvolt as a sub-class of electric potential difference but does not state how to convert between them, nor allow for the derivation of new units.

**QUDT** — Quantities, Units, Dimensions and Data Types in OWL and XML [106] is an ontology designed “...to provide a unified model of measurable quantities, their units, numerical values of quantities, and the data structures and types used to store and manipulate these objects in software”, originally developed as a NASA project. It defines Volt but not microvolt although this could be specified as a derived unit with conversion information.

**UOME List** — The Units of Measurement Expressions, UOME [107], has been developed for use in biological modelling. It has a core vocabulary for defining Unit Expressions, and uses this to extend the OBO Unit Ontology to pro-
provide a comprehensive list of units, intended “…to include every unit in common use, including multiples”. UOME’s design supports the automatic evaluation of nested unit definitions. Microvolt is defined as $10^{-6}$ of a Volt.

Because of UOME’s generality, its ability to define new units in terms of existing ones, and its origins in systems biology, BioSignalML specifies that URIs used for units should either be in, or be derived from those in, the UOME List Ontology. Other ontologies may be used, provided statements are made relating definitions to UOME concepts, so allowing for automatic conversion between compatible units.

Listing 2.2 shows definitions for two units not currently in the UOME List. Lines 1-9 specify `unit:Centilitre` as 0.01 of `uome-list:Litre`, and in lines 11-19, `unit:CentilitrePerMinute` is defined as the quotient of `unit:Centilitre` over `uome-list:Minute`.

```
1 unit:Centilitre
2  uome-core:derivedBy [  
3    uome-core:withFactor 0.01 ;  
4    uome-core:withUnit uome-list:Litre ;  
5    a uome-core:ScalingExpression  
6  ] ;  
7  uome-core:unitName "centilitre"^^xmls:string ;  
8  uome-core:unitSymbol "cl"^^xmls:string ;  
9  a uome-core:UnitOfMeasurement .

11 unit:CentilitrePerMinute
12  uome-core:derivedBy [  
13    uome-core:withUnit1 unit:Centilitre ;  
14    uome-core:withUnit2 uome-list:Minute ;  
15    a uome-core:QuotientExpression  
16  ] ;  
17  uome-core:unitName "centilitre per minute"^^xmls:string ;  
18  uome-core:unitSymbol "cl/min"^^xmls:string ;  
19  a uome-core:UnitOfMeasurement .
```

Listing 2.2 – User specified units of measurement.

### 2.3.4Specifying time

Using concepts from OWL-Time and the Timeline Ontology of Section 1.3.4, several different timelines are required to specify the temporal coordinate systems shown
in Figure 2.1 — a bsml:Recording would have a tl:RelativeTimeLine embedded in a
tl:PhysicalTimeLine, and each bsml:Signal would have a tl:DiscreteTimeLine and associated
 tl:TimeLineMap, relating sample number to elapsed time and corresponding
to our notion of Clock Time. Timing of bsml:Events would then be expressed using
tl:Instants or tl:Intervals on one of these timelines.

TimeLineMaps provide a means to relate different timelines. However their use
is cumbersome and too general purpose for our requirements. For example, Figure 2.3 shows timelines and maps required to fully specify a signal is sampled at
1000Hz.

![Figure 2.3 – Describing 1000Hz sampling with fully specified TimeLineMaps.](image)

In contrast BioSignalML uses a simpler model, shown in Figure 2.4, in which
each signal is considered to have an implicit tl:DiscreteTimeLine and a natural
tl:TimeLineMap. A bsml:Recording still has a tl:RelativeTimeLine, located in Universal
Time via the recording’s dct:created property, and this is used for locating events,
segments, and signals in time. If required, for example when reasoning over an RDF
triplestore, implied timelines and maps might be derived using the Semantic Web
Rule Language, SWRL.

For convenience, the BioSignalML Ontology defines equivalent classes for OWL-
Time’s classes of TemporalEntity, Instant and Interval.
2.3.5 Measurement uncertainty

The BioSignalML Ontology has defined the property `bsml:uncertainty` as “The URI of a resource describing the measurement uncertainty associated with a Recording, Signal, or Segment” in order to allow measurement errors of biosignal data to be described. UncertML\(^6\) is an XML language for describing and exchanging uncertainties [108, 109], which could be used to describe uncertainties associated with biosignal data. This would require an associated ontology, to define a class to represent an UncertML description, and have some uncertainty server return UncertML descriptions when the URI of a description was dereferenced.

Figure 2.5 illustrates how the mean of a ten second long signal segment can be described using BioSignalML and UncertML.

2.3.6 Annotation

From the Oxford English Dictionary, an annotation is “a note by way of explanation or comment added to a text or diagram”,\(^7\) and in BioSignalML an annotation is considered to be an additional note or statement about some resource.

The Open Annotation Community Group’s core data model (from Section 1.3.5) could be used for annotations in BioSignalML. It however appears to be overly complex for our requirements — for example a text annotation about a ten second seg-

\(^6\)http://www.uncertml.org/
\(^7\)http://oxforddictionaries.com/definition/english/annotation
2.3. REALISING THE MODEL

Figure 2.5 – Uncertainty associated with a signal segment.

A note describing a ten second segment of a recording, beginning 60 seconds from the start, would require the RDF graph shown in Figure 2.6.

Figure 2.6 – Using Open Annotation to describe a 10 second segment of a recording.

Instead a simpler model is proposed for BioSignalML, in which an Annotation has a subject (what is being annotated); a comment and/or one or more tags (the note or explanation), and possibly provenance details of how and when the annotation was created. Using this model, the above example has the graph shown in Figure 2.7.
It is important to note that any resource can be annotated, not just Recording, Signal, and Segment resources — annotations may be made about other annotations.

Figure 2.7 – BioSignalML annotation.

Annotating temporal fragments

Temporal regions can be identified in a variety of ways, including:

- Using the URI of the complete resource (i.e. recording or signal) and adding a suffix that specifies the temporal segment. This is the approach used for W3C Media Fragments, where the URI suffix is either a fragment (’#’) or query (’?’) part, formatted as t=begin,end, with begin and end representing times from the start of the resource. For example, http://demo.biosignalml.org/physiobank/mitdb/102#t=60,70 would indicate the interval between 60 and 70 seconds from the beginning of http://demo.biosignalml.org/physiobank/mitdb/102.

A difficulty with using fragment or query parts is that the complete resource can not be easily found by a SPARQL query, as the complete resource’s URI is different to those of its fragments. This can be overcome by performing more
2.3. REALISING THE MODEL

general queries and using FILTER conditions to select particular resources, but this is cumbersome, inefficient, and requires additional processing to extract temporal bounds.

- Having a resource property to directly specify a temporal region over which other properties are valid, for instance having an Interval directly associated with an Annotation. However, when used for temporal annotation, this method conflates the concepts of comment and subject of the comment, making it difficult to separate the annotation from its subject in order to make statements, such as for provenance, about the annotation.

- Using a secondary resource as a target, with separate properties specifying the complete entity and its fragment. This is done in Open Annotation using Fragment Selectors\(^9\) and shown in Figure 2.6 above.

BioSignalML uses this last method to identify fragments of larger temporal resources, as illustrated by the specification of a Segment in Figure 2.7. Although not as general as Open Annotation’s selectors, the Segment concept separates the RDF specification of a temporal region from the the model used for annotation, and is sufficient for annotating and making other statements about temporal fragments.

2.3.7 Provenance

BioSignalML uses the Provenance Vocabulary (also from Section 1.3.5) for describing the provenance of resources, and in particular the \texttt{prv:precededBy} property is used to link the current version of a resource to an earlier revision. For example, when an annotation is updated earlier annotation statements should be kept, with a new statement created and linked to the immediate prior annotation. The Provenance Vocabulary is also used when making statements about collections of RDF statements in a repository, as described in Section 3.1.7.

\(^9\)http://www.openannotation.org/spec/core/#SelectorFragment
2.4 An Example

To illustrate the use of BioSignalML, a real-world dataset is now described. The EDF+ file `ecgca102.edf` from the PhysioBank repository is one of a series of 55 multichannel abdominal non-invasive foetal electrocardiogram (FECG) recordings, taken from a single subject between 21 and 40 weeks of pregnancy. Figure 2.8 shows a segment of signal data displayed in an EDF viewer. The figure has been annotated to show some of the RDF statements describing the recording and signals.

The following listings contain a portion (in Turtle) of the full BioSignalML RDF description for the recording. First, in Listing 2.3, a base URI is given (it is used for the subsequent relative URIs) and the recording and its subject is defined; next, Listing 2.4 shows the specification of a signal in the recording, including details of electrodes used and pre-filtering performed; Listing 2.5 then describes a non-uniform signal, being a sequence of beat annotations produced by PhysioBank's `qrs` annotator; and lastly, Listing 2.6 shows details of an event.

```turtle
@base <http://demo.biosignalml.org/physiobank/> .

<nifecgdb/ecgca102>
  dct:created "2003-11-16T21:49:00"^^xsd:dateTime ;
  dct:extent "PT4M30S"^^xsd:dayTimeDuration ;
  tl:timeline <nifecgdb/ecgca102/timeline> ;
  dct:format "application/x-bsml+edf" ;
  dct:subject <nifecgdb_patient> ;
  pbterms:database <http://physionet.org/physiobank/database/nifecgdb> ;
  a bsml:Recording .

<nifecgdb_patient>
  pbterms:gestationalAge "P155D"^^xsd:dayTimeDuration ;
  pbterms:gender pbterms:Female ;
  a pbterms:Patient .

<nifecgdb/ecgca102.edf/timeline>
  a tl:RelativeTimeLine .
```

Listing 2.3 – A recording described in BioSignalML.

---

10 `http://physionet.org/physiobank/database/nifecgdb/ecgca102.edf`
Figure 2.8 – BioSignalML markup of an EDF+ file.
The recording, as a resource, is identified by a URI with a type of `bsml:Recording`, with its format given as BioSignalML's mime-type for European Data Format. The recording began at 21:49 on November 16, 2003 and is 4 minutes, 30 seconds long. Database specific properties are associated with the recording via the `dct:subject` property identifying a `patient` resource which gives the gestational age and gender of the unborn child, and illustrates how a BioSignalML description can be extended.
The recording has two signal datasets, each with their own resource URI of type `bsml:Signal`; a `label` property naming the signal; and a `units` property specifying their units of measurement.

Because the recording’s URI starts with `http`, a user would expect that dereferencing `http://demo.biosignalml.org/physiobank/nifecgdb/ecgca102` on the Web would return something. In this case they could get back either the actual EDF file, as imported from PhysioBank; a HTML page describing the recording; an RDF description; or signal data streamed using the HTML5 WebSockets protocol, using services provided by a BioSignalML repository and discussed in Chapter 3.

### 2.5 Summary

This chapter described an abstract model for describing biosignals and used it as the basis for the BioSignalML Ontology. The ontology, along with existing general-purpose and specialised ontologies, can then be used to describe biosignal datasets. Descriptions are extensible and able to be integrated with other RDF-based models.

The main ontologies and vocabularies used by BioSignalML are listed in Table 2.2 — some of these are well-established standards, while others are at various stages of specification. A summary of the BioSignalML RDF model is contained in Table 2.3.
— these classes and properties can map a biosignal dataset into the Abstract Model, regardless of its format.

<table>
<thead>
<tr>
<th>Name</th>
<th>Usual Prefix</th>
<th>URI and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dublin Core Metadata Terms</td>
<td>dct:</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies properties and classes applicable to a wide range of resources. Properties include creator, date, description, format, provenance and title [110].</td>
</tr>
<tr>
<td>OWL-Time</td>
<td>time:</td>
<td><a href="http://www.w3.org/2006/time#">http://www.w3.org/2006/time#</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defines temporal concepts including Instant and Interval along with relationships between them, such as before, hasBeginning and intervalOverlaps [69].</td>
</tr>
<tr>
<td>The Provenance Vocabulary</td>
<td>prv:</td>
<td><a href="http://purl.org/net/provenance/ns">http://purl.org/net/provenance/ns</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Provenance Vocabulary is a domain specific specialisation of the W3C PROV Ontology (PROV-O) providing Web data specific extensions.</td>
</tr>
<tr>
<td>RDF Schema</td>
<td>rdfs:</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RDF Schema is a general-purpose language for representing information about Web resources.</td>
</tr>
<tr>
<td>Timeline Ontology</td>
<td>tl:</td>
<td><a href="http://purl.org/NET/c4dm/timeline.owl#">http://purl.org/NET/c4dm/timeline.owl#</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defines the TimeLine concept as a coherent backbone for addressing temporal information [70].</td>
</tr>
<tr>
<td>Units of Measurements (UOME) List</td>
<td>uome-list:</td>
<td><a href="http://www.sbpax.org/uome/list.owl#">http://www.sbpax.org/uome/list.owl#</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Every unit in common use, specified using terms from the OBO Unit Ontology, and designed for automatic evaluation of nested unit expressions [107].</td>
</tr>
</tbody>
</table>

**Table 2.2** – RDFS and OWL ontologies used by BioSignalML.

The use of URIs to identify biosignal components means information about a component can be provided when its URI is dereferenced. The next chapter describes software for doing so, consisting of a core library implementing the Abstract Model, a web-based repository for storing resources, and tools to access the repository. Also described is an HDF5 file layout for the efficient storage of signal recordings.
<table>
<thead>
<tr>
<th>Abstract Concept</th>
<th>OWL Class</th>
<th>Attribute</th>
<th>Property</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common attributes</td>
<td></td>
<td>label</td>
<td>rdfs:label</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>comment</td>
<td>rdfs:comment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>description</td>
<td>dct:description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>precededBy</td>
<td>prv:precededBy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>creator</td>
<td>dct:creator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>created</td>
<td>dct:created</td>
<td>xsd:dateTime</td>
</tr>
<tr>
<td>Recording</td>
<td>bsml:Recording</td>
<td>investigation</td>
<td>dct:subject</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td></td>
<td></td>
<td>investigator</td>
<td>dct:creator</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format</td>
<td>dct:format</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataset</td>
<td>bsml:dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>source</td>
<td>dct:source</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>starttime</td>
<td>dct:created</td>
<td>xsd:dateTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>duration</td>
<td>dct:extent</td>
<td>xsd:dayTimeDuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>timeline</td>
<td>tl:timeline</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>generatedBy</td>
<td>prv:wasGeneratedBy</td>
<td></td>
</tr>
<tr>
<td>Signal</td>
<td>bsml:Signal</td>
<td>recording</td>
<td>bsml:recording</td>
<td>bsml:Recording</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clock</td>
<td>bsml:clock</td>
<td>bsml:SampleClock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rate</td>
<td>bsml:rate</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>period</td>
<td>bsml:period</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>units</td>
<td>bsml:units</td>
<td>uome-core:UnitOfMeasurement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sensor</td>
<td>bsml:sensor</td>
<td>bsml:Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prefilter</td>
<td>bsml:preFilter</td>
<td>bsml:Filter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataBits</td>
<td>bsml:dataBits</td>
<td>xsd:integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>minFrequency</td>
<td>bsml:minFrequency</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxFrequency</td>
<td>bsml:maxFrequency</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>minValue</td>
<td>bsml:minValue</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxValue</td>
<td>bsml:maxValue</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signaltype</td>
<td>bsml:signalType</td>
<td>bsml:SignalType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>offset</td>
<td>bsml:offset</td>
<td>xsd:dayTimeDuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>duration</td>
<td>dct:extent</td>
<td>xsd:dayTimeDuration</td>
</tr>
<tr>
<td>Event</td>
<td>bsml:Event</td>
<td>recording</td>
<td>bsml:recording</td>
<td>bsml:Recording</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eventtype</td>
<td>bsml:eventType</td>
<td>bsml:EventType</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>bsml:time</td>
<td>bsml:TemporalEntity</td>
</tr>
<tr>
<td>Annotation</td>
<td>bsml:Annotation</td>
<td>about</td>
<td>dct:subject</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tags</td>
<td>bsml:tag</td>
<td>owl:Individual</td>
</tr>
<tr>
<td>Segment</td>
<td>bsml:Segment</td>
<td>source</td>
<td>dct:source</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>bsml:time</td>
<td>bsml:TemporalEntity</td>
</tr>
<tr>
<td>Clock Times</td>
<td>bsml:SampleClock</td>
<td>resolution</td>
<td>bsml:resolution</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rate</td>
<td>bsml:rate</td>
<td>xsd:double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>period</td>
<td>bsml:period</td>
<td>xsd:double</td>
</tr>
<tr>
<td>Temporal entity</td>
<td>bsml:Interval</td>
<td>timeline</td>
<td>tl:timeline</td>
<td>tl:TimeLine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>start</td>
<td>tl:start</td>
<td>xsd:dayTimeDuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>duration</td>
<td>tl:duration</td>
<td>xsd:dayTimeDuration</td>
</tr>
<tr>
<td>Temporal entity</td>
<td>bsml:Instant</td>
<td>timeline</td>
<td>tl:timeline</td>
<td>tl:TimeLine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>start</td>
<td>tl:at</td>
<td>xsd:dayTimeDuration</td>
</tr>
</tbody>
</table>

Table 2.3 – Abstract Model mapped to ontologies.
Chapter 3

BioSignalML Software

An abstract model by itself is of limited use to researchers working with biosignals, unless their expertise includes software development and the Semantic Web. Instead the BioSignalML model has been implemented in software, and is available as a programming library, a web-based BioSignalML repository, and a new HDF5 file layout in which to store recordings.

