Software Requirements Specification for Pontarius XMPP 0.1 (Third Draft)

The Pontarius Project

27th of July 2011

Contents

1 Introduction

1.1 Purpose

The goal of this document is to clarify—for Pontarius[1] developers, the XMPP community, and to some extent the Haskell community—what we are implementing. We hope that it will help us in the Pontarius project to keep track of functionality and requirements, provide a basis for scheduling, and help us with our validation and verification processes.

1.2 Scope

Pontarius XMPP 0.1 will implement the client capabilities of RFC 6120: XMPP: Core and the depending specifications, such as RFC 6122: XMPP: Address Format, RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, RFC 4422: Simple Authentication and Security Layer (SASL), RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, and Extensible Markup Language (XML) 1.0, among others.

Support for common extensions such as Instant Messaging, Data Forms, Service Discovery, etc. will *not* be included in the 0.1 version, but will be added in later ("0.x") versions. Server components and XMPP server capabilities could also be added later.

While it is the goal of the Pontarius project to develop secure and privacy-aware "personal cloud" solutions on top of Pontarius XMPP, we want Pontarius XMPP to be a general-purpose—and the de facto—XMPP library for Haskell. It should be correct and efficient to work in.

We will not repeat the specifics of the requirements from the RFC 6120: XMPP Core specification and its depending specifications in this document unless we see any special reason to.

1.3 Legal notice

Pontarius XMPP is a free and open source software project. "The Pontarius project" is not a legal entity, but is like a synonym for Jon Kristensen. Jon Kristensen does DOES NOT TAKE ANY RESPONSIBILITY OR OFFER ANY GUARANTEES in regards to the software, its requirements or this document. Furthermore, the software is provided WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

1.4 Definitions, acronyms, and abbrevations

JID JabberID: An address for XMPP entities

Pontarius A free and open source software project that aims to produce XMPP-based, uncentralized, and privacy-aware software solutions

[1] For more information about the Pontarius project, see http://www.pontarius.org/.

PX01 Pontarius XMPP 0.1

REQ Requirement

RFC Request for Comments: A memorandum published by the Internet Engineering Task Force; some of these, including XMPP: Core and XMPP: Address Format, are published as Internet standards

TCP Transmission Control Protocol: A reliable network transport protocol

TLS Transport Layer Security: A secure network protocol, a successor to Secure Sockets Layer (SSL)

XMPP Extendable Messaging and Presence Protocol: An uncentralized, open, and near-real-time presence and messaging protocol

1.5 References

- Extensible Messaging and Presence Protocol (XMPP): Core, RFC 6120, March 2011, Internet Engineering Task Force (see also its depending specifications)

1.6 Overview

The second section provides an overall description of the requirements of PX01, going through the features in a non-strict fashion, talking shortly about the product functions, as well as some constraints and assumptions.

The third section simply lists the requirements, categorized by external interfaces, functions, performance requirements, design constraints, and software system (quality) attributes.

2 Overall Description

2.1 Product perspective

PX01 will be used by XMPP clients to manage presence and messaging in a uncentralized near-real-time environments. For this first milestone of the library, we have chosen to implement only the XMPP: Core specification, and only the client capabilities of it. The reason for this is that we want to get the library out quickly, and want to have the core functionality of the library particularly well tested. The X in XMPP stands for extendable, and Pontarius XMPP must be flexible in regards for extensions, such as RFCs and XEPs; we might end up implementing hundreds of them. This is one of the most important quality attribute of the software.

PX01 is designed to be used with Haskell.

PX01 must work on GNU/Linux, the main Free Software operating system. However, due to the platform support and high-level nature of Haskell, running it on other common operating systems is likely to work as well.

PX01 must work with (at least) the (estimated) most popular free and open source software XMPP server that works with the specifics of our implementation (such as SCRAM).

PX01 depends on the below (free and open source software) Haskell packages. I have omitted specification number [...]. I have also omitted the source, as they are all available on Hackage.

