Universita' degli Studi di Messina
Dipartimento di Matematica e Informatica

Programming II project

# NetLayer

11 June 2015

**Authors:**
Vittorio Romeo

**Professors:**
Massimo Vilari

http://vittoriomeo.info


http://unime.it

# Contents

# Part I

# Project specifications

The following part of the document describes the project and its design/development process without exploring its implementation details.

The part begins with a synthesis of the **client request**. After a careful analysis of the request, a **Software Requirements Specification** (SRS) was written.

Writing a correct and informative SRS is of utmost importance to achieve an high-quality final product and ensuring the development process goes smoothly.

The SRS will cover the following points in depth:

- **Scope and purpose**.
- **Feature and functions**.
- **External interface requirements**.
- **Functional requirements**.
- **Example use cases**.
- **Non-functional requirements**.
- **Analysis models**.

# Chapter 1

# Client request

The client requests the design and implementation of an **open-source multi-purpose C++14 networking library**.

The library must allow the client to develop its own **server-client** architectures and applications with ease, while still being performant and allowing low-level operations if required.

The client intends to use the library as a basis for the networking layer in applications belonging to different domains, ranging from **chat web applications** to **real-time games and simulations**.

The library must fulfill the following requirements:

- The library must be written in **modern C++14**, making use of the latest features to improve performance, readability and flexibility.
- The library must target **UNIX** systems, **Windows** and **MacOS**.
- The library must have a **layered architecture**, allowing developers using it to go as low-level/high-level they desire.
- The library must deal with **byte serialization** of native and user-defined classes. Nested serializable data structures must be supported.
- The library must provide a generic **tunnel abstraction** that represents a network entity providing and receiving data. A UDP socket tunnel implementation must be provided with the library.
- The library must provide an high-level abstraction for **server-client** multithreaded architectures, allowing applications to asynchronously interact with any number of sockets and conveniently handle received packets via function dispatching.

- The library must provide metaprogramming facilities to generate and bind packet types at compile-time, allowing performant code generation for serialization/deserialization and communication.

- The library must be released under an **open-source** license and promote collaboration and external contributions.

The client intends using the requested library **to build platforms** for various projects, both for internal company usage and public usage.

It is imperative for the library to be easily integrable with existing legacy system, such as architectures depending on relational databases.

For ease of development and deployment, the client requested the library to be optionally usable in **header-only** mode and compatibility with the **CMake** build system.

The abstraction provided by the library must work asynchronously by default, but an option to use blocking IO must be present.

# Chapter 2

# Software Requirements Specification

## 1. Introduction

### 1..1 Software engineering

**Software engineering** is the study and an application of engineering to the design, development, and maintenance of software.

The Bureau of Labor Statistics' definition is Research, design, develop, and test operating systems-level software, compilers, and network distribution software for medical, industrial, military, communications, aerospace, business, scientific, and general computing applications.

Typical formal definitions of software engineering are:

- The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.
- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.
- An engineering discipline that is concerned with all aspects of software production.
- The establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines.

#### 1..1.1 Background

The term **software engineering** goes back to the '60s, when more complex programs started to be developed by teams composed by experts.

There was a radical transformation of software: from **artisan product** to **industrial product**.

A software engineer needs to be a good programmer, an algorithm and data structures expert with good knowledge of one or more programming languages.

He needs to know various design processes, must have the ability to convert generic requirements in well-detailed and accurate specifications, and needs to be able to communicate with the end-user in a language comprehensible to him comprehensible.

Software engineering, is, however, a discipline that's still evolving. There still are no definitive standards for the software development process.

Compared to traditional engineering, which is based upon mathematics and solid methods and where well-defined standards need to be followed, software engineering is greatly dependent on personal experience rather than mathematical tools.

Here's a brief history of software engineering:

- **1950s**: Computers start to be used extensively in business applications.
- **1960s**: The first software product is marketed.

  IBM announces its unbundling in June 1969.
- **1970s**: Software products are now regularly bought by normal users.

  The software development industry grows rapidly despite the lack of financing.

  The first software houses begin to emerge.

### 1..1.2 Differences with programming

- A programmer writes a complete program.
- A software engineer writes a software component that will be combined with components written by other software engineers to build a system.
- Programming is primarily a personal activity.
- Software engineering is essentially a team activity.
- Programming is just one aspect of software development.
- Large software systems must be developed similar to other engineering practices.

## 1..2 SRS

This **Software Requirements Specification** (SRS) chapter contains all the information needed by software engineers and project managers to design and implement the requested forum creation/management framework.

The SRS was written following the **Institute of Electrical and Electronics Engineers** (IEEE) guidelines on SRS creation.