This chapter describes a realisation of the BioSignalML Abstract Model. Section 3.1 describes the repository structure and functionality, and provides examples of its use. The repository is built on top of the BioSignalML library, described in Section 3.2, which also has functions to allow user applications to work with remote repositories. Next the HDF5 file layout is described, followed by a discussion of why the particular system architecture, software resources, and tools used in the implementation were chosen.

3.1 The BioSignalML Repository

A BioSignalML Repository is a web-based application providing a repository for BioSignalML resources. It offers services for storing, browsing, querying, and ob-
taining BioSignalML resource information. Interfaces to a repository are as shown in Figure 3.1.

Each resource in a repository has its own URI. This locates the resource both in the repository and on the Web. When a resource’s URI is dereferenced some representation of the resource is returned, which, depending on the type of content requested, could be a web page describing the resource, an RDF description with links to related resources, a data file, or a stream of signal data.

Within the repository, recordings are kept in their original file format, or if a new recording is streamed in, as HDF5. All metadata pertaining to a recording is

Figure 3.1 – Key repository interfaces. HTTP services provide HTML, RDF, file and stream representations of resources.
extracted during import and held in a Named Graph [111], stored in an external triplestore. Different versions of the same recording have separate graphs and signal data files, described and linked with provenance statements. Normally a request for a resource (Recording, Signal, Event, Annotation, etc) from a repository results in the most recent version being accessed, although with suitable queries older versions may be retrieved.

Web browsers and user applications interact with the repository by connecting to it using standard HTTP methods. The repository has two specific endpoints providing different services:

- `/frontend/` serves HTML web pages, allowing users to browse and query a repository using standard web browsers.
- `/sparql/` provides a SPARQL service, allowing queries to be made against a repository.

All other URIs refer directly to BioSignalML resources and are used to store and retrieve both RDF and signal data.

### 3.1.1 Naming of resources

The repository places restrictions on the URIs of resources it manages:

- The URI must use the `http` scheme, in accordance with Linked Data guidelines.
- The hostname component must be identical to the repository’s domain name. This ensures that dereferencing the URI will result in a request being made to the repository.
- The first component of a resource URI’s path must be different from either of the above endpoint names.
3.1.2 Controlling access

It is important that access to a repository is restricted to suitably authorised individuals, especially on the general Internet — not all content may be publicly available; only some users or applications may be allowed to add content or annotation; and so forth.

Several standard ways exist to authenticate web services. These range from providing a name and password, as used for Basic HTTP Authentication [112], to the exchange of tokens over a secure channel, such as in OpenID [113] or SAML.\(^1\) Such authentication schemes are usually implemented as an additional wrapper around a web-service.

A BioSignalML repository uses a simple, token-based, authentication system. The repository provides a web-service\(^2\) that returns access tokens upon receipt of a valid username and password, using a database to hold user details, including privilege level, current token, and its expiry time.

To authenticate, a user supplies their username and password via a command-line utility, which in turn saves the returned token in a local file. Library code then passes this token when requests are made to the repository, which then checks validity and user privilege level before permitting (or declining) the requested function. User privilege levels allow view, extension, modification, deletion, and full administration of a repository’s contents.

3.1.3 Accessing resources

The repository application uses an architectural style called Representational State Transfer (REST) [114] to exchange BioSignalML resources over the Internet. Requests for resources specify a URI, HTTP method, and mime-type, and these determine the result, using a process known as content-negotiation:

\(^1\)http://saml.xml.org/saml-specifications
\(^2\)As part of the /frontend/ service.
• A GET method returns some representation of the resource, as determined by
the `Accept` attribute. Standard mime-types for RDF/XML, RDF/JSON, and Turtle serialisations, along with application specific types for EDF, WFDB, HDF5 and raw (binary) recordings, are supported.

• A PUT method results in new resources being created, or existing ones replaced, with the `Content-type` attribute specifying the data format, using the same mime-types as GET.

• A POST method results in an existing resource being extended with additional information. `Content-type` and mime-types are as for PUT.

• A DELETE method will remove resources from the repository.

The authentication process, described above, takes the HTTP method into account when determining the level of access.

Besides these standard HTTP methods, a request to a repository may instead be to use the WebSocket Protocol, which will result in signal data transfer over a socket connection. This is covered in Section 3.1.4.

Recordings may be directly exchanged with a repository using standard Unix command line tools such as `curl`,\(^3\) or by an application using the `libcurl` library. If authentication is enabled in the repository then a valid access token must be supplied as a `cookie` with the HTTP request — for simplicity this is omitted from the following example.

Line 4 of Listing 3.1 shows the command to import a recording into a repository and give it a URI of `http://demo.biosignalml.org/example/sinewave`. The command on line 8 retrieves this recording and saves it with a different name. Then on line 11, the retrieved recording is verified as identical to the original. Line 16 requests an RDF representation of a signal, with the result shown on lines 18-27. RDF browsers, such as Tabulator [115], are also able to directly display recording and signal metadata from a repository, as shown in Figure 3.2.

\(^3\)http://curl.haxx.se
# If authenticating, then "--cookie access=<token>" must be part of curl request.

# Import the file 'sinewave.edf'
$ curl -T sinewave.edf -H "Content-type: application/x-bsml+edf" http://demo.biosignalml.org/example/sinewave

201: Created http://demo.biosignalml.org/example/sinewave application/x-bsml+edf

# Retrieve http://demo.biosignalml.org/example/sinewave as an EDF file.
$ curl -H "Accept: application/x-bsml+edf" -o output.edf http://demo.biosignalml.org/example/sinewave

# And verify what came back matched what went in:
$ diff -s <(xxd sinewave.edf) <(xxd output.edf)
Files /dev/fd/63 and /dev/fd/62 are identical

# Now request RDF describing a signal in the recording:
$ curl -H "Accept: text/turtle" http://demo.biosignalml.org/example/sinewave/signal/0

@base <http://demo.biosignalml.org>
</example/sinewave>
a bsml:Recording .
</example/sinewave/signal/0>
bsml:index 0 ;
bsml:rate 100.0 ;
bsml:recording </example/sinewave> ;
bsml:units units:Microvolt ;
a bsml:Signal ;
rdfs:label "Sine\_wave" .

Listing 3.1 – Importing and retrieving a recording using curl.

### 3.1.4 Accessing signal data

Requests to a repository for signal data use the Upgrade HTTP header attribute to request the use of the WebSocket Protocol [116] instead of the usual exchange of HTTP content — WebSockets is designed to transport or stream binary data across the Internet using HTTP without requiring special network configuration or other administration.

Segments of signal data are encapsulated in blocks, using a Block Stream message format I have designed, before sending them to a remote application. Data transfer is bi-directional, allowing applications to send data to a repository as well as receive data. The Block Stream message format has also been implemented in
### 3.1. THE BIOSIGNALML REPOSITORY

#### Javascript, so allowing the efficient transfer of data to a web-browser and enabling the future development of applications such as a browser-based signal viewer. Appendix B describes the Block Stream message format.

#### Data transfer performance

The throughput of the Block Stream format was tested by retrieving signal data over the Internet via an ADSL2+ connection to a remote repository. 500,000 samples\(^4\) from

---

\(^4\)A 5000 second segment at 100 samples/second, transferred in ten blocks of 50,000 samples, using 64-bit double precision values.
a single signal can be repeatedly downloaded in under 3.2 seconds, giving a data rate of around 9.5 Mbps; for comparison, the raw download rate, measured by broadband speed tests, is between 11.5 and 12.5 Mbps.

If instead, ten separate segments each of 50,000 samples\(^5\) are retrieved, the time increases to around 6 seconds. This suggests an overhead of approximately 250 milliseconds per signal for setup. This will include a server component (for metadata retrieval and file opening) and a network component (for connection establishment etc). Downloading ten small segments, each of 100 samples, confirmed this overhead as the total retrieval time was around 2.4 seconds. Caching within the repository could be used to reduce the server overhead.

### 3.1.5 Web browser endpoint

The `/frontend/` endpoint allows a web-browser to be used with a BioSignalML repository. A user can:

- **Browse** — Recordings are presented in a tree view, that may be expanded and collapsed. Holding the mouse pointer (hovering) over a recording URI in the tree shows a summary of the recording’s metadata; clicking on a URI results in a fuller display showing details signals and allowing annotations to be created and viewed, as shown in Figure 3.3.

- **Search** — Text and metadata terms can be queried using a search form that allows terms to be combined using Boolean operators. Results are returned as a table with URIs matching objects; hovering over a URI shows a summary of the object’s metadata; clicking on it results in a fuller display.

- **Query RDF** — Users can issue SPARQL queries against RDF metadata in a repository, with results presented in a similar way to those from a search.

- **Browse RDF** — Users can browse RDF metadata by using SNORQL\(^6\) to explore a repository’s SPARQL endpoint, as shown in Figure 3.4.

\(^5\)From different signals, all sampled at 100 samples/second.

\(^6\)https://github.com/kurtjx/SNORQL
3.1. THE BIOSIGNALML REPOSITORY

3.1.6 SPARQL endpoint

The /sparql/ endpoint allows Semantic Web software to access RDF in a repository. The BioSignalML library uses this endpoint when obtaining information for client applications. Command line tools such as roquet\(^7\) can also be used to query the endpoint, as shown in Listing 3.2, where some provenance information has been obtained about http://demo.biosignalml.org/calcium/bursting/data.

\(^7\)http://librdf.org/rasqal/roquet.html
Provenance information about recordings is kept in a separate named graph using the Provenance Vocabulary. This tracks when recordings are entered into a repository, not actual provenance about the recording’s signal collection, as this is considered to be direct metadata about the recording that would normally would be kept with the recording in its metadata graph.
When a recording is imported into a repository, a new named graph (to hold metadata) and data file (for signal data) is allocated and statements describing their creation are added to the repository’s provenance graph. If the recording was already in the repository these statements include a link to the preceding version. When accessing recordings and associated resources the most recent revision of the recording’s named graph is normally used to obtain metadata. Figure 3.5 illustrates the addition of `<mitdb/102>` to a repository showing when it was added and by whom, and that a previous version already existed — a SPQARL query may be used to obtain details of the earlier versions.

![Figure 3.5 – Provenance of a BioSignalML recording.](image)

### 3.2 BioSignalML Library

The BioSignalML library allows application software to work with the Abstract Model and to access remote repositories. The library is implemented in Python and provides an object-oriented model for working with BioSignalML resources. The core part of the library is structured as shown in Figure 3.6.

---

8As identified by its URI.
3.2.1 Metadata mapping

To support the use of RDF to store metadata held in program variables, the relationships between an attribute of a software object and the corresponding ontological property is specified in mapping tables. These tables can be extended and overridden, to allow for user and domain-specific metadata, and the development and refinement of ontologies. Classes and properties from the BioSignalML Ontology and other vocabularies and ontologies are represented by class variables — for instance the Python class variable `DCT.created` is defined as equivalent to the URI `http://purl.org/dc/terms/created` (which is normally abbreviated as `dct:created`).

Listing 3.3 shows the mapping table for the AbstractObject class. For this class, the `created` attribute corresponds to the `dct:created` property, RDF values have a type of `xsd:datetime`, and helper functions are specified to convert between an internal `datetime` format and an ISO representation.
3.2. BIOSIGNALML LIBRARY

Listing 3.3 – Mapping table for AbstractObject.

3.2.2 Abstract Model implementation

The AbstractObject class is used as a base to defining Python classes corresponding to the ontological classes of the Abstract Model. These Python classes are shown in Figure 3.7, where their relationships are described in Unified Modelling Language (UML) [117] notation. This figure is comparable to Figure 2.2 from the previous chapter, which shows similar relationships between the ontological classes.

Concrete objects are realised by extending these abstract classes to incorporate actual RDF metadata and signal data, obtained either from SPARQL queries and data files, or for new objects, specified by an user application.
3.2.3 Data abstraction

An abstraction layer is used to support different data file formats, with interface modules converting between native formats and the common data abstraction, including from native representations of metadata to the BioSignalML RDF standard. Format specific metadata is supported by defining new classes (as extensions of core classes) to provide appropriate attributes and mapping table entries.

The library provides supports for EDF and EDF+, WFDB, and the BioSignalML HDF5 format — other formats require interface software to be developed.

3.2.4 Repository support

A number of classes are provided to support the development of BioSignalML repositories. These include:

- A provenance aware store for named RDF graphs, implemented over the SPARQL endpoint of an RDF store.
- Units-of measurement checking and value conversion, for units defined using the UOME ontologies.
- Sample rate conversion for uniformly sampled signals, using an external library.
- An implementation of the Block Stream protocol, used to send signal data over WebSocket connections.

3.2.5 Accessing remote repositories

The library provides classes to allow application programs to access and create BioSignalML resources in remote repositories. This interface is structured in a similar way to the core library (Figure 3.6), with metadata obtained by querying a repository’s SPARQL endpoint and data transferred using a Block Stream.
3.3 HDF5 Signal Storage

An HDF5 file layout has been defined in order to provide a domain-independent biosignal storage format that is compatible with a wide range of applications and tools. The structure of a BioSignalML HDF5 file is shown in Figure 3.8 — at the top level it has a metadata dataset containing serialised RDF, a recording group with subgroups containing signal and timing datasets, and a uris group that is used to locate datasets by URI.

The example shows part of the second signal dataset for PhysioBank’s mitdb/102 recording, which has 650,000 sample points. The core signal properties of rate, units and uri, along with numeric scaling factors, are held as dataset attributes; these core properties are also stored in RDF, as seen in the metadata text.

![Figure 3.8 – Structure of a BioSignalML HDF5 file.](image)

A full specification of the format is available as Appendix C — besides uniformly sampled data it supports irregularly sampled signals; discontinuous signals; and signals containing multiple channels sharing the same timing.
3.3.1 Performance

Using the BioSignalML HDF5 layout to store PhysioBank WFDB recordings (see Chapter 4) provides an opportunity to compare the performance of the HDF5 and WFDB formats. HDF5 has options for data compression, and by default we use the Szip\(^9\) option. This was enabled in the following tests.

For small to medium recording files the two formats are comparable in their use of disk space. The \texttt{mitdb} database has 48 recordings and occupies 90MB of space, with each recording being just under 2MB in size; the same database as a set of HDF5 files occupies 98MB of disk space.

However for large signal datasets HDF5 can provide significant savings in disk space. The \texttt{mghdb} database has 250 recordings that are each around ten times larger than those in \texttt{mitdb}, totalling 4.2GB of disk in WFDB format; this reduces to 2.3GB when stored using HDF5.

To compare sequential access to signal data between the two formats, programs were written in C to read all eight signals in each of the first ten \texttt{mghdb} recordings and calculate simple statistics. To see how the WFDB library handled the independent reading of signals, two WFDB versions were produced — one which read all of a recording’s signal values at a sample time before moving to the next sample point (the \textit{parallel} version); the other which read all data for one signal before advancing to the next signal (the \textit{serial} version).\(^10\) A variant of the programs read a single signal from each recording.

The total execution times of the different versions was measured using the Unix \texttt{time} utility,\(^11\) and are as shown in Table 3.1.

As can be seen, when all signals are read from a recording the read performance of the HDF5 layout is similar to WFDB’s, provided signal data values are read in

\(^9\)http://www.hdfgroup.org/doc_resource/SZIP/

\(^{10}\)Our reading of HDF5 datasets is inherently serial.

