Odil: implementation of a DICOM library in C++

Julien Lamy

ICube, Université de Strasbourg-CNRS

C++ DICOM libraries today

- We’re developing DICOM tools:
  - In C++
  - As free software
  - Using network services, client *and* server
- DICOM libraries (C++, free software):
  - Conquest, DICOM SDL: no API documentation, dead (?)
  - dicom3tools, Imebra: no network support
  - GDCM: incomplete network support (client only)
  - Winner by default: DCMTK

Why develop a new library?

- API is getting old
  - Duplication of many standard library features (OFList, OFVector, OFString, OFStream, etc.)
  - Errors not based on exception: expressed through OFCondition

    ```
    OFString value;
    OFCondition const condition =
      dataset.findAndGetOFString(DCM_PatientID, value);
    ```

  - No `const`-correctness, duplication of code due to lack of templates, callbacks are `void (*)(void*)`, etc.
- Inconsistent memory management
  - Some functions take pointers, other take references
  - Transfer of ownership not always specified (memory leak)
- Documentation not up to date: need to look in `.cpp` files, undocumented parameters

A newcomer: Odil

- Started as a set of wrappers around DCMTK: DCMTK++
- Quickly evolved to native implementation
- Time-line:
  - April 2015 (v0.1): DCMTK wrappers (data set accessors, network, association, messages)
  - June 2015 (v0.2): native data set (with conversion to and from DCMTK), JSON
  - September 2015 (v0.3): native reading and writing, native Basic Directory (i.e. DICOMDIR) creation, XML
  - January 2016 (v0.4): rename to Odil, native association, DUL, services (C-STORE, C-FIND, C-MOVE, C-GET)

Language: C++11 goodness

- C++ 11 brings cleaner-looking code, easier to maintain
  - Type inference:
    ```
    auto const & x = std::min(v.begin(), v.end());
    ```
  - Range-for: `for(auto const & item: sequence) { /* … */ }`
  - Lambda expressions:
    ```
    auto const f = [] (int x) { return x+1; }
    ```
  - Initializer list, threading, algorithms, variadic templates, etc.
- Get help from Boost for missing features (XML, networking, regular expression, etc.)
- Bleeding edge not necessary: Odil can be built on Debian 7, Ubuntu 12.04, CentOS 6

Dictionary

- List of known elements: tag, keyword, VR, VM
- Registry: namespace holding the tags
- Dictionary: mapping a tag to its keyword, VR and VM
- Automated generation using the Docbook version of the DICOM standard

```
// Get tag from registry
auto const patient_name = odil::registry::PatientName;
// Get information from dictionary
auto const entry = odil::public_dictionary.find(patient_name).second;
std::cout
  << entry.name << ": " << entry.vr << ", " << entry.vm
  << "\n";
```

Data set

- VR and types: reduce the 31 VRs to basic types (integers, reals, strings, binary)
- Every element is an array of values

```
odil::DataSet data_set;
data_set.add(odil::registry::PatientName, { "Doe^John" });
// Tag-based access
data_set.as_string(odil::registry::PatientName) = { "Doe^Jane" };
// Keyword-based access (slower)
std::cout << data_set.as_string("PatientName", 0) << "\n";
```

I/O: binary

- Stream-based (`std` or `boost::filesystem`)
- Field-tested with with various transfer syntaxes (little endian with implicit and explicit VR, big endian, JPEG lossless, etc.)
- Choice of transfer syntax, partial reading, optional group length, etc.

```
auto const header_and_data_set = odil::Reader::read_file(
  std::ifstream("foo.dcm"));
std::cout <<
  header_and_data_set.second.as_string("PatientName", 0) << "\n";
odil::DataSet header;
header.add("SourceApplicationEntityTitle", {"MYSELF"});
odil::Writer::write_file(
  header_and_data_set.second, header, std::ofstream("bar.dcm"));
```

I/O: JSON, XML

- Standard representation of data sets in JSON and XML
- Both input and output
- JSON: JSONCpp, XML: Boost.ptree

```
std::ifstream istream("foo.json");
Json::Value json;
istream >> json;

auto const data_set = odil::as_dataset(json);
std::cout << data_set.second.as_string("PatientName", 0) << "\n";

auto const xml = odil::as_xml(data_set);
std::ostream ostream("foo.xml");
boost::property_tree::xml_parser::write_xml(ostream, xml);
```

Networking: association

- Under the hood: Boost.Asio
- Handles both TCP/IP and DICOM layers

```
odil::Association association;
association.set_peer_host("184.73.255.26");
association.set_peer_port(11112);
association.update_parameters()
  .set_calling_ae_title("myself")
  .set_called_ae_title("AWSPIXELMEDPUB")
  .set_presentation_contexts({
    {
      1, odil::registry::PatientRootQueryRetrieveInformationModelFIND,
      { odil::registry::ExplicitVRLittleEndian }, true, false
    }
  });
```

Networking: services

- “Classic” services (C-STORE, C-FIND, C-GET, C-MOVE)
- Callback-based (function, functor or lambda)

```
odil::DataSet query;
query.add("PatientName", { "Doe^*" });
query.add("PatientBirthDate");

odil::FindSCU scu(association);
scu.set_affected_sop_class(
  odil::registry::PatientRootQueryRetrieveInformationModelFIND);
scu.find(
  query,
  [](odil::DataSet const & data_set)
  {
    std::cout << data_set.as_string("PatientName", 0) << ": "
      << data_set.as_string("PatientBirthDate", 0);
  });
```

Networking: services

- SCP example

```
odil::Association association;
odil::EchoSCP echo_scp(association,
  [](odil::message::CEchoRequest const &)
  {
    std::cout << "Somebody pinged us!\n";
    return odil::message::Response::Success;
  });

association.receive_association(boost::asio::ip::tcp::v4(), 11112);
auto const message = association.receive_message();
echo_scp(message);
```

Availability

- Free software, examples available with the source code: everybody welcome aboard!
- Works on Linux and OS X
- GitHub repository: `https://github.com/lamyj/odil`
- Packaged officially in Debian (testing, unstable) and Ubuntu (xenial), thanks to Debian-Med
- Unofficial packages: `https://github.com/lamyj/packages`
- Used in other projects
  - Dicomifier: DICOM converter
  - Dopamine: document-oriented PACS

Work in progress

- Python wrappers
  - Data set and I/O: done

```
header, data_set = odil.read("foo.dcm")
print data_set.as_string("PatientName")[0]
```

  - Networking, XML, JSON: in progress
- Web services (STOW, QIDO, WADO)
- OS X packaging (Homebrew), Windows