Name (Mnemonic)	Version Number	Purpose
base	N/A	N/A
base64-string	N/A	N/A
binary	N/A	N/A
bytestring	N/A	N/A
containers	N/A	N/A
crypto-api	N/A	N/A
enumerator	N/A	N/A
hslogger	N/A	N/A
network	N/A	N/A
pureMD5	N/A	N/A
QuickCheck	N/A	N/A
random	N/A	N/A
regex-posix	N/A	N/A
text	N/A	N/A
tls	0.4.1	N/A
transformers	N/A	N/A
utf8-string	N/A	N/A
xml-enumerator	N/A	N/A
xml-types	N/A	N/A

Every PX01 client will open up at most one TCP port on the system. PX01 in itself does not write anything to the file system storage of the operating system, with the exception of the optional logging facility, disabled by default, which may be configured to write to disk.

We will utilize *at least* Transport Layer Security to help protect clients using the library from attacks. PX01 will not provide any spam protection.

As we expect a very limited amount of concurrent XMPP clients, and a (relatively) limited actitivity over XMPP streams–even when they are being fully active–we are not specifying any detailed memory or (process) performance requirements for PX01. However, we will stress test the library.

We see no hardware or user interfaces to take into account.

2.2 Product functions

PX01 implements XMPP: Core and allow clients to do roughly the following: Open a TCP connection to a server, exchange XML information with the server to configure the (XML) stream, handle stream errors and encoding issues, have

the connection secured by TLS, authenticate using a SASL mechanism and binding a resource to the stream, as well as sending and receiving so-called XMPP stanzas, certain "top level" XML elements in the stream for communicating messages and presence.

2.3 User characteristics

We expect developers using PX01 to understand the XMPP: Core specification (and its depending specifications), Haskell, and monads, including the StateT monad transformer.

2.4 Constraints

In addition to the requirements in XMPP: Core, XMPP applications should be reliable in that they should be able to be online (active or inactive) for a very long period of time, without problems of memory leaks, unnecessary CPU usage, or similar, arising.

2.4.1 Assumptions and dependencies

We assume that the Glasgow Haskell Compiler (GHC) is available on the system where PX01 applications are built.

3 Specific requirements

3.1 External interfaces

The software is accessible from Haskell and may be configured to do logging.

REQ-1 The system shall be importable and fully functional from Haskell through a full import of the Network.XMPP namespace.

REQ-2 The system shall be able to log information through arbitrary and flexible log handlers.

REQ-3 The system shall run under GNU/Linux.

REQ-4 The system shall work against (at least) the (estimated) most popular free and open source software XMPP server supporting the features that we require (such as SCRAM).

3.2 Functions

If an error arises due to a bug in the software, the system shall throw a runtime exception and the process should exit. If an "acceptable" exception is possible to arise, such as the XMPP server times out, then the related functions should reflect that by being able to return values such as Nothing or null. This allows the XMPP clients to take the appropriate actions.

REQ-5 The system shall be validating input from all functions exposed through any of the system's APIs.

REQ-6 Each incoming stanza should move through a stack of (client) handlers, where each handler may block the stanza from being delivered to handlers further up in the stack. The exceptions to this rule is stanza errors and IQ result stanzas, which may be delivered directly to a callback provided when generating the stanza (and possibly surpressed there).

Rationale: We have so far found it useful to stack XMPP client handlers on top of each other, letting them all manage their particular responsibilities and surpress their messages to handlers further up the stack.

REQ-7 When the client is disconnected, a "disconnect event" is generated in a way similar to in REQ-5. It is moved through the stack handlers; however, these events move down the stacks completely and can't be surpressed. The same thing applies for incoming stream errors, which can either be recoverable or unrecoverable. (?)

Rationale: We want the handlers to all do the appropriate clean-up activities.

REQ-8 The API shall allow for opening a stream to a given IP or host name on a given port, returning either a success value or the reason for failing.

REQ-9 The API shall allow for securing an opened stream with TLS, returning either a success value or the reason for failing.