\(^{11}\)All testing was performed on a lightly loaded OS/X 10.6.8 system with a 2.33 GHz Intel Core 2 Duo processor, 3 GB RAM and solid-state disk drive.
3.4. SYSTEM DESIGN AND IMPLEMENTATION

<table>
<thead>
<tr>
<th></th>
<th>WFDB Parallel</th>
<th>WFDB Serial</th>
<th>HDF5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eight signals:</td>
<td>4.9</td>
<td>23.4</td>
<td>5.0</td>
</tr>
<tr>
<td>One signal:</td>
<td>2.9</td>
<td>2.9</td>
<td>0.65</td>
</tr>
</tbody>
</table>

*Table 3.1 – Time, in seconds, to read data from HDF5 and WFDB formatted recordings.*

Parallel; if signals are accessed independently WFDB is significantly slower. If only a single signal is read, both WFDB versions are comparable (the time is about one-eighth of serially reading eight signals, as expected); HDF5 is significantly faster (again, about one-eighth of the time to read eight signals).

While by no means a detailed analysis, these simple measures show an HDF5 based signal format can offer significant benefits when compared with WFDB.

3.4 System Design and Implementation

The software described in this chapter is written using Python and makes use of several open-source packages. It is designed to be used as a client/server web-resource although will also run locally on a workstation under any modern, multi-tasking, operating system. The following gives reasons for this design and provides details of the main third-party packages used.

3.4.1 Web based

The World Wide Web and Internet has become a major platform on which distributed, heterogeneous applications are built and deployed. Because of this BioSignalML software has been designed using current and emerging Web standards. This should ensure BioSignalML applications remain compatible with future Web development and computing environments.

Besides Semantic Web standards, upon which BioSignalML is based, and the use of web-languages (HTML, CSS and JavaScript), content-negotiation and the
REST architectural style is used to exchange BioSignalML resources (detailed in Section 3.1.3), and the WebSocket Protocol (described in Section 3.1.4) used to stream binary signal data.

3.4.2 Developed in Python

The main requirement I had for a programming language was that it must directly support object-oriented programming, so as to allow abstract concepts to be implemented with minimal program overhead. My other requirements were that it should provide a high level, modular framework; have the ability to work with other languages; and have mature libraries for web development. Of the object-oriented programming languages in common use, a few stood out:

**C++**: A popular, general-purpose, statically-typed\(^{12}\) language, C++ [118] was first released in 1983 as an enhancement to C, and has many mature and well-supported libraries available. However, because it is not widely used for web development, it does not have a large range of tools and libraries supporting this.

**Java**: Java [119] was first released in 1995. Its syntax is similar to C and C++, but simpler than C++ for specifying objects. Like C++, Java is also statically-typed. A large number of third-party libraries and packages are available to Java users, including several well established packages for general and Semantic Web development.

**Python**: First released in 1991, Python [120] is a dynamically-typed\(^{13}\) language directly supporting high-level data structures and containers (e.g. lists, dictionaries, and sets), with which it is easy to write succinct code. Python has many mature, third-party packages available, including several web frameworks.

\(^{12}\)The types of all variables must be declared at compile time.

\(^{13}\)The type of a variable is not pre-declared but instead is determined by the type of value assigned to it at run time.
3.4. SYSTEM DESIGN AND IMPLEMENTATION

Ruby: Although not as widely used as the above languages, Ruby [121] is a popular language for web applications. Ruby dates from 1995, and like Python, is dynamically-typed.

Of these, Python was selected — it allows clear, concise programming; is well supported; and is widely used for web applications.

Using BioSignalML from C

The majority of programming languages are able to use code developed using C — everything from Fortran, C++, Ruby, Perl, Java, Python, etc. to Matlab, Octave, Labview and the R statistical language — so it is imperative that there is some way to access the BioSignalML framework from C. Although it is straightforward to use C from Python, the reverse is non-trivial because it usually means having to embed a Python interpreter in an application.

The next phase of development is to write a C++ library to enable more applications to make use of BioSignalML repositories; this development will include code to allow its use from C, and eventually would provide common library code for use with Python.

3.4.3 Open-source packages used

Several freely available open-source software packages are used in the BioSignalML software implementation. These include:

Redland RDF libraries: Manipulation and querying of RDF graphs is intrinsic to both a repository and client libraries. Redland\(^{14}\) [122] is a mature and stable C library that can be used with Python; other open-source RDF libraries either don’t support both Python and C, or are not so well documented and supported.

\(^{14}\)http://librdf.org/
**Virtuoso:** Although Redland can directly store RDF it is not as efficient as standalone triplestores, which are designed to work with billions of triples, and have the advantage of separating the store’s resource requirements from that of a repository. To be suitable for our use, a store must support SPARQL 1.1 Update [58] and provide adequate query performance. The top two performers in the most recent run\(^\text{15}\) of the Berlin SPARQL Benchmark [123] were considered:

**4store:** 4store [124] is a freely available open-source RDF database and SPARQL engine,\(^\text{16}\) with ongoing work to integrate reasoning [125].

**Virtuoso:** Virtuoso [126] is a large package providing several different data management functions including RDF and SPARQL services, and available in both commercial and open-source versions.\(^\text{17}\)

I initially chose 4store for repository implementation. However, as development proceeded it became obvious that 4store’s response time for queries containing a `UNION` graph pattern was impacting repository performance. A trial of Virtuoso found it was noticeably faster, so its use is now recommended. The use of Virtuoso has the added benefit of a more comprehensive free-text search facility than that provided by 4store.

**NumPy:** NumPy [127] is a scientific computing package for Python that provides efficient storage of numerical arrays.

**h5py:** h5py\(^\text{18}\) provides a simple, general-purpose Python interface to the HDF5 scientific file format.

**Tornado:** There are many Python web-server frameworks. Of them, Tornado\(^\text{19}\) is simple and easy to use, has direct support for WebSockets and authentication systems, and is a high-performing, production capable server [128].

---

\(^{15}\)http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V6/
\(^{16}\)http://4store.org/trac/wiki/Documentation
\(^{17}\)http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/
\(^{18}\)http://alfven.org/wp/hdf5-for-python/
\(^{19}\)http://www.tornadoweb.org/
jQuery: Of the various libraries available which simplify the use of Javascript in web-browsers and hide their idiosyncrasies, jQuery\textsuperscript{20} is a mature library with good support.

pint: Pint\textsuperscript{21} is a Python package for defining and manipulating units of measurement. It is used as the basis for unit conversion, with my code parsing unit definitions (made using the UOME ontology) to generate instances of Pint objects, that can then be compared.

libsamplerate: The Secret Rabbit Code\textsuperscript{22} library, libsamplerate, is used to perform sample rate conversion when signals are streamed from a repository, with a sinc based convertor (SRC\_SINC\_MEDIUM\_QUALITY) providing fast, good-quality conversion. Possible future work is to compare libsamplerate’s performance with other sample rate convertors — it is straightforward to use another convertor, due to the modular nature of the BioSignalML library.

\section*{3.5 Summary}

This chapter has described a web-based, client/server repository for BioSignalML resources in which data and metadata are of equal importance, along with the BioSignalML library, used by the repository and available to user applications. Current versions of the library and repository are available on GitHub, at https://github.com/dbrnz/biosignalml-corelib and https://github.com/dbrnz/biosignalml-server respectively, with the library also available from the Python Package Index (PyPI) as the BioSignalML package.\textsuperscript{23}

Other software developed as part of this thesis includes a standalone application to browse, annotate and visualise recordings, and utilities for repository man-

\textsuperscript{20}http://jquery.com/
\textsuperscript{21}https://github.com/hgrecco/pint
\textsuperscript{22}http://www.mega-nerd.com/SRC/
\textsuperscript{23}https://pypi.python.org/pypi/BioSignalML
agement, data migration, and streaming data. Some of these packages are currently available, as described in Appendix D. All software has been released under an open-source license.

The next two chapters look at ways in which BioSignalML can be used, first by providing a framework to work with metadata, and then as a source of data.
Chapter 4

Applying BioSignalML to PhysioBank

This chapter describes how BioSignalML might be applied to PhysioBank, a public archive of physiological signal recordings. The use of BioSignalML provides a number of benefits — existing metadata is converted to use standard vocabularies and checked for consistency; annotation tools enable users to assist with curation; a provenance layer is available; and resources in the archive are accessible to Semantic Web search tools and able to become part of the Linked Data web.

The following first describes the PhysioBank resource and highlights a number of limitations with its current implementation, before discussing what is required to validate any conversion process. A conversion process is next described and illustrated using examples from actual PhysioBank recordings. The chapter concludes with a summary discussing how current limitations have been addressed.

4.1 PhysioBank

PhysioBank\(^1\) [27] is a public Web repository that provides format independent access to physiological signals. It currently contains over 36,000 recordings or records, organised into more than 50 databases or collections, and each month attracts around

\(^1\text{http://physionet.org/physiobank}\)
45,000 visitors who retrieve about 4 terabytes of data. Each recording in the repository is usually made up of several data signals and one or more annotation sets (where an annotation set is a temporal sequence of events describing some attribute or measure, for example beat annotations derived from ECG signals). Most collections of recordings are stored using the WaveForm DataBase (WFDB) format, although some collections are stored in European Data Format (EDF) and made available as WFDB through a software library.

The PhysioBank Index\(^2\) provides a master index to the repository. It describes all recordings, signals and annotation sets, and some details about the subject of a recording.

The repository and its recordings can be accessed in various ways:

- **Via a web-browser:**
  - The Archive Index page\(^3\) summarises each of the different databases and provides links to their full descriptions. These pages include a directory of a database’s recordings, from which individual files may be downloaded.
  - The Automated Teller Machine,\(^4\) ATM, displays individual recordings and annotations, histogram and summary information, and allows portions of recordings to be downloaded.
  - The Record Search\(^5\) screen allows complex searches to be made against the master index to find recordings; they can then be viewed and downloaded using the ATM.
  - The entire text of the web site can be searched to find databases and recordings.

- **Via command line tools on a user’s computer** — utilities are provided to search the master index and to selectively download recordings and annotations.

- **By using the WFDB library** — this is the recommended way for user applications to access recordings, as storage format details are hidden. The library

\(^2\)http://physionet.org/physiobank/database/pbi
\(^3\)http://physionet.org/physiobank/database
\(^4\)http://physionet.org/cgi-bin/atm/ATM
\(^5\)http://physionet.org/cgi-bin/pbs/pbsearch
provides access to recordings in both the user’s local filesystem and in remote PhysioBank repositories.

4.1.1 Limitations

Even though PhysioBank is a widely used Web resource it has a number of restrictions and limitations, especially when considered in terms of Web evolution and multicore/multitasking personal computers:

- It is not a Semantic Web resource — the repository provides HTML web pages and recording files without semantic markup, and offers no RDF representations of its resources.
- Standard vocabularies and ontologies are not used for all metadata terms, leading for instance, to inconsistencies with units of measurement.\(^6\)
- Recording metadata contained in the PhysioBank Index is in free text, often with an undocumented, database-specific structure.
- Recordings and metadata are updated as inconsistencies are found. However the curation process does not result in public provenance and version information.
- The WFDB library uses static variables and so cannot be considered to be thread safe.
- The library’s documentation describes how to have more than one recording open but warns against multiple recordings having different sample rates. There is no ability to specify the recording from which to read signals — the two functions provided to read data:

\[
\begin{align*}
\text{int } \text{getvec}(\text{WFDB\_Sample } * \text{vector}) \\
\text{int } \text{getframe}(\text{WFDB\_Sample } * \text{vector})
\end{align*}
\]

read from the last opened recording.

\(^6\)These have since been rectified by the curator following my notification.
• Access to an individual signal is via a vector of data points of all signals in a recording, for each sample time. As seen in Section 3.3.1, there is no efficient way to read signals independently.

• Not all information held in the repository is available via the WFDB library or PhysioNet tools.

• PhysioBank’s Record Search screen allows recordings to be found using information held in the Index, including the type of an annotation event. The Index though does not contain information about where an event is located in a recording — a user has to search the annotation signal to find an event’s location.

These limitations can be overcome by encapsulating PhysioBank recordings in BioSignalML. This will allow more general tools to be used to find and extract information, and provide concurrent access to recordings from multi-threaded applications. The following sections show how this can be done.

4.2 Software Tools

Two different types of software tools are required for converting PhysioBank recordings to use BioSignalML — one is software that does the conversion, development of which is described in Section 4.3; the other type are tools to query and examine the resulting BioSignalML recordings to verify the conversion process and facilitate ongoing use, annotation, and curation of recordings after conversion.

Figure 4.1 shows a verification process for conversion. This requires software to allow users to:

• browse, query, and examine BioSignalML recordings, including signal visualisation.

• export recordings and selected subsets of them.

• comment and make annotations about recordings and their component signals and events.
4.2. SOFTWARE TOOLS

- keep a history of changes in the form of provenance records.

In the following, this software is referred to as a **Browser Tool** as I expand upon its requirements and illustrate its use.

Interaction with a Browser Tool is usually to obtain information about something of interest. In typical use cases, the user will either:

1. have a general interest and wish to browse and explore signals/recordings in a collection;
2. or have a set of criteria they are looking for and want to perform some query to find matching signals/recordings in a collection;
3. or know specifically what signals/recordings they want and have identifiers (URIs) referencing them.

In terms of the BioSignalML abstract model from Section 2.1, a signal collection consists of **Recording**, **Signal**, **Event** and **Annotation** resources, with possibly associated resources such as **Patients** (the subject of recordings), and **Databases** (groups
of related recordings). A Browser Tool must be aware of these core classes and allow users to select and sort within them.

By definition, resources of class **Recording** hold data and other information captured during signal acquisition and subsequent processing — a recording provides context (data, temporal, and metadata) for other associated entities. Consequently it is recordings that are considered to be the fundamental type of entity held in a collection, and they (and their temporal segments) are the primary objects queried, viewed and annotated with a Browser Tool.

### 4.2.1 Querying a recording collection

A Browser Tool could be expected to present a user with at least two query interfaces: one for simple or basic searches; the other for advanced searches. An example entry form for simple queries is depicted in Figure 4.2, which shows a query for finding recordings with annotation text containing either the words *PVC*, *PVCs*, or the phrase *premature ventricular*, and also those recordings with *pbank:pvcBeat* events; results from the query will be summarised to show frequency counts instead of individual event details.

![Image of a simple query user interface.](image)

**Figure 4.2** – Simple query user interface.
Simple queries are made up of search terms combined using the \textit{AND}, \textit{OR}, and \textit{AND NOT} operators, where each search term is a triple in the form:

\[(<\text{property}>, <\text{relation}>, <\text{value}>)\]

which finds items that have \textit{<property>} values with the given \textit{<relation>} to the supplied \textit{<value>}

Property pulldown lists for terms are defined by configuration settings which specify property URIs and labels, possible comparison relationships, and how to generate SPARQL. When a query is submitted it is translated to SPARQL and executed against RDF held for the collection of recordings. Queries may also be saved for later reuse; an advanced search form allows SPARQL queries to be directly entered and executed.

The result of a query is a set of URIs of recordings that contain resources that satisfy the search conditions, along with resource URIs, type of resource, details of the property and matching value, and optionally, summary counts. Temporal offsets of resources that have an associated start time are also returned. Result sets can be presented to a user and browsed as described below.

### 4.2.2 Browsing a set of recordings

Browsing a set of recording URIs resulting from a query is similar to browsing a complete collection, and the same type of user interfaces can be used for both functions.

The use of tables is shown in Figure 4.3, which displays part of the result set from the query in Figure 4.2 — one table contains counts of events found; the other event times. The display window allows a user to sort columns and scroll through results, and to save the result set to a file for later analysis. Particular recordings or events can be viewed in a visualisation module by double clicking on their row.
Using scrolling tables to browse large data sets, whether from a query or recording collection, can be difficult, both because of the sheer number of rows to scroll and because of delays due to data transfer and computer memory limits. These difficulties can be minimised by providing summary data, such as counts of resource frequency, and by exploiting hierarchical relationships between resources in an expanding/collapsing tree control.

The top level of a tree control might show the Database or Collection a recording belongs to; the next level recordings in the database; and then a level with the recording’s associated signals, events and annotations. Browsing might also be enhanced by having a secondary search field to filter the display of rows. Figure 4.4 shows a possible browser interface with these characteristics.

### 4.2.3 Signal visualisation

Plotting and viewing signal data against time, whether on a strip-chart, oscilloscope, or by computer visualisation, is a traditional way of examining signals. Any set of
signal browsing tools ought to have this ability. At a minimum, a visualisation tool ought to be able to:

- display several signals from a recording using a common time axis.
- show annotations and events using the shared time axis.
- zoom and scroll along the time axis.
- select temporal sub-segments so they can be annotated.
- select temporal sub-segments for data export.

