QWAMP – C++ WAMP Client Library for Qt Applications

DD2463 Project Specification
KTH Royal Institute of Technology

Elvis Stansvik
stansvik@kth.se

August 31, 2015

Abstract

Qt application developers wishing to use the WAMP protocol in their applications are currently limited in their choices. Although multiple C++ WAMP client libraries exists, none of them integrates well into Qt applications. This project aims to improve this situation by implementing a C++ WAMP client library using classes provided by Qt, and on top of the Qt event loop.

Contents

1 Introduction

This is the specification for a software development project to be carried out by Elvis Stansvik during the Fall semester of 2015 at KTH. The project deliverable is a C++ WAMP client library based on the Qt framework [10]. The project goal is to facilitate easy development of WAMP [9] application components written in C++ using the Qt framework. The project will be carried out as part of DD2463 Advanced Individual Course in Computer Science (15.0 credits) at KTH.

2 Background

This section contains a brief presentation of the two involved technologies, Qt and the WAMP protocol, followed by a rationale for the project.

2.1 About Qt

Qt is a cross-platform C++ application framework developed as an open source project with The Qt Company, a subsidiary of Digia Plc, as the main commercial backer. The framework is primarily used for developing graphical applications, but non-graphical features include SQL database access, XML and JSON parsing, thread management and network support. At the core of Qt is an event loop that dispatches asynchronous events and a signal/slot mechanism to simplify implementation of the observer pattern. The bulk of the framework consists of a large number of classes to simplify cross-platform application development.

2.2 About WAMP

The Web Application Messaging Protocol (WAMP) is a newly developed application level network protocol offering remote procedure calls (RPC) and a publish/subscribe mechanism. The protocol is used for communication between components in distributed applications. Implementations of the protocol exists for multiple programming languages. The protocol is routed, meaning a central WAMP router is required that manages registered remote procedures and subscriptions and forwards all messages (calls, return values, publications) between clients. Several WAMP routers exists, but the most feature complete is called Crossbar and is developed by Tavendo GmbH, the company behind the WAMP protocol and the main author of its specification.

2.3 Why QWAMP?

Qt application developers wishing to use WAMP in their applications are currently limited in their choices. Although three different C++ WAMP client libraries exists (Autobahn|Cpp [7], CppWAMP [4] and WAMP_POCO [6]), none of them integrates well with Qt's event loop.

Autobahn|Cpp and CppWAMP both make use of the Boost.Asio library [3] for asynchronous network communication, and are thus tied to its event loop. It's possible to use these two libraries in Qt applications by replacing the Qt event dispatcher for the application with a custom one that will also

drive the Boost.Asio event loop, as has been done by qtasio [11], but this approach has been criticized by seasoned Qt developers for being to invasive. The approach taken by qtasio is also limited to POSIX systems, as it makes use of `asio::posix::stream_descriptor`, thus spoiling the benefits of cross-platform development that Qt offers.

The WAMP_POCO library is a fork of Autobahn|Cpp that uses the POCO library [2] for communication instead of Boost.Asio, but the story here is the same: Qt application developers wishing to use the library would need to find some way of running the Qt and POCO event loops in tandem. To the best of the author's knowledge, there's currently no easy way to do this.

To summarize the situation, there seems to be a need for a C++ WAMP client library that integrates better into Qt applications. Several of the required constituents for such a library are already offered directly by the Qt framework: Classes are available for asynchronous TCP and WebSocket [5] communication as well as (de)serialization of JSON. By virtue of running on top of Qt's own event loop, the library would integrate well with existing and future Qt applications, and by making use of Qt types in its API, using the library would "feel" natural to Qt application developers.

3 Deliverable

The WAMP specification is divided into a "Basic Profile", which is mandatory and considered stable, and an "Advanced Profile", which is optional and has parts that are still under active development. The project deliverable is a C++ WAMP client library that implements the Basic Profile of the WAMP specification, with three exceptions:

