BioSignalML in OpenCOR — an update:

David Brooks, The University of Auckland, New Zealand

Background

BioSignalML

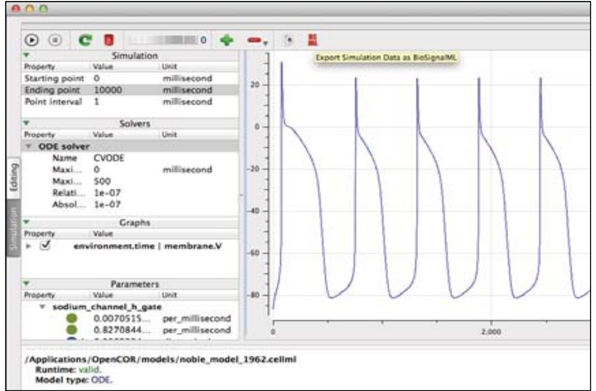
- Is an abstract framework to work with biosignal data and metadata.
- It addresses challenges with data exchange and the use of biosignals in physiological modelling software that are due to a lack of standardisation in how metadata is represented and assigned meaning.
- It uses Semantic Web standards for metadata — URIs, RDF, and OWL.
- It works with existing data formats and standards, and optionally uses HDF5 for efficient storage of large datasets.
- Enables development of web-based signal repositories providing RDF and SPARQL services.
- Allows verification and transformation as data is retrieved and used.
- Designed to facilitate data integration and to provide metadata consistency, both within and between research groups, and across a wide range of disciplines.
- Open, flexible, and extensible.

OpenCOR

- Is a cross-platform modelling environment for organising, editing, simulating and analysing CellML models.
- Plugin architecture allows modular extension.
- http://opencor.ws

Resource Description Framework (RDF)

- Is an abstract framework to work with biosignal data and metadata.
- Describes resources (things identified by a URI) using statements in the form (subject, predicate, object).
- BioSignalML is built on RDF and specifies the types of subjects and objects, and a vocabulary for predicates.
- A RDF software library is a key component of any BioSignalML implementation.



```
Simulation
Property        Value   Unit
Starting point  0       millisecond
Ending point    10000   millisecond
Point interval  1       millisecond

Solvers
Property        Value   Unit
ODE solver
  Name          CVODE
  Maxi...       0       millisecond
  Maxi...       500
  Relati...     1e-07
  Absol...      1e-07

Graphs
Property        Value
  environment.time | membrane.V

Parameters
Property        Value          Unit
  sodium_channel_h_gate
    0.0070515...   per_millisecond
    0.8270844...   per_millisecond

/Applications/OpenCOR/models/noble_model_1962.cellml
  Runtime: valid.
  Model type: ODE.
```

Hodgkin-Huxley simulation in OpenCOR

Adding BioSignalML to OpenCOR

Implementation differences:

- BioSignalML software is written in Python, a modern, readable language with good cross-platform support for RDF.
- OpenCOR is written in C++ and makes extensive use of the Qt5 framework.
- OpenCOR's simulation data classes were declared as part of each type of simulation. This hindered the ability to use common methods to supply, store, visualise, and annotate data. Simulation data has since been abstracted into a generic data plugin with a cleaner software interface.

Option 1: Embed Python in OpenCOR.

- This would allow the direct use of existing BioSignalML code inside OpenCOR.
- Would allow OpenCOR plugins to be written in Python.
- Would also allow OpenCOR users to use their own Python code for post-processing of simulation results.
- This has been demonstrated in a Unix environment (OS/X), by exporting simulation result sets into a BioSignalML repository (for viewing and annotation), and also by applying a Fourier transform to a simulation-derived signal, using Python's `scipy` package.
- However embedding Python is non-trivial under MS Windows, as standard Windows' Python is built using an older version of Visual Studio than that used for OpenCOR. (Python version 3.5 and Visual Studio 2015, both due late 2015, should resolve this incompatibility.)