Displayed information, besides actual graphical data, ought to include:

- the recording’s description and start time.
- the time-axis units and coordinate values.
- for each signal, its label, units, and coordinate values.
- data values of signals at a given time point.
- details of annotations.

Figure 4.5 shows a recording being viewed in such a tool, developed as part of this project.
Figure 4.5 – A recording visualisation tool.
The plotting region is divided into horizontal bands where individual data and event signals are displayed. The viewer provides vertical temporal markers that can be moved to give signal values at the corresponding time position. Events are displayed using symbols, specific to the class of event signal, with descriptive text shown when selected. Signal axes are auto-scaled to accommodate each signal’s range and the time axis can be magnified and scrolled over the current segment’s temporal range. Temporal regions are able to be selected by clicking and dragging using a mouse; selected regions can then be annotated or have their signal data exported to a file.

A separate screen, shown in Figure 4.6, is used to control what portion of the recording is viewed. It provides a list of annotations and events associated with the recording, allowing a user to filter and find events of interest and then, by double-clicking, positioning the viewer to display signals in the event’s temporal neighbourhood. The bottom part of this window controls the display of signals, both their position on the plot and whether or not they are shown.

![Figure 4.6 – Controlling what is viewed.](http://devi.bio.signal.org/remote/physiobank/mtb3/102)
4.2.4 Annotation and curation

Additional information about signals and recordings, and their temporal regions, is given by annotations. These can take the form of comments, keywords, or semantic tags, defined in ontologies. The ability to search annotations (such as described in Section 4.2.1) is an important way to find items of interest; the display of annotations in their context with signals is of equal importance.

As seen in Figure 4.5, annotations of temporal regions are displayed as coloured bars spanning the annotated interval, with annotation text shown as a tool-tip. New annotations can be made by selecting a region and using the mouse’s right-click menu; existing annotations may be edited, which will create a new annotation linked to the original via provenance statements. The viewer’s control window also lists annotations and their timing information.

Curation

One common difficulty in managing a large collection of resources is in having sufficient people able to maintain and interpret the collection. For electronic resources this workload can be distributed to a wider, web-based, community by providing annotation tools that allow comments to be made, not only about resources but also about other comments. Annotations made in BioSignalML allow this, with provenance statements tracking previous versions of both resources and comments. When used with appropriate software, the BioSignalML annotation model would allow researchers to provide commentary about the contents of a biosignal repository and be a significant component in a collaborative curation process.

4.3 The Conversion Process

The majority of metadata about recordings in PhysioBank is available in the PhysioBank Index file, which describes each recording and its signals over several lines of
4.3. THE CONVERSION PROCESS

formatted text, and this is used as the primary source of metadata when generating BioSignalML. Some recordings also have metadata and/or signal data in supplementary files, and these are processed during conversion.

Each line in the PhysioBank Index consists of a number of tab-separated columns, described fully in [129] — the first column gives the recording’s name; the second either a category or signal type, along with sequence number; the remaining columns are category dependent. Types of categories are:

- **AgeSex**, **Diag**, **Info**, or **Meds**, describe features of a recording and the patient from whom the recording was collected.
- **AnnM** or **AnnR**, describe annotation sets — **AnnR** designates a reference annotation, that has been verified by experts; **AnnM** a machine generated annotation. Annotation information includes the annotator name and a summary of stored values, the timing resolution (as a frequency), counts and duration, and the time ranges spanned by values.\(^7\)
- A class of data signal, such as **ECG**, **BP** or **EEG**. Information about signals include its label, sampling rate, gain and units, a duration, and optionally, the time intervals when the signal is present.

As part of this project, I have developed a software utility to parse index lines describing a recording and generate BioSignalML statements, using terms and concepts from the BioSignalML ontology and from Semantic Web ontologies (described in Chapter 2). PhysioBank specific terms and concepts are described using a simple OWL ontology\(^8\) which is extended as new terms are discovered during the conversion process.\(^9\) URIs for drugs and medications are obtained from an RDF version of DrugBank\(^10\) [130]. The software utility also creates RDF from metadata in supplementary files, making use of these ontologies.

---

\(^7\)Note however that an annotation in PhysioBank corresponds to a BioSignalML Event signal, not an Annotation.

\(^8\)Available online at [http://www.biosignalml.org/ontologies/examples/physiobank](http://www.biosignalml.org/ontologies/examples/physiobank).

\(^9\)Any full conversion of PhysioBank to BioSignalML would require a more comprehensive ontology and/or the use of existing medical ontologies, such as those based on UMLS and SNOMED.

\(^10\)[http://www4.wiwiss.fu-berlin.de/drugbank/](http://www4.wiwiss.fu-berlin.de/drugbank/)
The utility creates **Recording**, **Signal**, **Event** and **Annotation** BioSignalML resources as required. Because the Index also describes the patient or subject of a recording, **Patient** resources (from our local PhysioBank ontology) are created and used when generating statements from **AgeSex**, **Meds**, and **Diag** index entries.

During conversion, each annotation in a WFDB annotation signal is mapped to a separate **Event** occurring at the annotation time; events corresponding to beat rhythm changes and signal quality changes are also given a duration. This allows subsequent queries to find particular classes of events, along with their temporal positions.

Errors detected by the conversion utility generate annotations about the recording in error, containing an error message and having a tag of `bsml:ErrorTAG`. This allows queries to be made against the resulting set of BioSignalML recordings to find and resolve those with errors — by a curator, if due to an error in the original PhysioBank recording; or by adapting the conversion utility, if due to an unexpected type of input.

Using the software utility to generate RDF is part of a general process to analyse and convert a given PhysioBank database to BioSignalML. As shown in Figure 4.7, a number of steps are undertaken to analyse variations in metadata structure and incorporate them into both conversion software and the local PhysioBank ontology, before all recordings in a database can be translated:

1. The process begins with reviewing the database’s online description to see whether it has supplementary data that is not available in the master index nor via the WFDB library.
2. If so, specific code is written so that this additional data will be added to the resulting BioSignalML recording.
3. Master index entries for selected recordings in the database are then examined (using command line search tools and editors) to determine common terms, their encoding, and the structure of patient and recording information.
4. If needed, new terms and concepts are added to the local ontology (for PhysioBank specific concepts) or resolved in external ontologies (e.g. prescribed medications in DrugBank).

5. If needed, database specific code to extract metadata and map it to BioSignalML statements is added to the conversion utility.

6. The conversion utility is then run against a single recording in the database and its output checked for error messages.

7. Software modifications are made as required to resolve errors. Any new concepts found are added to the local ontology.

8. Inconsistencies in generated RDF and HDF5 data are then checked for and resolved. Testing with selected recordings is repeated until all errors have been eliminated.

9. The utility is run against all recordings in the database, re-running as necessary if further modifications are required to resolve errors.

Translated metadata and all signals and events from the original recording are then stored in an HDF5 file, structured as described in Appendix C, thus allowing concurrent access by different processes and threads. The resulting file can either be used directly by processing and analysis tools or imported into a BioSignalML repository.

As described below, this process has, as part of this project, been applied to three PhysioBank databases, with databases chosen to highlight different aspects of the conversion process.

### 4.3.1 MIT-BIH Arrhythmia Database

The MIT-BIH Arrhythmia Database [131], mitdb,\(^{11}\) consists of 48 half-hour excerpts of two-channel ambulatory ECG recordings, chosen to include both common and less common but clinically significant arrhythmias. Each beat has been annotated by two or more cardiologists working independently and then resolving discrepancies,

\(^{11}\text{http://physionet.org/physiobank/database/mitdb/}\)
New PhysioBank database
Determine structure of recording and patient information
New metadata terms?
No
Yes
Non WFDB files?
Add new terms to local ontology(s)
Add metadata mapping code to conversion software
Run conversion software against database
Repository Import
Resolve cause(s) of errors. May require new metadata terms and/or software changes.
Convert single recording from database
Error messages?
No
Yes
Run conversion software against database
No
Yes
Yes
Examine generated RDF attributes
Inconsistencies?
No
Yes
Yes
Convert all recordings in database
Error messages?
No
Yes
Yes
Write and add database specific code to conversion software
Non WFDB files?
Determine structure of patient information
New metadata terms?
No
Yes
Add new terms to local ontology(s)
Add metadata mapping code to conversion software
Run conversion software against database
New PhysioBank database
Figure 4.7 – General process to convert a PhysioBank database to BioSignalML.
4.3. THE CONVERSION PROCESS

so as to produce a reference set of standard recordings. The database has been used
to evaluate arrhythmia detectors and for basic research into cardiac dynamics.

All machine readable information about \texttt{mitdb} recordings is contained in WFDB
formatted files and the PhysioBank Index; there are no supplementary files.

Index entries for a typical recording are shown in Listing 4.1. Lines 1–4 describe
the recording and patient; lines 5–6 describe two signal channels (ECG leads V5 and
V2), both sampled at 360Hz with 200 digital units per millivolt; the remaining lines
describe an annotation set produced by the reference ECG beat annotator, \texttt{atr} —
line 7 states there is a total of 2192 beats in the recording, with timing accurate
to \((1/360)^{th}\) second; lines 8–13 describe individual classes of beat annotations: three
annotations indicate the start of pacing (\texttt{atr}/(P); there are 2028 paced beats (\texttt{atr}/); 56
beats are a fusion of paced and normal beats (\texttt{atr}/f); two annotations indicate the start
of normal rhythm (\texttt{atr}/(N) with 99 normal beats (\texttt{atr}/N); and there are four premature
ventricular contractions (complexes), or PVCs (\texttt{atr}/V).

\begin{verbatim}
1 mitdb/102 Info1 1525 167 x1
2 mitdb/102 Meds1 Digoxin
3 mitdb/102 Info2 The rhythm is paced with a demand pacemaker. The PVCs are
   multiform.
4 mitdb/102 AgeSex 84 F
5 mitdb/102 ECG1 V5 360 200 adu/mV 1806
6 mitdb/102 ECG2 V2 360 200 adu/mV 1806
7 mitdb/102 AnnR1 atr 360 2192 1805 0-1805
8 mitdb/102 AnnR1 atr/(P 360 3 1643
9 mitdb/102 AnnR1 atr// 360 2028 1805 0-1805
10 mitdb/102 AnnR1 atr/f 360 56 1725 76-1802
11 mitdb/102 AnnR1 atr/(N 360 2 82
12 mitdb/102 AnnR1 atr/N 360 99 1614 81-1695
13 mitdb/102 AnnR1 atr/V 360 4 1472 95-1567
\end{verbatim}

\textbf{Listing 4.1} – PhysioBank index entries for mitdb/102.

Generating BiosignalML

Translating PhysioBank Index entries for \texttt{mitdb} recordings to BioSignalML is
straightforward. Following an undocumented \texttt{Info1} line, most recordings are de-
Figure 4.8 – The <mitdb/102> recording with patient metadata.

scribed by one or more lines of text contained in subsequent Info lines. An Annotation is generated for the Info1 line; multiple lines of descriptive text generate another (single) annotation.

Figure 4.8 shows RDF statements describing the <mitdb/102> recording, as generated by the conversion program. These include statements asserting that the Recording is 30 minutes and 6.6 seconds long; its source is the mitdb/102 record from Physiobank’s mitdb database; and has as subject a resource with a URI of <mitdb/102_patient> with the class Patient, being an 84 year old female who was being prescribed medication having a DrugBank number of DB00390. RDF describing other resources associated with this recording is shown in Figure 4.9, including two different classes of Signal, an Event, and an Annotation.

---

12http://www.drugbank.ca/drugs/DB00390
4.3. THE CONVERSION PROCESS

(a) The V5 ECG signal in <mitdb/102>.

(b) The atr annotation signal in <mitdb/102>.

(c) A beat event in <mitdb/102>.

(d) An annotation about <mitdb/102>.

Figure 4.9 – Components of the <mitdb/102> recording.
Finding events

Recordings usually contain a few atypical types of beats, around which a researcher may wish to view signal waveforms. With BioSignalML, each annotated beat results in an Event, as shown in Figure 4.9c; these beat events can be queried to find ones of interest, along with their type and time of occurrence, as discussed in Section 4.2.1; and then viewed in their temporal context, as seen in Figure 4.5.

4.3.2 MGH/MF Waveform Database

The Massachusetts General Hospital/Marquette Foundation (MGH/MF) Waveform Database [132], mghdb, contains recordings of haemodynamic and electrocardiographic waveforms from 250 patients in critical care units, operating rooms, and cardiac catheterisation laboratories, representing a broad spectrum of physiologic and pathophysiologic states. Typical recordings include three ECG signals, arterial and venous pressure, respiratory, and calibration signals; some recordings also have intra-cranial, left atrial, ventricular and/or intra-aortic-balloon pressure signals.

Generating BiosignalML

All machine readable information about mghdb recordings is contained in WFDB formatted files and the PhysioBank Index; there are no supplementary files. PhysioBank Index entries for a typical mghdb recording are described in Appendix E — as can be seen, the Index contains notes about the recording, patient, and procedure, which while searchable as free text, are not designed for machine interpretation.

These notes are stored as a single annotation about the recording. They are also parsed to extract drug information (patient metadata) and to create annotations for procedural events and rhythm/pressure changes. Errors encountered during this process are logged, generating error annotations for later resolution.

13http://physionet.org/physiobank/database/mghdb/
4.3. THE CONVERSION PROCESS

Figure 4.10 is a view of part of the converted mghdb/mgh137 recording, showing annotation text, signal data, and supraventricular premature beats.

4.3.3 Gait Dynamics in Neuro-Degenerative Disease Database

The Gait Dynamics in Neuro-Degenerative Disease Database [133, 134], gaitndd, contains recordings of force from under each foot whilst walking from patients with Parkinson’s disease, Huntington’s disease, and amyotrophic lateral sclerosis, as well as from healthy controls. The recordings are part of a project to quantify gait dynamics so as to better measure the effectiveness of therapeutic interventions for neurodegenerative diseases affecting gait and mobility.

Besides WFDB formatted files (with force data) and associated PhysioBank Index entries, each recording has a separate text file containing measures derived from the force signals, of stride, swing, stance and support. Another text file contains clinical information for each subject, including age, gender, height, weight, walking speed, and a measure of disease severity or duration. Data held in these files is not available via the WFDB library, nor from the standard programming interface to PhysioBank recordings, nor from the PhysioBank Index.

Generating BiosignalML

PhysioBank Index entries for gaitndd recordings categorise the force signals as having a type of ECG, even though on-line documentation about the database states these were obtained using force-sensitive resistors and are a measure of a force. These signals were reclassified to a new type of Force. Also as part of conversion, derived measures (stride, stance, etc) were added to the recording as non-uniformly sampled signals, and patient information was extracted from its separate file and used to generate patient metadata for the recording.

http://physionet.org/physiobank/database/gaitndd/
Figure 4.10 – Part of the <mghdb/mgh137> recording.
Figure 4.11 shows typical BioSignalML metadata for a **gait** recording following conversion, and includes patient and stride data loaded from supplementary files.
(a) The `<gaitndd/als12>` recording with patient and annotation metadata.

(b) The left foot force signal in `<gaitndd/als12>`.

(c) The left stride interval derived signal in `<gaitndd/als12>`.

Figure 4.11 – BioSignalML about the `<gaitndd/als12>` recording.
4.4 Results and Discussion

All recordings in each of the above databases have been converted to BioSignalML and are available from http://demo.biosignalml.org. In particular, recording metadata can be queried and browsed using generic SPARQL and RDF tools, and signal data accessed and displayed using the visualisation tool of Section 4.2.3.

For some recordings, when reading signal data to store as HDF5, the WFDB library printed console messages of "getvec: checksum error in signal X" and/or "getvec: unexpected EOF in signal X" errors, with X being a signal number, and these resulted in error annotations against the recording. However, because the library only programmatically returns an message about the last signal when several signals have errors, only this last message is in the annotation. These errors indicate there are problems with source WFDB data for which resolution will require investigation and correction by the PhysioBank curator; in terms of BioSignalML this would result in a new version of the particular recording along with provenance statements linking it to the original.

4.4.1 Metadata inconsistencies

Conversion of the above PhysioBank databases to BioSignalML highlighted differences in metadata encoding.

As an example, the Meds category of the PhysioBank Index is documented as describing “One or more medications”. For the databases examined above, mitdb recordings use the Meds index entry to list a patient’s medications as a comma separated list of drug names; in contrast, mghdb recordings use the Meds index entry to describe how medications were administered, and instead list prescribed drugs and their dosage in a PHARMACOLOGIC SUPPORT section of general Info index entries. (The gaitndd database does not have Meds entries.)
Scanning PhysioBank Index entries for other databases finds further differences in the use of Meds index entries: ptbdb recordings categorise medications into pre admission and after discharge groups, and have a space separated list of drug names; some qtdb recordings have a comma separated list of drug names in this field, whereas for others the field is used to describes recording equipment used.