- The specification allows two possible serialization formats for WAMP messages, JSON and MessagePack, and mandates that implementations support them both. Support for JSON serialization is a primary goal for this project, while MessagePack serialization is to be considered a stretch goal.
- The WAMP protocol is an officially registered subprotocol of WebSocket at IANA [1], and the WAMP specification mandates that the WAMP protocol variant (`wamp.2.json` or `wamp.2.msgpack`) be negotiated during the WebSocket opening handshake between peers using the WebSocket subprotocol negotiation mechanism. Unfortunately, there is no support for subprotocol negotiation in Qt's WebSocket implementation yet[1], which means the library will not be able to adhere to this part of the specification. Crossbar and several other WAMP routers have support for turning off this mandatory feature, and instead force one protocol variant or the other. Testing of the library against a WAMP router is thus still possible.
- Publications made by a publisher in the WAMP protocol are unacknowledged by default. This behavior can be changed with the publication option `PUBLISH.Options.acknowledge|bool`. With the option turned on, the WAMP router will acknowledge all publications made by sending

[1] A limitation which has been reported as a bug with the Qt project and will be fixed in a future Qt version.

PUBLISHED messages to the publisher. Support for this option is to be considered a stretch goal for this project.

4 Method

The project is divided into two phases, corresponding to 5 and 10 credits of the 15 credit DD2463 course, respectively. The two phases are described in the sections below. The library will be released under an open source license, either LGPL or a more permissive license[2], and published on GitHub[3].

4.1 Phase 1 (5 credits): Session Management

The milestone for the first phase is a set of building blocks required for the completion of the project. This includes data structures and routines for session establishment and closing against a WAMP router, which requires support for the HELLO, WELCOME, ABORT and GOODBYE messages. By the end of this phase, an application developer should be able to use the library to connect to and disconnect from one or more WAMP routers as well as joining and leaving WAMP realms available at a WAMP router.

Sub-goals to reach this milestone include the creation of a set of data structures for representing WAMP messages, serialization/deserialization of these messages to/from JSON according to the WAMP specification as well as implementing the session management logic within the library, including a user-facing API. The milestone functionality will be tested against running instances of the Crossbar WAMP router.

Phase 1 encompasses 5.0 credits of the DD2463 course and corresponds roughly to chapters 1 through 4 of the WAMP specification Basic Profile.

4.2 Phase 2 (10 credits): Publish/Subscribe and RPC

The milestone for the second phase is support for the publish/subscribe and RPC messaging patterns described in the Basic Profile of the WAMP specification. This involves support for the SUBSCRIBE, SUBSCRIBED, UNSUBSCRIBE, UNSUBSCRIBED, PUBLISH, PUBLISHED and ERROR messages for the publish/subscribe pattern, and support for the REGISTER, REGISTERED, UNREGISTER, UNREGISTERED, CALL, RESULT and ERROR messages for the RPC pattern.

For the publish/subscribe pattern, the existing signal/slot mechanism in Qt will probably be used for the user-facing API. For the RPC pattern, asynchronous call results will probably be delivered through a returned future object (e.g. `std::future`) or a callback/lambda. The author suspects some good inspiration for the RPC API can be found in the existing CppWAMP library.

It is expected that a significant amount of time in phase 2 will be spent on testing the library against "known good" WAMP peers. Since there is no official conformance test suite for WAMP available yet, a set of manual test cases will have to be written that exercise the library functionality. The test cases for publish/subscribe will use the library to publish messages to a topic

[2]Exact license for the library is still to be determined.

[3]At `http://github.com/estan/qwamp` which is not yet available at the time of writing.

URI while subscribing to this topic from another peer, and vice versa. The test cases for RPC will use the library to register remote procedures and then call these procedures from another peer, and vice versa. In all cases, the "other peer" will be a program written in Python using the Autobahn|Python WAMP client library developed by Tavendo GmbH [8].

Phase 2 encompasses 15.0 credits of the DD2463 course and corresponds roughly to chapters 5 and 6 of the WAMP specification Basic Profile.

5 Evaluation

The project result will be evaluated by running a set of test programs that exercise the full functionality of the developed library. The results from running the test programs must give a good indication of how well the library conforms to the WAMP specification, and how well it covers the planned functionality.

The first phase, encompassed by the first 5.0 credits of the DD2463 course, is concluded by a brief report being handed over to the examiner. The report covers work done up to that point. The report must include a description of the written code, motivations for the design decisions taken as well as a summary of the result from running test programs that exercise the session management functionality of the library under development. The report must also discuss any obstacles encountered. If planned functionality is missing, the reasons for this must be explained.