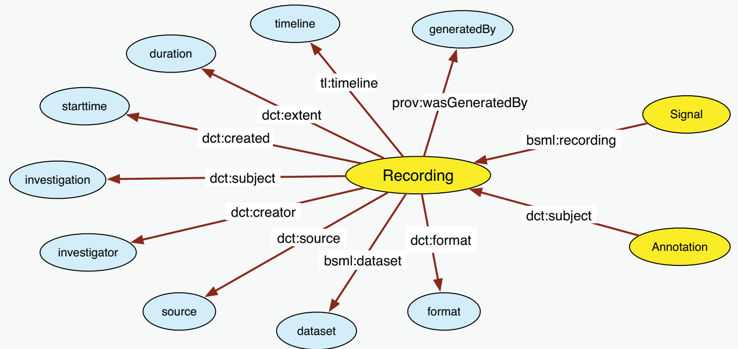
Option 2: Implement BioSignalML in C++.

- The main challenge is Microsoft Windows, as of the two open-source C/C++ libraries for RDF, one (librdf) is not supported under Windows, and the other (sord) does not support RDF/XML nor the retrieval of RDF across a network.
- Microsoft's C++ compiler (Visual Studio) still only has limited support for the latest version of C++; this will improve with Visual Studio 2015.
- Working with RDF in Python is easy because of the dynamic nature of the language; this is harder in C++ as it is a static, compiled language.

A RDF framework for C++:

- The aim iss to provide a generic framework for working with typed RDF resources in C++.
- A general system has been implemented and used for defining the BiosignalML RDF model in C++. It can also be used by other C++ applications that need to process RDF.
- Meta-programming techniques (macros, templates, and code generation) aere used to simplify mapping RDF structures into C ++ classes.
- Both sord and raptor (librdf's RDF parsing library, its one component that can be built under Windows) libraries have been used to implement C++ classes for RDF.

The BioSignalML Recording class in C++:



Specification:

```
  class Recording : public Object
  /*---------------------------*/
  {
    TYPED_OBJECT(Recording, BSML::Recording)
    PROPERTY_STRING(format, DCT::format)
    PROPERTY_NODE(dataset, BSML::dataset)
    PROPERTY_NODE_SET(source, DCT::source)
    PROPERTY_NODE(investigation, DCT::subject)
    PROPERTY_NODE(investigator, DCT::creator)
    PROPERTY_DATETIME(starttime, DCT::created)
    PROPERTY_DURATION(duration, DCT::extent)
    PROPERTY_OBJECT(timeline, TL::timeline, RelativeTimeLine)
    PROPERTY_NODE(generatedBy, PROV::wasGeneratedBy)
    PROPERTY_OBJECT_RSET(signals, BSML::recording, Signal)
    PROPERTY_OBJECT_RSET(annotations, DCT::subject, Annotation)
    } ;
  } ;
```

Processing RDF:

```
// RDF graphs would usually come from a file or network resource, or via a SPARQL query.
  rdf::Graph graph(R"(<http://demo.biosignalml.org/examples/sinewave>
    tl:timeline <http://demo.biosignalml.org/examples/sinewave/timeline> ;
    dct:created "2003-01-01T10:00:00+13:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
    dct:description "Startdate 01-JAN-2003 X X X" ;
    dct:extent “PT5.5S”^^<http://www.w3.org/2001/XMLSchema#dayTimeDuration> ;
    dct:format "application/x-bsml+edf" ;
    dct:subject "X X X X" ;
    bsml:dataset <file:///recordings/f12a3f1e-8784-11e2-9e2f-00163edeed03.edf> ;
    a bsml:Recording .)", rdf::Graph::Format::TURTLE) ;

  bsml::Recording rec("http://demo.biosignalml.org/examples/sinewave", graph) ;
  std::cout << rec.uri() << std::endl ;
  std::cout << "  Format: "        << rec.format() << std::endl
            << "  Investigation: "<< rec.investigation() << std::endl
            << "  Timeline: "      << rec.timeline()->uri() << std::endl
            << "  Duration: "      << rec.duration().to_string() << std::endl ;
```

Output:

```
<http://demo.biosignalml.org/examples/sinewave>
  Format: application/x-bsml+edf
  Investigation: X X X X
  Timeline: http://demo.biosignalml.org/examples/sinewave/timeline
  Duration: PT5.5S
```