Differences such as these create difficulties for automated web-crawlers trying to make sense of what they encounter. These differences can be accommodated with database specific code in a one-off conversion process; once converted to a standard format and vocabulary, information is then available to RDF-aware web agents and able to be incorporated into search indexes, data links, and reasoners.

Units of measurement

A number of typographical errors in the units of measurement field were found in the PhysioBank Index when it was first scanned to discover coding. These were notified to the PhysioBank curator for rectification and a new Index file obtained for use in the conversion process.

The Sleep-EDF Database [135], sleep-edf\(^\text{15}\) database in PhysioBank describes some of its signals as being measured in units of \(\mu V-mrs\), meaning that values have been obtained by first rectifying (root square) the sampled signal (which is in \(\mu V\)) and then taking the mean of two successive, one second, samples.

One way to describe the units of these signals is to state the signals have units of \(\mu V\) and annotate each signal with a description of how it was derived; another way would be to describe the characteristics of the low-pass filter corresponding to the \(mrs\) transform and associate this filter with each signal; a third way would be to define a new unit, \(V-mrs\) as another type of electric potential, and use it.

\(^{15}\text{http://physionet.org/physiobank/database/sleep-edf/}\)
General metadata

Archived sets of reference biosignals are invaluable when comparing different models and processing algorithms. It is important that these datasets contain as much metadata as possible, even information that may appear inconsequential at recording time.

PhysioBank includes several recordings of arterial pressure which could be used with anatomically correct blood flow models. However, sample recordings selected from the repository do not specify which artery was used to obtain pressure signals, nor give the position of the cannula needle along the artery. This makes these particular signals of little value for validating blood flow models.

4.4.2 Other PhysioBank databases

Recordings in the sleep-edf database are stored in EDF format (instead of WFDB). Although each recording is described in the PhysioBank Index and accessible via the WFDB library, some metadata can only be obtained by directly reading the EDF file. The conversion process is able to read this additional metadata and incorporate it in the resulting BioSignalML.

The Multiparameter Intelligent Monitoring in Intensive Care Databases [136], mimic2\(^{16}\) contain physiological and vital signs signals for tens of thousands of intensive care patients along with comprehensive clinical data obtained from hospital medical information systems. Converting the clinical database (currently held in SQL) into an RDF format would provide a comprehensive linked data resource for the Semantic Web. In order to be effective, such a resource would need to use terms from RDF or OWL versions of medical ontologies, such as UMLS and SNOMED CT, and require expert input from specialists in this field.

\(^{16}\)http://physionet.org/mimic2/
Having such a dataset available as RDF then provides opportunities to use automated reasoning over it, to both check consistency and for knowledge discovery through inference.

### 4.4.3 PhysioBank Web presence

PhysioBank currently provides HTML pages for display in web browsers. These pages could be enhanced by using RDFa [137] or a similar technology to embed RDF markup within HTML content. This, coupled with providing RDF representations of recordings (described using BioSignalML) via content negotiation, would allow PhysioBank to become a Linked Data resource. Additionally, a stream endpoint would allow applications built using the BioSignalML library to directly use PhysioBank recordings.

### 4.5 Summary

This chapter has described a general method for converting PhysioBank recordings into BioSignalML and demonstrated its feasibility by converting three recording databases, each different in the amount and type of data and metadata. The conversion process does not discard information — instead existing data and metadata is retained, made more accessible, and may now be extended. When in BioSignalML:

- PhysioBank resources may be made available to the Semantic Web.
- Standard vocabularies and ontologies provide metadata consistency.
- Discrepancies in free-text metadata may be identified and resolved, with historical versions tracked via provenance statements.
- Signal data values are efficiently and reliably available to multithreaded/multicore/multitasking applications.
- All supplementary information associated with a recording is available as part of the recording resource.
• Individual waveform events can be found using standard search tools.

Challenges are provided not so much by the metadata directly describing a recording and its signals, but by associated clinical and patient data, as the semantic description of this information requires the use of specialised ontologies and domain experts. BioSignalML provides a framework in which to do this work.

The focus of this chapter has been on improving the quality of metadata associated with biosignal recordings; the next chapter describes the use of BioSignalML to improve interfaces between signal data and applications.
Chapter 5

Using BioSignalML for Data

This chapter describes how BioSignalML can be used to resolve common interface issues with the use of signal data, whether as part of the modelling and simulation of physiological processes, or when used in data analysis and signal processing.

5.1 Introduction

A general model for using signal data is shown in Figure 5.1. In this model one or more signals, along with other data and parameters, form the input to a Process, which produces some kind of output. The Process could be any type of analysis, simulation, filtering, processing or other function — just something that operates on time-series data.

A digital realisation of a Process is likely to require signal values at time points\(^1\) that are different from actual data sample times, hence necessitating some form of interpolation or rate conversion. Similarly, although some functions (such as simple filters) can be implemented independently of physical units, it is more likely that specific units of measurement will be required for each of the input signals, con-

\(^1\)More generally, at points in a signal's sampling dimension.
CHAPTER 5. USING BIOSIGNALML FOR DATA

Figure 5.1 – A general model for using signal data.

Figure 5.2 – An interface to detect and reconcile data attribute mismatches.

Because BioSignalML explicitly associates sample timing and units of measurement with signals, it enables such interfaces to be realised as an intrinsic feature of
the data model. Additionally, as outlined in Chapters 3 and 4, the data and repository structures used by BioSignalML provide a framework with which to integrate biomedical data from a variety of sources in an open and extensible fashion.

In this chapter, I provide two examples which demonstrate these features — firstly, an example of providing data to a cell physiology simulation; and secondly, by integrating polysomnography and respiratory data, acquired from diverse sources, into a shared, open resource which is then used in the physical simulation of human breathing patterns.

## 5.2 Modelling and Simulation

As described in Section 1.4, the development and simulation of mathematical models is an integral part of researching physiological processes. The simulation of a model may use biosignal data for its parameters and inputs, and this is where having suitable metadata standards can assist with verifying data consistency between external sources and the input requirements of particular models and their simulations.

Some models specify time-varying input quantities as the output of an earlier (sub-)model instead of coming from an external source. As an example, models describing the effects of oscillatory Ca^{2+} signals usually have at least two components, one being a model to generate oscillations; the second, a model for some process driven by the signal. This is seen in [138] where a Ca^{2+}-induced Ca^{2+} release (CICR) signal is generated for input to a protein phosphorylation model, and in [139], where both sinusoid and CICR signals are generated and later used.

As an alternative, reference sets of experiment data could be held in a repository and used for model development and verification. A simulation’s output signals could also be stored in a repository, with the relationships between different datasets, simulations, and models described using appropriate metadata.
5.2.1 Calcium driven phosphorylation

As a demonstration, I have taken the phosphorylation model of Gall et al [139], for the activation of glycogen phosphorylase within hepatocytes, and instead of driving the model with generated CICR signals, used experimentally obtained \( \text{Ca}^{2+} \) signals taken from [140].

Perc et al. [140] describes the measurement of hepatocyte \( \text{Ca}^{2+} \) concentrations, before determining their stochastic or deterministic nature by applying non-linear time-series analysis. These measured \( \text{Ca}^{2+} \) signals are reproduced here as Figure 5.3,\(^2\) having been extracted as an image from an electronic copy of the original paper.

Digital time-series data were extracted from this figure by writing an ad-hoc Python script to scan the image horizontally and detect vertical contrast changes, with pixel coordinates scaled to obtain numerical values as per the axes. The resulting time-series are stored as BioSignalML recordings at http://demo.biosignalml.org/cellml/calcium/spiking and http://demo.biosignalml.org/cellml/calcium/bursting. No units of measurement for these signals are given in the original paper; however by comparing their value range to similar data in [141], it is clear that concentrations have been measured using nanomolar (nM) units.

The phosphorylation model from [139] was encoded into CellML\(^3\) and converted to Python using the Language Export Definition Service (CeLEDs) of the CellML API — the model’s mathematics are shown in Figure 5.4. The resulting Python code was then manually edited to accept a URI as a runtime parameter, pass this to a BioSignalML repository interface module, bsmilrepo, and use the returned Ca\(^{2+}\) signal instead of computed Ca\(^{2+}\) concentrations. As an option, state and algebraic variables resulting from the simulation can be stored as a BioSignalML recording with associated metadata, including provenance statements with details of the CellML model, input dataset, and simulation.

\(^3\)CellML sources are available at http://models.cellml.org/w/dbrooks/gall_2000.
Details of the source code changes and the \texttt{bsmlrepo} interface module are in Appendix F. Amongst other things, the interface code ensures \( \text{Ca}^{2+} \) concentrations have the correct units (experiment data values are stored in nanomolars; the model uses micromolars); scales time units (experiment timing is stored using seconds; the model uses minutes); and linearly interpolates between sample points, should the simulation code require in-between data values.

Output from the simulation is stored as a recording in a BioSignalML repository, as shown in Figure 5.5. This allows provenance statements to be made, as seen in Figure 5.5b where the PROV Ontology is used to assert the recording \texttt{wasGeneratedBy} an \texttt{Activity} that \texttt{used} a specific model and dataset; other statements might also provide details about the \texttt{Agent} that \texttt{wasAssociatedWith} the \texttt{Activity}.

(a) State and algebraic signals resulting from a simulation.

(b) Metadata showing how recording was generated.

\textbf{Figure 5.5} – Simulation results stored in BioSignalML repository.

Actual data values generated from running the simulation, for both phenylephrine (spiking) and ATP (bursting) induced \( \text{Ca}^{2+} \) signals, are shown in Figure 5.6, where the upper (red) traces show the fraction of active phosphorylase (variable
5.2. MODELLING AND SIMULATION

Pha), and the lower (green) traces, experiment derived Ca$^{2+}$ concentrations (variable $Z$), plotted against time. The Pha output could be used for model verification, should a phosphorylase activity signal have been recorded when Ca$^{2+}$ signals were measured.

(a) Active phosphorylase in response to phenylephrine.

(b) Active phosphorylase in response to ATP.

Figure 5.6 – Simulated active phosphorylase using experiment derived Ca$^{2+}$ signals.
Storing simulation results in a repository makes them available for later viewing, analysis and subsequent modelling, and allows other researchers to annotate and use regions of interest.

5.3 Data Integration

As part of their research and development programme, Fisher & Paykel Healthcare, a manufacturer of continuous positive airway pressure (CPAP) devices,\textsuperscript{4} has a growing collection of polysomnography and respiratory flow recordings. These recordings are available to researchers in a somewhat ad-hoc manner, with individuals having their own copies and exchanging disk files as they work with different datasets.

Polysomnography recordings are stored as EDF files, whilst hypnograms (sleep stages) and apnea and breath events extracted from a recording are stored in separate files, each having their own proprietary format. Separately, respiratory flow data, collected by CPAP devices during patient trials, are stored as files in another proprietary format. Annotation may be performed using a specialised application; however, the resulting comments are not available in an open format. Using such a file-based system to store recordings and derived information means:

- there is no overview of the collection as a whole, apart from as a list of files.
- researchers cannot query the collection to find recordings of interest.
- relationships and links between different recordings cannot be specified.
- specialised interfaces have to be developed to allow other applications to access data.

A subset of these recordings and their derived information have been brought together into a BioSignalML repository to provide researchers with a shared, metadata-aware resource in which data may be enhanced through annotation and linking to larger contexts.

\textsuperscript{4}A CPAP device supplies air under pressure to help keep a patient’s airways open during sleep, and is used in the treatment of obstructive sleep apnea.
5.3.1 Bringing polysomnography data together

Polysomnograms are analysed and processed at Fisher & Paykel Healthcare using a software package called Minerva. Minerva is a large application which is part of a long-term and ongoing development effort. The application is written specifically for Microsoft Windows and access to its source code is restricted. Although Minerva supports common signal file formats such as EDF, it uses its own proprietary formats for derived information. Additionally, the headers of EDF files created by Minerva were, until recently, not fully compliant with the EDF standard and so not recognised by other applications. The closed nature of Minerva means there are three possible alternatives when adding a new processing algorithm into this setting: 1) enhance Minerva by extending its code (the traditional approach); 2) develop interfaces for other applications so they can process Minerva files; or 3) convert Minerva datasets to an open format.

Processing a polysomnogram in Minerva results in associated hypnogram (HYP) and breath event (EVT) files, each stored in separate proprietary formats. The set of three files making up an analysis (the polysomnogram, in Minerva’s version of EDF, and the HYP and EVT files) have been loaded into a BioSignalML repository using a purpose-built data import utility. The utility repairs bad EDF headers and annotates the resulting recording to indicate this. Hypnogram and breath events are converted to BioSignalML events and added to the recording.

The result of importing an analysed polysomnogram is seen in Figure 5.7, which shows selected signal traces, with coloured bars denoting events and sleep stages, and event details and frequency counts (seen in the bottom left corner). All polysomnographic data are now in an open format and readily accessible by applications; event and other metadata is able to be queried; and further annotation and metadata about recordings can be readily added.

5Details of relevant file formats were obtained by studying Minerva’s source code.
Figure 5.7 – A polysomnography recording with event annotation.
5.3. DATA INTEGRATION

Beside hypnograms, the repository has been populated with respiratory flow recordings (seen in the top left of Figure 5.7) and their application is discussed below.

5.3.2 Testing CPAP devices

In order to objectively test and evaluate CPAP devices, Dr Mark Titchener has developed a piston pump that emulates realtime human breathing. This allows CPAP devices to be challenged with a range of simulated physical breathing patterns corresponding to normal and abnormal respiratory activity. The BioSignalML repository provides a shared location in which to build a reference collection of breath patterns for use in this work, along with recordings and annotation resulting from device evaluation.

CPAP devices have an inbuilt data logging capability which collects repository flow data in a proprietary format, keyed to the device’s serial number. These flow recordings are imported into the repository using a utility that transfers data using the Block Stream Format of Appendix B. Once in the repository, respiratory flow signals are processed by the PERTECS toolkit to identify breath types and features, including breath number, inspiratory and expiratory volumes, maximum inspiratory flow rate, and timing of breath components. The resulting data stream is stored in the repository as a new recording, with provenance linking it to the flow recording used. Further processing detects flow limitation and breath-to-breath variation, classifies breaths into characteristic types, and infers patient state information, such as awake/sleep state. This information is also stored in the repository with appropriate provenance, for later query and retrieval.

Figure 5.8 depicts the evaluation of a CPAP device using a preselected breathing pattern. The pattern is retrieved from the repository by an interface utility (described below) and passed as a data stream to a series of interlinked PERTECS modules which are configured as a realtime pump controller.

---

6Programmable Environment for Real-Time Emulation of Continuous Systems, see http://tcode.auckland.ac.nz/~mark/Signal%20Processing%3A%20PERTECS.html
The piston pump emulates human breathing, producing an airflow that matches the respiratory flow of the source breaths. This airflow is coupled to the device under evaluation, with data logging (built into the device) recording the emulated respiratory flow. The data log may then be subsequently retrieved, analysed, and compared with the original source data.

The set of BioSignalML recordings associated with a CPAP device and its evaluation are linked through provenance and other metadata, including the device’s serial number, allowing these recordings to become part of the audit record documenting a device’s testing and evaluation.

5.3.3 BioSignalML stream interface

A wide range of signal processing applications are currently available but cannot directly work with BioSignalML because they do not support metadata retrieval from a SPARQL endpoint. Their focus is on processing signal data as a sequence of numer-
ical values with associated timing. This data is either read from a file, or for realtime signal processing, obtained from some input channel as a data stream.\footnote{In computing, the transmission of a time-varying data is often referred to as a stream, whether it be the sending of telemetry data over a network or the exchange of a sequence of characters between applications.}

Instead of modifying existing applications, I have developed a utility that provides a general interface between time-series data in a BioSignalML repository and telemetry data streams, or by redirecting input and output, with simple numerical arrays. Although metadata attributes required for signal data are external to an application, they may be considered to be part of the configuration (scripts, parameters and settings) defining some particular signal processing instance.

The interface is specified using a simple connection definition language, as illustrated in Figure 5.9, which allows the utility to validate and convert signal data in both temporal (sample rate, time synchronisation) and value (units of measurement, numeric type) dimensions. Data values are exchanged as a simple, text-based, stream of telemetry frames at a constant rate, with each frame carrying data for a fixed number of signal channels. Unix named pipes\footnote{http://www.tldp.org/LDP/lpg/node15.html} [143] are used to transfer telemetry streams.