The second phase, encompassed by the remaining 10.0 credits of the DD2463 course, is concluded with a slightly longer final report for the entire project being handed over to the examiner, as well as a 10 minute oral presentation. The report should cover the full project work and must include a description of the final library and its design, including motivations for the design decisions, as well as result from running test programs that exercise the session management, publish/subscribe and RPC functionality of the final library. The report should also bring up any deficiencies or unfinished parts in the library, as well as a self-evaluation where the author makes a judgment of his own performance. Finally something should be said about how the project results can be improved in the future.

6 Schedule

The project is to be carried out during the 2015 Autumn term at KTH, stretching from August 31, 2015 to January 18, 2016. Time is allocated to phase 1 and 2 as follows:

- **Phase 1:** August 31, 2015 – October 15, 2015
- **Phase 2:** October 16, 2015 – January 18, 2016

The deadline for the first report is **October 15, 2015, 23:59** while the deadline for the second (final) report is **January 3, 2016, 23:59**. The oral presentation of the project is to be held some time in the days following the second deadline, but no later than **January 18, 2016**.

7 Learning Outcomes

Since the project work is done as part of the DD2463 course at KTH, it has a set of associated learning outcomes. After passing this individualized course, the student will be able to solve an implementation task with limited supervision, and report the result of this implementation task. More specifically, the student will be able to

- obtain and evaluate information applicable to the implementation of the WAMP network protocol using C++ classes available in the Qt C++ framework,
- choose an approach for the implementation and motivate the choices made using well-reasoned arguments based on industry best-practices and what is taught in the computer science program at KTH,
- reflect on aspects of API design and qualities such as usability, maintainability and testability of the solution,
- report on the results both orally (in Swedish or English) and in writing (in English), in a professional manner,
- show increased knowledge in areas of computer science dealing with software architecture, programming interface design and communication protocols.

These project specific learning outcomes correspond roughly to the course learning outcomes as stated in the KTH course catalog.

8 About the Author

Elvis Stansvik is a last year student in the Program System Construction track of the Computer Science program at KTH. He did his Master Thesis project in the Spring of 2015 at Orexplore AB, where he developed remote control and sensing software for a mineral analysis machine using Python. The chosen software architecture was a set of decoupled application components communicating over WAMP. Elvis has also been active in the free software community and participated in the Google Summer of Code program in 2009 and 2011 for the KOffice and Scribus projects, both times using C++ and the Qt framework. He also worked at Nine Lives Games AB during the summers of 2013 and 2014, developing an application level network proxy for an online real-time strategy game using Java. The design of the proxy followed the Actor pattern and used the Vert.x framework.

9 Supervisor / Examiner

Supervisor at KTH for the project is REDACTED while examiner for the course is REDACTED.

References

[1] Alexey Melnikov et al. *WebSocket Protocol Registries at IANA*. 2015. URL: http://www.iana.org/assignments/websocket/websocket.xml (visited on 08/26/2015).

[2] Applied Informatics Software Engineering GmbH. *Official website for the POCO C++ libraries*. 2015. URL: http://pocoproject.org/ (visited on 08/26/2015).

[3] Christopher M. Kohlhoff. *Official website for the Boost.Asio library*. 2015. URL: http://www.boost.org/doc/libs/1_59_0/doc/html/boost_asio.html (visited on 08/26/2015).

[4] Emile Cormier. *Official website for the CppWAMP library*. 2015. URL: https://github.com/ecorm/cppwamp (visited on 08/26/2015).

[5] Ian Fette and Alexey Melnikov. *The WebSocket Protocol*. RFC 6455. RFC Editor, 2011. URL: https://tools.ietf.org/html/rfc6455 (visited on 05/28/2015).

[6] Rafael Stahl. *Official website for the WAMP_POCO library*. 2015. URL: https://github.com/rafzi/WAMP_POCO (visited on 08/26/2015).

[7] Tavendo GmbH. *Official website for the Autobahn|Cpp library*. 2015. URL: http://autobahn.ws/cpp/ (visited on 08/26/2015).

[8] Tavendo GmbH. *Official website for the Autobahn|Python library*. 2015. URL: http://autobahn.ws/python/ (visited on 08/26/2015).

[9] Tavendo GmbH. *The Web Application Messaging Protocol*. 2015. URL: http://wamp.ws/ (visited on 08/26/2015).

[10] The Qt Company. *Official website for the Qt framework*. 2015. URL: http://qt.io/ (visited on 08/26/2015).

[11] Unknown. *Official website for the qtasio library*. 2015. URL: https://github.com/peper0/qtasio (visited on 08/26/2015).