REQ-10 The API shall allow for authenticating an opened (possibly TLS secured) stream, returning either a success value of the reason for failing. If the credentials were wrong, the system shall allow the client to make as many retries as allowed by the server, without restarting the stream.

REQ-11 The API shall allow for perform resource binding on an authenticated stream, either by trying to set a specific resource, or have the server generate one.

Rationale: Even though most clients wants to do REQ-8, REQ-9, REQ-10, and REQ-11 in one action, some uses of XMPP (such as In-Band Registration) demands more flexibility.

REQ-12 The API shall provide a convenience function for opening a stream, securing the stream with TLS, authenticating an XMPP account, and perform resource binding in one function call, returning either a success value or the reason for failing.

Rationale: This makes the library easier and more efficient to use for most uses cases.

REQ-13 The API shall provide the possibility for clients to close the stream.

REQ-14 The API shall provide the possibility for clients to send stream errors.

REQ-15 The API shall allow a convenient way for "standard disconnect" the client.

REQ-16 The API shall provide a facility for convenient stanza (message, presence, info/query) creation, eliminating the risk of illegal stanzas where feasible.

REQ-17 The API shall allow for convenient construction of JabberIDs.

REQ-18 The API shall provide utility functions to check whether or not a JID is full or bare.

REQ-19 The API shall provide conversion functions to convert from a string to ("Maybe") JID, and JID to string.

REQ-20 The API shall provide an optional way for clients to receive time-out events on requests made, such as an IQ or connection attempt. The time-out interval should be customizable.

REQ-21 The library should generate an internal infinite list of unique stanza IDs; the API should provide a way for application developers to acquire any amount of such IDs

REQ-22 The system must offer timeout callbacks to be called if an asynchronous result is not guaranteed to be produced in a timely fashion.

REQ-23 The system must a convenient API to deal with stanza and stream errors.

3.3 Performance requirements

There is only one XMPP client per instance of the system.

REQ-24 Regular desktop computers should be able to run hundreds of Pontarius XMPP 0.1 clients.

REQ-25 Pontarius XMPP 0.1 should support virtually as many stanzas per second as (non-throttled) XMPP servers are able to route. This goes for both lightweight, heavy and mixed stanzas.

REQ-26 Processing (parsing, generating, and firing the event) a received stanza should take at most 0.01 seconds.

3.4 Design constraints

3.4.1 RFC 6120: XMPP: Core

REQ-27 The system shall support one persistant TCP stream/connection between the XMPP client and the XMPP server.

REQ-28 The system shall implement (at least) the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite for TLS.

REQ-29 The system shall support (at least) the SHA-1 variant of SASL Salted Challenge Response Authentication Mechanism (SCRAM-SHA-1).

REQ-30 XML namespaces for stanzas should always be known to the client.

REQ-31 We must validate incoming XML (though not against the XML schemas).

REQ-32 The system shall always check for the appropriate features before trying to use them.

REQ-33 End-to-end presence or anything else presence-related defined outside of XMPP: Core (such as in XMPP: Instant Messaging) is *not* supported.

3.4.2 RFC 6122: XMPP: Address Format

REQ-34 JIDs should be validated, transformed, and internationalized in accordance with the stringprep profiles Nodeprep, Nameprep, and Resourceprep.

REQ-35 JIDs should be able to use hostnames, IPv4 addresses, and IPv6 addresses, as domainparts.

3.4.3 Standards compliance

REQ-36 The project and its source code shall adhere to the guidelines prestented in the guidelines found at http://www.haskell.org/haskellwiki/Programming_guidelines.

3.5 Software system attributes

REQ-37 The system shall be *extendable*; it must be flexible in regards for extensions, such as RFCs and XEPs.

REQ-38 The system shall be *reliable*; it should produce not crash, lag behind, or get some data corruption under very heavy and lengthy use.

REQ-39 The system shall be *secure*; steps should be taken to protect the clients against man-in-the-middle attacks and the like.