A connection is defined by specifying the signal channels that make up a stream and the named pipe on which the stream is sent to, or received from. An output stream requires URIs of the signals that make up its channels and optionally may have the output frame rate and the units of each channel specified, and also be restricted to a temporal segment of recording. An input stream requires URIs to be given for the resulting recording and signals, along with a (common) sample rate and units for signals; optionally, labels, descriptions, and arbitrary metadata may be given. For a new recording provenance information is automatically generated.

Further information about the interface utility is contained in Appendix G, which gives an example of using it with PERTECS to extract breath features.
5.4 Discussion

While the above has demonstrated the use of BioSignalML data in two different settings, both cases have their limitations and provide scope for future work.

5.4.1 Modelling and simulation

Currently the use of BioSignalML with a CellML simulation requires customised program code, in particular for:

- obtaining attributes describing model variables, such as units and labels.
- specifying the source of input data.
- using external data during a simulation, rather than model generated data.
• specifying the destination of simulation results.

Simulation code generated by the CellML API (CeLEDS) is intended to be used as part of a larger application rather than executed as a standalone program — access to a model's variables is provided by generic arrays (called for instance \textit{algebraic}), with no direct lookup mechanism to discover the correspondence between model and simulation variables, and units of measurement are not readily available. This information is available indirectly — for example in the calcium model used the array element \texttt{legend\_algebraic[0]} contains the string "Z in component Z (micromolar)" , from which it can be deduced that the model variable \texttt{Z} is represented as \texttt{algebraic[0]} with units of \textit{micromolar}. Having CeLEDS produce code with direct relationships would simplify linking generated code with a simulation's description.

CellML has its own, extensible, system of specifying units of measurement, whereas BioSignalML's preference is to use URIs from the UOME List and provides a mapping from common unit abbreviations (as used by CellML) to such URIs. A more robust interface would be for the CellML API to use some ontology for a variable's units; ideally this would be UOME. This could be done without changing the way units are defined in the CellML language by having the API build and map unit expressions.

The SED-ML language is intended to provide a standard format to describe how simulations are configured. The current version of the language however does not have a way to specify the use of external data nor what units data should be in, and restricts specification of outputs to how data should be presented to a user (that is, as a chart or table). Until SED-ML is enhanced to allow the specification of \textit{dataSources} and \textit{dataSinks}, it can not be used to link a simulation with external data repositories such as BioSignalML.

An alternative (to directly wrapping CeLEDS generated code) is to integrate BioSignalML into a simulation environment such as OpenCor.\footnote{This is under discussion amongst the editorial board (https://docs.google.com/document/d/1rrs0YuKFr4fgL1b7eGw5naLhRPW6NdXwAajY0ZNNY).} This is scheduled
and is largely a technical exercise, requiring the development of C++ interfaces to BioSignalML. However to ensure portability, the specification of inputs and outputs should be independent to both a model and its simulation environment, and the use of a language such as an enhanced version of SED-ML is required to enable this.

5.4.2 Data integration

The effectiveness of the BioSignalML approach is demonstrated by its application to managing the collection of polysomnography and related recordings used in the development and evaluation of CPAP devices. A disparate collection of these recordings is now kept together as an integrated resource in a BioSignalML repository. Recordings are available as a shared network resource, with the collection as a whole able to be queried to discover common patterns, relationships, and specific items of interest. Recordings may be extended with annotation and other metadata; relationships with other resources defined; and data is accessible to applications in an open format.

A BioSignalML repository can provide a reference source of breathing patterns for CPAP device evaluation. By keeping output signals from an evaluation, with provenance relating outputs back to input sources, the repository may become part of the audit trail documenting CPAP device development and testing.

Ongoing work includes expanding the capabilities of tools to allow more integration with existing workflows, and bringing BioSignalML to a wider audience by porting these tools to the Microsoft Windows platform. This work does not require changes to BioSignalML itself.

5.4.3 Streaming signals and events

The current implementation of the BioSignalML/stream interface, described in Section 5.3.3, is designed for data transfer only and does not transfer metadata. The reason for this is that PERTECS, the source and sink of data, is not metadata aware.
A possible enhancement is to stream metadata on a separate channel as serialised RDF — metadata aware processing modules would then be able to inspect, use, and add to information on this channel; non-metadata aware modules could be easily configured to transparently pass the channel on. At a minimum, this channel might carry URIs identifying signals on other channels, so enabling a processing module to obtain additional information about channels via direct SPARQL queries.

Access to metadata streams, whether those generated by processing modules or from external sources, provides opportunities for automated querying and reasoning, similar to that proposed in [144] where stream reasoning is introduced as a framework to integrate data streams with the Semantic Web and reasoning systems. In the context of realtime polysomnography monitoring, such a system might generate alerts by using a set of inference rules to reason over realtime event streams, a priori information about a patient, and an ontological knowledge base.

5.4.4 Tracking software use

The management of software scripts, applications, and settings used in processing data is an often overlooked area for ensuring reproducibility of experiments, possibly more so away from formal modelling environments — modellers use well defined languages to describe data, models, and simulations; experimentalists deal with data from actual physical and biological systems, often in a less formal setting.

While BioSignalML provides a metadata framework in which to assert data provenance, it can only do so within an environment where configuration management is enforced. Unless researchers are from a software engineering background, the use of configuration management tools is unlikely — a possible future project is the development of a general signal processing environment in which the provenance of both data and processing modules is automatically generated and associated with BioSignalML signals; in the more immediate future, PERTECS configurations and scripts are being brought under automatic code management as part of ongoing development of the interface utility described above.
Chapter 6

Discussion and Conclusion

In this thesis I present BioSignalML as a new framework for handling physiological time-series data to address existing difficulties when using and exchanging biosignals.

The thesis begins with a review of file formats used for biosignals, highlighting both common features and differences, and difficulties caused by a lack of clear standards, especially in the representation and meaning of metadata. An XML-based language is often created as a way to unify and describe data in a problem domain, and a number of such languages have been proposed for biosignal data. Their use however is not widespread — XML is designed to be used with structured text, not numerical data, and so does not provide an ideal solution to a researcher’s difficulties with biosignal storage and exchange.

The domain specific nature of biosignal storage formats, along with the lack of standardised metadata, has also hampered the integration of biosignal data with physiological modelling software — one solution has been to simply ignore machine readable metadata, although this approach may cause difficulties when reproducing simulation experiments.

BioSignalML is proposed here as a way to address these difficulties. It is based on Semantic Web technologies, so as to provide a standard framework in which to
define and work with metadata, and the optional use of HDF5, a format for efficiently storing structured numeric data. In essence, everything, apart from numeric data, is stored as RDF, with numeric data held in either its original format or converted to HDF5.

So far as I know, BioSignalML is unique in providing a general framework to store and work with biosignals and their metadata that makes use of ontologies and Semantic Web standards. Other projects addressing similar issues, described in Section 1.6, are either SQL based, or don’t use Web technologies, or don’t formally specify concepts using ontologies.

BioSignalML is presented as an Abstract Model and associated Ontology, along with an illustrative software implementation. At its core are concepts of Recording, Signal, Event, Segment, and Annotation. These concepts along with related properties, and the use of existing general-purpose ontologies and specialised domain and application ontologies, provide a framework in which to describe biosignals largely independent of their underlying storage format. This framework is extensible and able to be integrated with other RDF-based models.

Resources in BioSignalML are identified by http URIs, which allow their reference by Web applications and languages, and enable standard Web technologies to be used when providing information about resources. To demonstrate this, a web-based, client/server repository for BioSignalML resources is provided, along with a software library intended to help developers incorporate BioSignalML into user applications. The repository presents a SPARQL view of biosignal metadata and this, together with the use of public ontologies, links biosignal data into the Semantic Web.

The ability to work with BioSignalML using standard web tools, along with the library of software components for use in applications, is intended to facilitate the adoption of metadata standards — users benefit from improved query, annotation, and data services without having to know the details of what is happening under-the-hood.
Because of its abstract framework, BioSignalML can describe more than the core features of a set of signals — the framework is flexible and allows extension. This is demonstrated by its use to describe different classes of recordings held in PhysioBank, including those with extensive and unstructured metadata. The resulting descriptions do not discard information, but instead existing data and metadata are retained, made more accessible, and may now be extended with annotation and cross-referencing.

The BioSignalML model couples together data and metadata. This allows verification and transformation of signal data as it is used, in both temporal and value dimensions, as demonstrated by using BioSignalML data in the simulation of a physiological model, and as input to a realtime signal processing application. Additionally, BioSignalML’s support of provenance allows recordings to be used in audit trails documenting product development and testing.

The use of a BioSignalML repository addresses challenges that result from the range of formats used to store biosignals, presenting a common view of both data and metadata that is largely independent of underlying storage formats. Resulting signal datasets from model simulation and signal processing can also be described using BioSignalML and may include automatically generated provenance information. When stored in a repository, these datasets are available for annotation, viewing, and subsequent reuse.

Whilst these different applications demonstrate the utility of the BioSignalML model and associated software, providing quality metadata is crucial to the usefulness of BioSignalML. A challenge for researchers is to obtain this metadata, from data collection onwards. Amongst other things, this requires the general availability of domain specific ontologies, the use of standard metadata formats and models, and metadata-aware tools and applications that are easy to use. BioSignalML is part of this framework.

BioSignalML in its present form has evolved significantly over the course of this project, and while many of the underlying concepts are sound, at the implementa-
tion level details are likely to continue to evolve, especially as it is applied in different areas. BioSignalML’s design though allows for extension and growth, including the ability to use updated and new ontologies. As a proposed open standard, BioSignalML’s use doesn’t preclude future developments.

6.1 Limitations and Extensions

The BioSignalML model and software implementation is capable of being applied in its present form. However various limitations could prevent its wider use:

- There is no definition of what exactly is a conforming BioSignalML description. What is the minimal set of RDF statements, using terms from our model, which can be considered to represent BioSignalML? Further work is needed to define this, including the possible development of a BioSignalML validation service.

- Software development has focused on demonstrating the feasibility of our model, and for this, the use of Python has been more than sufficient. Providing a C++ library is now a priority, to ensure wider adoption of BioSignalML and to allow its direct use in simulation environments such as OpenCor.

- The web-browser interface was developed to demonstrate a repository’s use and only provides minimal functionality. This could be extended to include an advanced search screen, a better interface for browsing linked resources, signal visualisation tools, recording upload/download, and user authorisation.

- BioSignalML’s annotation model is simpler than the Open Annotation Data Model and Ontology,\(^1\) Although the BioSignalML model has proved sufficient for our purposes, nothing precludes the use of Open Annotation outside of our software interfaces, or as a future enhancement, directly within BioSignalML — it would be a straightforward exercise to convert existing BioSignalML an-

\(^1\)http://www.openannotation.org/spec/core/, released in February 2013 by W3C.
6.2. FUTURE WORK

notations to the Open Annotation model, and would allow better provenance to be kept about annotations.

• The W3C Provenance Model is used to track recording resources within a repository. Unless users directly issue SPARQL queries, only limited provenance information is available, and this interface could be improved by adding a user-friendly interface. Provenance of other resource classes, and/or more provenance details, could also be recorded.

• The BioSignalML model and HDF5 file structure consider a signal to be a sequence of measurements of some varying quantity, ordered by a sampling dimension, and as such could be used to describe and store complex multi-dimensional signals with spatial variation. Signals such as these would be better described in FieldML [145], a meta-language for describing fields over domains. Future work might be to develop mappings between FieldML and BioSignalML.

6.2 Future Work

Besides the extensions noted above, there are several opportunities for further work. These include the wider deployment of BioSignalML repositories, releasing a Semantic Web aware version of PhysioBank, and using BioSignalML with modelling and data annotation services.

6.2.1 PhysioBank

The work described in Chapter 4 could be extended to provide a BioSignalML version of the full PhysioBank archive as a Semantic Web resource. Amongst other things this would entail the development and/or selection of suitable ontologies to describe all current metadata and able to be extended as PhysioBank grows; providing an RDF formalisation of database and archive collections; the provision of RDF
versions of existing web pages; and the development of conversion and validation tools, in particular for the complexities of the MIMIC-II database.

Providing semantic descriptions of clinical and patient data associated with a recording will require the use of specialised ontologies and domain experts and is likely to prove challenging, although the result can only enhance the value of PhysioBank as a physiological reference. BioSignalML provides a framework in which to do this work.

6.2.2 Repository deployment

To date an instance of the BioSignalML repository is being used in a specific research project with a limited number of users. Deployment in a wider setting, say for use as a general resource in a research institution, would require a number of enhancements including improved user management and access control, transaction logging, and collection management.

Such a deployment would be valuable, as user feedback would not only lead to refinement of the repository and software tools, but would also help determine extensions and revisions of the BioSignalML model. Amongst repository enhancements might be the tracking of provenance at a more detailed level with explicit versioning, possibly in conjunction with the Physiome Model Repository, PMR2 [146]. The deployment of BioSignalML repository may also lead to greater use of RDF and ontologies within the research environment in which a repository is hosted, although this is likely to depend upon the availability of tools that hide the details of RDF from end-users.

To be more than an archive, the repository must be actively used by both experimentalists and modellers — not only for depositing and exchanging data, but also as the place in which metadata is linked. This will require greater integration between BioSignalML (and its repository services) with several things, including simulation languages and tools, workflow provenance, more external ontologies, and
user-friendly annotation services, all of which are significant projects in their own right.

6.2.3 Annotation frameworks

The linking of models and data through biological annotation, as exemplified by the RICORDO Project, is important, and the use of standards for ontologies and metadata is a large part of this process. For annotation to be effective, data resources must be identifiable in an unambiguous and universal way, and for biosignal data, a URI referencing a BioSignalML repository provides such an identifier.

As part of using BioSignalML in this larger context, concepts from its ontology need to be embedded into upper-level ontologies, such as the Descriptive Ontology for Linguistic and Cognitive Engineering, DOLCE [147], and the Basic Formal Ontology, BFO [148], or into a reference ontology such as the Ontology of Physics for Biology, OPB [83]. Doing so will allow automated reasoning and inference services to make fuller use of biosignal metadata descriptions.

6.3 Publications

The work contained in this thesis has been presented in the following articles and conference proceedings:

Appendices
Appendix A

The BioSignalML Ontology


This documentation is generated from Version 0.93.5 of the Ontology.

A.1 Classes

Class: bsml:Annotation

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Annotation

A general note, comment, or qualitative measure about the whole of, or some portion of, a Recording, Signal or Event.

Properties include: bsml:tag

Class: bsml:BP_Filter

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#BP_Filter

A band-pass filter.

Sub class of: bsml:Filter
Class: bsml:Device

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Device

A physical device that converts the output of a sensor into a format able to be stored.

Sub class of: bsml:Source

Class: bsml:Electrode

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Electrode

An electrical conductor in contact with non-conducting material, through which electrical activity can be measured.

Sub class of: bsml:Sensor

Class: bsml:ErrorTag

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#ErrorTag

A tag to indicate that an annotation relates to some form of error.

Sub class of: bsml:SemanticTag

Class: bsml:Event

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Event

Something that has occurred in time, possibly for some duration.

Sub class of: http://bhi.washington.edu/OPB#OPB01060

Properties include: bsml:eventType

Class: bsml:EventType

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#EventType

Something which is the class or type of an Event.

Used with: bsml:eventType

Class: bsml:Filter

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Filter
The class of filter that has been applied to a signal during data collection.

**Has sub class:** bsml:BP_Filter, bsml:HP_Filter, bsml:LP_Filter, bsml:Notch_Filter

**Properties include:** bsml:filterFrequency

**Used with:** bsml:preFilter

Class: bsml:HP_Filter


A high-pass filter.

**Sub class of:** bsml:Filter

Class: bsml:Instant


A particular point in time.

**Equivalent Class:** time:Instant

**Sub class of:** bsml:TemporalEntity

Class: bsml:Interval


A period in time.

**Equivalent Class:** time:Interval

**Sub class of:** bsml:TemporalEntity

Class: bsml:LP_Filter


A low-pass filter.

**Sub class of:** bsml:Filter

Class: bsml:Notch_Filter


A notch (blocking) filter.
Sub class of: bsml:Filter

Class: bsml:Recording
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Recording
A collection of Signals held as a named entity, all pertaining to one thing (the subject) and which have been recorded in the same session.
Used with: bsml:recording

Class: bsml:RecordingGraph
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#RecordingGraph
A RDF graph containing Recording metadata, used for managing provenance.

Class: bsml:SampleClock
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#SampleClock
An increasing sequence of sample coordinates.
Several signals may use the same clock.
Properties include: bsml:resolution
Used with: bsml:clock

Class: bsml:Segment
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Segment
A portion of a Signal or Recording, defined by restricting samples to some interval in the sampling dimension.

Class: bsml:SemanticTag
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#SemanticTag
A tag used to classify a resource.
Has sub class: bsml:ErrorTag
A.1. CLASSES

Class: bsml:Sensor

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Sensor

What actually captured a signal – an electrode, transducer, etc.

Has sub class: bsml:Electrode, bsml:Transducer

Used with: bsml:sensor

Class: bsml:Signal

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Signal

A sequence of periodic measurements of some physical quantity, ordered by some sampling dimension, normally time. A Signal is part of some Recording.

Sub class of: http://bhi.washington.edu/OPB#OPB01015

Has sub class: bsml:UniformSignal


Class: bsml:SignalType

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#SignalType

The class or type of signal (e.g. EEG, ECG).

Used with: bsml:signalType

Class: bsml:Simulation

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Simulation

A computer simulation or modelling process that created the Signal or Recording.

Sub class of: bsml:Source

Class: bsml:Source

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Source

The source (i.e. device, simulation, etc) of a Signal or Recording.

Has sub class: bsml:Device, bsml:Simulation
Class: bsml:TemporalEntity
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#TemporalEntity
Some measurement of time, either a particular point in time or some interval.
Equivalent Class:  time:TemporalEntity
Has sub class:  bsml:Instant, bsml:Interval
Used with:  bsml:time

Class: bsml:Transducer
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#Transducer
A device that converts a measurable quantity into an electrical signal (e.g. thermistor, pressure sensor, strain gauge).
See also: http://dbpedia.org/resource/Transducer
Sub class of:  bsml:Sensor

Class: bsml:UniformSignal
A signal that has been sampled at a constant rate.
Sub class of:  bsml:Signal
Properties include:  bsml:period, bsml:rate

Class: bsml:UnitOfMeasure
URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#UnitOfMeasure
The class used to represent the types of measurement units used by signals.
Measurement units would normally be entities in a specialised units of measure ontology.
See also: http://www.sbpax.org/uome/list.html
Equivalent Class:  http://www.sbpax.org/uome/core.owl#UnitOfMeasurement
Used with:  bsml:units
A.2 Properties

Property: bsml:clock

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#clock

The sampling coordinates associated with a signal's data values.

Domain: bsml:Signal
Range: bsml:SampleClock

Property: bsml:dataBits

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#dataBits

The binary-bit resolution of the analogue-to-digital convertor or sampling device used to digitise the signal.

Domain: bsml:Signal
Range: xsd:integer

Property: bsml:dataset

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#dataset

The location of actual data, in a format suitable for computer processing.

Range: http://purl.org/dc/dcmitype/Dataset

Property: bsml:eventType

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#eventType

The class or type of an Event.

Domain: bsml:Event
Range: bsml:EventType
Property: bsml:filterFrequency

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#filterFrequency

The cutoff frequency, in Hertz, of a filter.

Domain: bsml:Filter
Range: xsd:double
owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:index

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#index

The 0-origin position of a signal in a physical recording.

Domain: bsml:Signal
Range: xsd:integer
owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:maxFrequency

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#maxFrequency

The maximum frequency, in Hertz, contained in the signal.

Domain: bsml:Signal
Range: xsd:double
owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:maxValue

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#maxValue

The maximum value of the signal.

Domain: bsml:Signal
Range: xsd:double
owl:DatatypeProperty
owl:FunctionalProperty
Property: bsml:minFrequency

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#minFrequency

The minimum frequency, in Hertz, contained in the signal.

Domain: bsml:Signal
Range: xsd:double
owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:minValue

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#minValue

The minimum value of the signal.

Domain: bsml:Signal
Range: xsd:double
owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:offset

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#offset

The temporal offset, from the beginning of a recording, to a signal’s first sample.

Sub property of: dct:temporal
Range: xsd:duration
owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:period

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#period

The sampling period, in seconds, of a uniformly sampled signal.

Domain: bsml:UniformSignal
Range: xsd:double
owl:DatatypeProperty
owl:FunctionalProperty
Property: bsml:preFilter

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#preFilter

Pre-filtering applied to a signal as part of collection.

Domain: bsml:Signal
Range: bsml:Filter

owl:FunctionalProperty
owl:ObjectProperty

Property: bsml:rate

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#rate

The sampling rate, in Hertz, of a uniformly sampled signal.

Domain: bsml:UniformSignal
Range: xsd:double

owl:DatatypeProperty
owl:FunctionalProperty

Property: bsml:recording

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#recording

The Recording a Signal is part of.

Domain: bsml:Signal
Range: bsml:Recording

owl:FunctionalProperty
owl:ObjectProperty

Property: bsml:resolution

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#resolution

The resolution, in seconds, of a clock’s timing.

Domain: bsml:SampleClock
Range: xsd:double

owl:DatatypeProperty
owl:FunctionalProperty
A.2. PROPERTIES

Property: bsml:sensor

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#sensor

What was used to collect or derive an electrical signal.

Domain: bsml:Signal
Range: bsml:Sensor
owl:ObjectProperty

Property: bsml:signalType

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#signalType

A signal's generic type.

Domain: bsml:Signal
Range: bsml:SignalType
owl:FunctionalProperty
owl:ObjectProperty

Property: bsml:tag

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#tag

A semantic tag given to a resource by an annotation.

Tags are effectively controlled keywords.

Domain: bsml:Annotation
owl:ObjectProperty

Property: bsml:time

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#time

An instant or interval associated with a resource.

Range: bsml:TemporalEntity
owl:FunctionalProperty
owl:ObjectProperty

Property: bsml:uncertainty

URI: http://www.biosignalml.org/ontologies/2011/04/biosignalml#uncertainty
A resource describing the measurement uncertainty associated with a Recording, Signal, or Segment.

owl:ObjectProperty

**Property: bsml:units**

URI: [http://www.biosignalml.org/ontologies/2011/04/biosignalml#units](http://www.biosignalml.org/ontologies/2011/04/biosignalml#units)

The physical units that are represented by a signal’s data values.

Specification of units allows for consistency checking and automatic conversion.

**Domain:** bsml:Signal  
**Range:** bsml:UnitOfMeasure  

owl:FunctionalProperty  
owl:ObjectProperty
Appendix B

Block Stream Format

This appendix describes a message protocol used to transport BioSignalML data blocks over a WebSockets stream.

A block stream is a sequence of blocks, where a block is a sequence of 8-bit bytes. A block is of some type, given by a single, alphabetic character; has a header, consisting of a set of (name, value) pairs in JavaScript Object Notation (JSON) format, where the allowed names and values are specific to the block type; has some (possibly empty) content; and includes an optional checksum. The structure of a general block is defined in Listing B.1, using Extended Backus-Naur Form (EBNF) and regular expression notation.

<table>
<thead>
<tr>
<th>Line</th>
<th>EBNF Representation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>&lt;block&gt;</code> ::= '#' &lt;type&gt; &lt;version&gt; &lt;header&gt; &lt;length&gt; &lt;LF&gt;</td>
<td>A block starts with a single character followed by type, version, and header</td>
</tr>
<tr>
<td>2</td>
<td><code>content</code> '##' &lt;checksum&gt;? &lt;LF&gt;</td>
<td>The content is followed by a double hash and an optional checksum</td>
</tr>
<tr>
<td>4</td>
<td><code>&lt;type&gt;</code> ::= [a-zA-Z] /* A single, case-significant letter. */</td>
<td>The type is a single character</td>
</tr>
<tr>
<td>5</td>
<td><code>&lt;version&gt;</code> ::= &lt;INTEGER&gt; 'V' /* The version of the protocol. */</td>
<td>The version is an integer followed by 'V'</td>
</tr>
<tr>
<td>7</td>
<td><code>&lt;header&gt;</code> ::= &lt;jsonlen&gt; &lt;json&gt;</td>
<td>The header is a JSON object</td>
</tr>
<tr>
<td>8</td>
<td><code>&lt;jsonlen&gt;</code> ::= &lt;INTEGER&gt; /* The number of bytes of JSON that follow. */</td>
<td>The number of bytes of JSON</td>
</tr>
<tr>
<td>9</td>
<td><code>&lt;json&gt;</code> ::= '{' ... '}' /* A well-formed JSON object. */</td>
<td>The JSON object is enclosed in curly braces</td>
</tr>
<tr>
<td>11</td>
<td><code>&lt;length&gt;</code> ::= &lt;INTEGER&gt; /* The length of the content part. */</td>
<td>The length of the content part</td>
</tr>
<tr>
<td>12</td>
<td><code>&lt;content&gt;</code> ::= [#x00-#xFF]* /* A sequence of bytes. */</td>
<td>The content is a sequence of bytes</td>
</tr>
<tr>
<td>14</td>
<td><code>&lt;checksum&gt;</code> ::= [0-9a-fA-F]{40} /* An optional, 40 character SHA1 hex */</td>
<td>The checksum is a 40-character hexadecimal SHA1 digest</td>
</tr>
<tr>
<td>15</td>
<td>/* digest of the block, including opening */</td>
<td>The opening '#' character</td>
</tr>
<tr>
<td>16</td>
<td>/* and closing '##' characters. */</td>
<td>The closing '##' characters</td>
</tr>
<tr>
<td>17</td>
<td><code>&lt;integer&gt;</code> ::= [0-9]+</td>
<td>An integer</td>
</tr>
<tr>
<td>18</td>
<td><code>&lt;LF&gt;</code> ::= #x0A</td>
<td>The line feed character</td>
</tr>
</tbody>
</table>

Listing B.1 – Structure of a message block defined using EBNF.
B.1 Block Types

A BioSignalML block stream makes use of three block types, defined in Listing B.2. Details of their header fields follow below.

Listing B.2 – Block types used by BioSignalML.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;type&gt; ::= &lt;data_request&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;data_request&gt; ::= 'd'</td>
</tr>
<tr>
<td>4</td>
<td>&lt;data&gt; ::= 'D'</td>
</tr>
<tr>
<td>5</td>
<td>&lt;error&gt; ::= 'E'</td>
</tr>
</tbody>
</table>

B.1.1 Data request

A request for time-series data for a recording or a list of signals. Data is returned in data blocks. A request block has the following header fields and no content.

uri (string or list of strings) — The URI of a recording or URI(s) of signal(s). REQUIRED.
A single URI can be that of a recording or signal; all URIs in a list must refer to signals. If the URI is that of a recording then data for all signals in the recording is returned. Several data blocks may be generated to span the requested duration; when the request is for multiple signals or for a recording, each signal’s data will be in one or more separate blocks.

start (float) — The time, in seconds from the start of the signal’s recording. The first sample point returned should not be before this time. REQUIRED when multiple signals or when no offset is given.

duration (float) — The duration, in seconds, of time-series data to return. A value of -1 means to return all sample points from the start time or offset until time-series’ end. REQUIRED when multiple signals or if no count is given.

offset (integer) – The index in the signal’s time-series of the first sample point in the result. REQUIRED when start is not specified; can only be used when requesting data from a single signal.

count (integer) — The number of sample points to return in the result. A value of -1 means to get all samples, from the start position until the end of the time-series.
B.1. BLOCK TYPES

REQUIRED when \textit{duration} is not specified; can only be used when requesting data from a single signal.

\textbf{maxsize} (integer) — The maximum number of sample values to return in a data block.

OPTIONAL.

If unspecified, the data source will determine the maximum. If \textit{maxsize} is given, the data source may elect to impose a smaller value.

\textbf{dtype} (string) — The required numeric type for data points, in the format \texttt{<f4} as defined and used by numpy’s array interface.

OPTIONAL.

\textbf{ctype} (string) — The required numeric type for samples, in the form \texttt{<f4} as defined and used by numpy’s array interface.

OPTIONAL.

The time of the first sample point in the resulting time-series will not be before \textit{start}; that of the last sample point will be before \textit{start} + \textit{duration}. If the signal’s data finishes before the requested duration a shorter time-series will be returned; if the period spanned in a signal contains discontinuous segments they will be returned as separate blocks.

\section*{B.1.2 Data block}

Contains time-series data for a segment of a signal as an array of sample values, optionally preceded by an array of sample times. Sample values are either all scalars or all 1-D arrays, each with the same bounds.

A data block’s header has the following fields:

\textbf{uri} (string) — The URI of the signal whose data is in the block.

REQUIRED.

\textbf{start} (float) — The time in seconds, from the start of the signal’s recording, of the first sample value.

REQUIRED.

\textbf{offset} (integer) — The index of the first sample value in the signal’s time-series.

REQUIRED.

\textbf{count} (integer) — The number of sample values in the data block.

REQUIRED.

\textbf{dims} (integer), default = 1 — The number of data points in a single sample value.

OPTIONAL.
**APPENDIX B. BLOCK STREAM FORMAT**

**dtype** (string) — The numeric type of a single data point in a sample value, in the form `<f4` as defined and used by numpy’s array interface. REQUIRED.

**rate** (double) — The rate, in Hertz, of sample values. REQUIRED if no `ctype` is given, otherwise MUST NOT be given.

**ctype** (string) — The numeric type of a sample time, in the form `<f4` as defined and used by numpy’s array interface. REQUIRED if no `rate` is given, otherwise MUST NOT be given.

A data block’s content consists of `count` binary numbers of type `ctype` (when `ctype` is specified), followed by `count*dims` binary numbers of type `dtype`.

**B.1.3 Error block**

An error response — the block’s content contains an error message as text; its header will contain the field `type`, with its value the type character of the request block which caused the error, along with all header fields from the original request.

**B.2 Version**

The initial version of the Block Stream Format is 1 and is placed in a block’s preamble immediately after the block type.

It is expected that later versions, while possibly introducing new features, will maintain backwards compatibility with prior versions.
Appendix C

BioSignalML HDF5 File Layout

HDF5\(^1\) provides a mature, robust, multi-platform, well supported data model and file format able to describe and efficiently store and retrieve very large quantities of data. This Appendix specifies an HDF5 file layout for storing biosignals, having as its basis the BioSignalML abstract model.

C.1 General Structure

Figure C.1 illustrates the general structure of a BioSignalML HDF5 file, showing the HDF5 \textit{groups} and \textit{datasets}, and the relationships between them. Details of this structure follow.

C.1.1 Single recording per file

A BioSignalML HDF5 file MUST only contain a single Recording.

A Recording is the container object of the BioSignal model to which other objects are related, and requiring a one-to-one relationship between Recordings and files is a natural fit. HDF5, because of its hierarchical structure, would support multiple Recordings in a file, and providing for this is a possible future extension.

\(^1\)http://www.hdfgroup.org/HDF5/
C.1.2 Version attribute

A BioSignalML HDF5 file MUST have a string valued attribute with a name of *version* on the root group ("/"), and whose value MUST start with the five characters BSML and then be followed by major and minor version numbers, separated by a period (".").

The initial version of the BioSignalML HDF5 File Layout SHALL have a major version number of ‘1’ and a minor version number of ‘0’ — i.e. the version string is BSML 1.0.

It is expected that new releases, while possibly introducing new features, will maintain backwards compatibility with prior versions.
C.1.3 URI index

A BioSignalML HDF5 file MUST have a group named /uris. This group's attributes MUST have names which are the URIs of the Recording, its Signals, and any Clocks in the file. The value of an attribute MUST be that of an HDF5 reference to the group or dataset of the Recording, Signal or Clock.

C.1.4 Recording group

A BioSignalML HDF5 file MUST have a group named /recording and this group MUST have a string attribute named uri. The value of this attribute is the URI of the Recording.

This group will have a signal sub-group, and possibly a clock sub-group, containing respective signal and timing datasets.

C.1.5 RDF metadata

A BioSignalML HDF5 file MAY have a dataset named /metadata, containing an RDF serialisation of metadata associated with the Recording, stored as a UTF-8 string. The dataset MAY have an attribute named mimetype, giving the format of the RDF serialisation, using standard IANA registered mimetypes. If the attribute is missing the format is assumed to be application/rdf+xml.

This RDF would typically be the contents of the named graph containing all metadata about the Recording, and be as stored in a repository or triplestore.

C.1.6 Signal group and datasets

A BioSignalML HDF5 file MUST have a /recording/signal group to contain signal datasets.

The /recording/signal group MAY contain one or more signal datasets. Signal datasets MUST be consecutively numbered, starting from zero. (i.e. the first signal dataset will have a name of /recording/signal/0).

Each signal dataset MUST have an attribute named uri. The value of this attribute must either be the URI of the signal as a string, or it must be an array of strings. If an
array, the signal is made up of a number of channels (sharing the same timing), each with a URI given by the corresponding array element.

Each signal dataset MUST have an attribute named *units*. The value of this attribute must either be a string giving the physical units of measurement of the signal, or when the signal consists of multiple channels, it must be an array of strings with each element giving the physical units of measurement of the corresponding signal channel.

Each signal dataset MUST have one, and only one, attribute with a name of either *rate*, *period* or *clock*.

If a signal dataset has an attribute named *rate* or *period*, its value MUST be a floating point number. The dataset MAY then have a *timeunits* attribute, giving the units of measurement of the sampling *period*, with the units of measurement of the sampling *rate* being the reciprocal. Default time units are seconds, that is the default units for *rate* is Hertz.

If a signal dataset has an attribute named *clock* then its value MUST be a reference to some HDF5 dataset in the */recording/clock* group.

Each signal dataset MAY have an attribute named *starttime* as a floating point number, giving the *timeunits* offset from the start of the Recording to the start of the signal. The default value is 0.0.

Each signal dataset MAY have attributes named *gain* and/or *offset*, having floating point values, defaulting to 1.0 and 0.0 respectively. The physical value of a signal is obtained by subtracting any offset from the value stored in a dataset before multiplying by the gain.

### C.1.7 Clock group and datasets

If any signal dataset has a *clock* attribute the BioSignalML HDF5 file MUST have a group named */recording/clock*, to contain clock datasets.

The */recording/clock* group MUST contain a dataset for each separate *clock* attribute of signal datasets. Clock datasets MUST be consecutively numbered, starting from zero. (i.e. the first clock dataset will have a name of */recording/clock/0*).

Each clock dataset MUST have an attribute named *uri*. The value of this attribute must be a string, the URI of the clock.
Each clock dataset MAY have an attribute named *units*, whose value is the physical units of measurement of the clock. The default units of measurement is seconds.

Each clock dataset MAY have an attribute named *starttime* as a floating point number, giving the *units* offset from the start of the Recording to the start of the clock’s times. The default value is 0.0.

Each clock dataset MAY have an attribute named *scale*, being a floating point number with which to multiply values stored in the dataset to obtain physical values of time points. The default value is 1.0.

### C.2 Discontinuous Signals

A signal may consist of several discontinuous segments. In this case, a signal’s dataset is replaced by a group and reference by the signal’s URI entry in the attributes of the /uris group.

#### C.2.1 Signal group and segment datasets

If the HDF5 object corresponding to a signal’s URI is an HDF5 group (instead of an HDF5 dataset) then the signal is discontinuous and the group MAY contain one or more segment datasets. Segment datasets MUST be consecutively numbered, starting from zero. (i.e. the first segment dataset of the first signal will have a name of /recording/signal/0/0).

The signal group MUST have attributes named *uri* and *units* with values having the same meaning as defined for signal datasets in Section C.1.6 above.

Each segment dataset MUST have one, and only one, attribute with a name of either *rate*, *period* or *clock*, and MAY have an attribute named *timeunits*, with values having the same meaning as defined for signal datasets in Section C.1.6 above.

Each segment dataset MUST have an attribute named *starttime* as a floating point number, giving the *timeunits* offset from the start of the Recording to the start of the segment.

Each segment dataset MAY have attributes named *gain* and/or *offset*, with values having the same meaning as defined for signal datasets in Section C.1.6 above.
C.3 Units of Measurement

Units of measurement SHOULD be specified using pre-defined unit URIs from the Units of Measurement Expressions List at http://www.sbpax.org/uome/list.html, and if not, then as URIs of resources defined using the UOME Core vocabulary.
Appendix D

Software Tools

In addition to the BioSignalML software library and repository application, various other tools and utilities have been developed as part of this project. These are summarised in Table D.1. However, unlike the library and repository which are now relatively stable, these tools are under development as BioSignalML is applied into different areas, and at best they should be considered to be of no more than alpha quality.

<table>
<thead>
<tr>
<th>Category</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General utilities</td>
<td><a href="https://github.com/dbrnz/biosignalml-utils.git">https://github.com/dbrnz/biosignalml-utils.git</a></td>
<td>Various utilities and scripts for working with BioSignalML.</td>
</tr>
<tr>
<td>Streaming interface utilities</td>
<td><a href="https://github.com/dbrnz/biosignalml-streams.git">https://github.com/dbrnz/biosignalml-streams.git</a></td>
<td>Utilities to allow telemetry streams to be exchanged with a BioSignalML repository.</td>
</tr>
</tbody>
</table>

Table D.1 – Important biosignal formats.
Appendix E

PhysioBank Index for a MGHDB Recording

Listing E.1 shows typical PhysioBank Index entries about a recording in the mghdb database. Besides information describing data and annotation signals, Index entries for mghdb recordings have a substantial block of text (in some cases over 80 lines) containing clinical observations and notes of events that occurred when a recording was collected.

These notes are structured, consisting of section headers introducing categories of text. An analysis of header lines though finds some variation in text,\(^1\) although having identified differences, they can be incorporated into conversion software. The layout of a category’s content is also not consistent — fields are not always well-defined nor are controlled vocabularies used. On-line documentation states that the notes are a result of scanning a printed copy of the original MGH/MF Patient Guide\(^2\) and this is may have also introduced errors.

\(^1\)For example PACEMAKER DATA, PACEMAKER INFO, PACEMAKER INFORMATION, and PACER INFO are all used to denote pacemaker details.

\(^2\)http://physionet.org/physiobank/database/mghdb/patient-guide.shtml
APPENDIX E. PHYSIOBANK INDEX FOR A MGHDB RECORDING

| ECG Lead I | 360 | 290 adu/mV | 5190 |
| ECG Lead AVF | 360 | 290 adu/mV | 5190 |
| ECG Lead V3 | 360 | 290 adu/mV | 5190 |
| Pressure 1 | 360 | 9.025 adu/mmHg | 5190 |
| Pressure 2 | 360 | 7 adu/mmHg | 5190 |
| Status 1 | No signal | 360 | 1000 adu/mV | 5190 |
| Pressure 3 | 360 | 2.73 adu/mmHg | 5190 |
| Status 2 | No signal | 360 | 1000 adu/mV | 5190 |

Listing E.1 – Master index entries for a MGHDB recording.
Appendix F

Code Modifications for CellML Simulation

The CellML API includes the Language Export Definition Service, CeLEDs, to translate a CellML model into programming language source code, that can then be run to provide a simulation of the model. A number of common languages are supported, including Python.

CeLEDs though does not provide any direct support for interfacing with BioSignalML, so the generated source code requires a number of manual edits before it can be run. Those required for the calcium phosphorylation model of Section 5.2.1 are shown in Listing F.1.

First common code from the bsmlrepo interface module is made available and a global variable declared, SIGNAL_REPO, referring to data held in the BioSignalML repository. Next, in lines 8–9 and 11–12, values for the first algebraic variable (which is Ca\(^{2+}\) concentration) are obtained from the repository instead of being directly computed. Similarly, lines 14-15 obtain data timing from the repository signal, instead of computing it.

Lines 17–25 show a new instance of repository data being created (on line 22, the createlegends() function is one supplied by CeLEDs and returns labels for all variables used in the simulation, including their units-of-measurement), along with a simulation created (on line 25) to hold metadata about this run.

Lines 27-33 allow simulation results to be saved in the repository, provided the user gave a URI under which to store them. This includes metadata about the simulation.
Interface Code for Python CeLEDs Simulations

Listing F.2 contains common code for interfacing between a CeLEDs derived simulation and a BioSignalML repository. The interface provides two classes:

- The Simulation class, a sub-class of AbstractObject, used to generate RDF statements describing a simulation.
• The `SignalRepository` class, which provides data and time points for a single
signal URI, as well as supplying a method for saving simulation results.

Code includes: requesting data to have the units-of-measurement required by
the model (lines 52 and 54); ensuring time units are as per the model, not the
signal (lines 56–59); interpolating to find data values for time points that fall
between sample times (lines 63–71); and when saving results, converting from
the model’s timing to BioSignalML’s default time units (lines 75–77).

Interface code currently supports obtaining input data for a single parameter;
extending it to support multiple time-varying parameters is a straight forward exer-
cise. This would require a common simulation rate to be specified that becomes part
of the data request (on line 54) — the repository will convert rates to ensure sample
timing is the same for all parameters.

```python
import math

import numpy as np

from biosignalml.client import Repository
from biosignalml.data import UniformTimeSeries
import biosignalml.units as units
import biosignalml.utils as utils

from biosignalml.rdf import PROV, XSD
from biosignalml.model.core import AbstractObject
from biosignalml.model.mapping import PropertyMap

class Simulation(AbstractObject):
    #================================
    metaclass = PROV.Activity
    attributes = [ 'used', 'agent', '_start', '_end' ]
    mapping = { 'used': PropertyMap(PROV.used, to_rdf=PropertyMap.get_uri,
        functional=False),
        'agent': PropertyMap(PROV.wasAssociatedWith),
        '_start': PropertyMap(PROV.startedAtTime, XSD.dateTime,
            utils.datetime_to_isoformat,
            utils.isoformat_to_datetime),
        '_end': PropertyMap(PROV.endedAtTime, XSD.dateTime,
            utils.datetime_to_isoformat,
            utils.isoformat_to_datetime),
    }

    def __init__(self, repo, **kwds):
        #--------------------------------
        super(Simulation, self).__init__(repo.uri.make_uri(), **kwds)
        self._start = utils.utctime()
```


def end(self):
    #-----------------
    self._end = utils.utctime()

class SignalRepository(object):
    #================================

def __init__(self, uri, legends):
    #---------------------------------
    self.repo = Repository(uri)
    signal = self.repo.get_signal(uri)
    if signal is None:
        raise KeyError("Signal not in repository")
    self.uri = signal.uri
    self.legends = legends  # (states, algebraic, voi, constants)
    units = self._label_units(self.legends[1][0])[1]
    self.data = np.array(0)
    for ts in signal.read(units=units):  # yields DataSegments
        self.data = np.append(self.data, ts.data)
    self.units_voi = self._label_units(self.legends[2])[1]
    to_model_time = units.UnitConverter(self.repo.store).mapping(
        signal.time_units, self.units_voi)
    self.period = to_model_time(signal.period)  # Convert from seconds
    self.times = self.period*np.arange(len(self.data), dtype='f8')
    self.duration = self.period*(len(self.data) - 1)

def get_data(self, t):
    #-----------------
    a = float(t)/self.period
    n1 = math.floor(a)
    n2 = math.ceil(a)
    if n2 >= len(self.times):
        n1 -= 1
        n2 -= 1
    return self.data[n1] + (a-n1)*(self.data[n2]-self.data[n1])

def save_results(self, simulation, out_uri, voi, states, algebraic):
    #-------------------------
    to_seconds = units.UnitConverter(self.repo.store).mapping(
        self.units_voi, units.get_units_uri('s'))
    rate = 1.0/to_seconds(signal.period)
    output = self.repo.new_recording(out_uri, generatedBy=simulation)
    outputdata = [(states[i], self.legends[0][i])
                  + [(algebraic[i], self.legends[1][i])
                      for i in xrange(len(algebraic))]
    for n, (d, l) in enumerate(outputdata):
        lu = self._label_units(l)
        sig = output.new_signal(None, units=lu[1], id=n, rate=rate,
                                 label=lu[0])
        sig.append(UniformTimeSeries(d, rate=rate))
    output.close()
@staticmethod
def _label_units(legend):
    #------------------------
    label = legend.split("in", 1)[0]
    u = legend.rsplit("(" , 1)
    if len(u) > 0: unit = units.get_units_uri(u[1].split(")", 1)[0])
    else: unit = None
    return (label, unit)

Listing F.2 – Python module for CeLEDs simulations.
Appendix G

Breath Processing with $P_{ERTECS}$

This appendix describes the use of the interface utility from Section 5.3.3 when processing respiratory flow signals with $P_{ERTECS}$.

The flow of data is as shown in Figure G.1, where the utility uses connection definitions to specify how data is transferred between a BioSignalML repository and a set of processing steps via named pipes (pipe names have their common /tmp/ prefix omitted for clarity).

![Figure G.1 – Breath feature extraction from a flow signal using $P_{ERTECS}$](image)

Listing G.1 contains the connection definitions. Lines 1–4 specify the flow signal as coming from signal/0 of the http://polly.local/Aus_Patients/14_Day/12170352/FLW0001 recording and sent to a pipe called /tmp/flow. Signal data values should be in units of cl/min, and the frame rate of data should be 50 Hz (this is unrelated to the processing rate for $P_{ERTECS}$ specified later). Next, in lines 6–10, the DC removed flow signal from the /tmp/flowDCrem pipe is saved into a new record-
ing, http://polly.local/Aus_Patients/14_Day/12170352/FLW0001/processed as the signal signal/flowDCRM. Similarly, in lines 12–25, a new recording (features) is created to store the several channels of breath information from the /tmp/thresholdflow pipe, and in lines 27–31 the breath start times from /tmp/Flowgen_c.InspStarts are saved as the sequences recording.

```
stream <http://polly.local/Aus_Patients/14_Day/12170352/FLW0001> to /tmp/flow
rate = 50
signals = [ <signal/0> units=cl/min ]

recording <http://polly.local/Aus_Patients/14_Day/12170352/FLW0001/processed>
from /tmp/flowDCrem
description = "DC removed using extract.pert"
rate = 50
signals = [ <signal/flowDCRM> label="Flow (DCRM)" units=cl/min ]

recording <http://polly.local/Aus_Patients/14_Day/12170352/FLW0001/features>
from /tmp/thresholdflow
description = "features from threshold.pert"
rate = 50
signals = [ <signal/FLOWx100> label="Flow x 100" units=l/min
<signal/breathcount> label="BREATHE-COUNT" units=Count
<signal/expvol> label="EXPIRATORY VOLUME" units=l
<signal/vol> label="VOLUME" units=l
<signal/com> label="COM-TIME WRT BOI" units=ms
<signal/boi> label="BOI START-WRT EOI" units=ms
<signal/period> label="RESPIRATORY PERIOD" units=s
<signal/maxflow> label="MAX INSPIRATORY FLOW" units=l/min
<signal/insvol> label="INSP-VOLUME" units=l ]

recording <http://polly.local/Aus_Patients/14_Day/12170352/FLW0001/sequences>
from /tmp/Flowgen_c.InspStarts
description = "sequences derived from annotation"
rate = 1
signals = [ <signal/starts> label="Inspiration Starts" units=Count ]
```

Listing G.1 – An interface specification between BioSignalML and data streams.

The requirement that the flow rate be in units of centilitres/minute (used in the PERTECS model) results in data values being converted as they are streamed (this will make use of the definitions in Listing 2.2).

Listing G.2 shows the commands used to process data. Several Unix processes are piped together in lines 1–6. First PERTECS takes data from the named pipe at /tmp/flow and uses its extract configuration settings to remove any DC offset and drift, at a processing rate of 1,000,000 frames/second. The resulting signal, without DC off-
sets, is output as a stream from PERTECS and piped into the `tee` Unix tool, which saves a copy in the pipe at `/tmp/flowDCrem` and passes the stream onto another PERTECS process (line 3). This uses the `threshold` configuration settings to detect the beginning and end of breaths and calculate a range of measures. The resulting stream is passed to further invocations of `tee` (lines 4 and 5) which copy the stream to both `/tmp/thresholdflow` and `/tmp/tempthresholdflow`, before using the `awk` tool to save the first channel of the stream into `/tmp/Flowgen_c.Flow`, for further processing (not shown here). Lines 8 and 9 use `awk` to produce a stream containing just the start times of each breath, passing it to the `/tmp/Flowgen_c.InspStarts` named pipe.

```bash
pertecs -c extract -rate 1000000 -ineof < /tmp/flow |
   tee /tmp/flowDCrem |
   pertecs -c threshold -rate 1000000 -ineof |
   tee /tmp/thresholdflow |
   tee /tmp/tempthresholdflow |
   awk '{ print $1, $2/1000 }' > /tmp/Flowgen_c.Flow
awk '{if (($3> 0) && ($3 != last)) {print last=$3, $1; fflush();}}' |
   /tmp/tempthresholdflow > /tmp/Flowgen_c.InspStarts
```

**Listing G.2** – Using PERTECS to extract breath measures.
Bibliography


[137] B. Adida, M. Birbeck, S. McCarron, and I. Herman, Eds., *RDFa Core 1.1 — Syntax and processing rules for embedding RDF through attributes*. W3C