## 1..3 Purpose

The SRS chapter is contained in the **non-technical** part of the thesis.

Its purpose is providing a **comprehensive description** of the objective and environment for the software under development.

The SRS fully describes **what the software will do** and **how it will be expected to perform**.

## 1..4 Scope

### 1..4.1 Identity

The software that will be designed and produced will be called **NetLayer**.

### 1..4.2 Feature extents

The complete product will:

- Provide a library for the **development of multi-purpose network applications and architectures**.
- Provide abstractions for all the major **operating systems' networking layer**.
- Provide an extensible and flexible **data serialization** module for primitive and user-defined classes.

NetLayer, however, will not be a complete framework for the development of applications. Every part of an application that does not deal with networking issues will not be covered by the product.

### 1..4.3 Benefits and objectives

Development using NetLayer will give companies and individuals several benefits over from-scratch development.

- Usage of NetLayer will provide access to an **easy-to-integrate** and **easy-to-use** networking library.
- Development and testing time will be **significantly reduced**.
- Code making use of the library will be **modern, efficient and readable** thanks to C++14 features and abstractions.

# 2. General description

## 2..1 Product perspective and functions

The product shares many basic aspects and features with existing networking libraries, improving upon them in the following ways:

- A layer-based architecture allows developers to make use of both low-level constructs and operations and high-level abstractions in the same application.
- The library will optionally allow developers to use a programming style similar to **functional programming**, making use of callbacks and first-class functions to deal with packet management and function dispatching.
-

## 2..2 User characteristics

NetLayer is targeted towards modern C++ developers experienced with C++11 and C++14 features. The library makes heavy use of modern metaprogramming paradigms and techniques - unfamiliar users will not be able to make full use of the library.

A m̈ore functionalïnterface is provided where possible, allowing users to use convenient abstractions for data serialization functions and networking functions.

Familiarity with multithreading and synchronous computation is also required to use the library.

# 3. Glossary

The following list contains all the main elements that compose the architecture of NetLayer.

- **Packet Buffer**: dynamically resizable buffer that can store and provide serialized generic data.
- **Address**: union of an IP address and a port.
- **Payload**: abstraction consisting of an Address and PcktBuf. It can be sent to and received by Tunnel instances.
- **Tunnel**: abstraction of a Payload provider/receiver. The default Tunnel is an UDP socket.

- **Thread Safe Queue**: a lock-based thread safe queue that supports concurrent enqueueing and dequeueing.
- **Managed Packet Buffer**: abstraction consisting of a Thread Safe Queue and a reference to a Tunnel. It can be either a **Managed Receive Buffer**, which enqueues received data from the tunnel, or a **Managed Send Buffer**, which enqueues data that will be sent through the tunnel.
- **Managed Host**: union of an Address, a Tunnel, a Managed Receive Buffer and a Managed Send Buffer. Represents a network entity capable of sending and receiving data through a tunnel.
- **Serializable**: abstraction over a tuple of generic types that automatically allows the user to serialize and deserialize data. Serializable packets can also be nested and contain dynamically-resizable data structures.
- **Packet Bind**: compile-time bind of a Serializable to a Packet type. Used to generate a dispatch table.
- **Dispatch Table**: compile-time function table that binds a function to specific packet binds. Used to handle received packets.
- **Context Managed Host**: union of a managed host and a dispatch table.

# 4. Specific requirements

## 4..1 External interface requirements

**External interface requirements** identify and document the interfaces to other systems and external entities within the project scope.

### 4..1.1 User interfaces

The product will not provide any graphical user interface. The users of the library will be able to access its functions and types using C++14.

### 4..1.2 Software interfaces

The **open-source policy** of NetLayer will allow its users to expand or improve existing functionality and to interact with other existing technologies.

## 4..2 Functional requirements

In software engineering, a **functional requirement** defines a function of a system and its components.

Functional requirements may be **calculations**, **technical details**, **data manipulation and processing** and other specific functionality that define what a system is supposed to accomplish.

Behavioral requirements describing all the cases where the system uses the functional requirements are captured in **use cases**.

### 4..2.1 User/group management

- **Users**: users will be managed by the system. Users can register (or be manually added by an administrator). Registration can be configured to require a confirmation email or not.
- **Groups**: every user will be part of at least one group at all times. Groups are part of an hierarchy: they can inherit from each other. Groups can have permissions specific to sections and system-wide permissions.

Figure 2.1: User/group hierarchy example.

### 4..2.2 Content hierarchy

- **Posts**: posts will be the base of the content hierarchy. They will contain HTML-enabled text and any number of attachments. Posts can be edited and deleted by the original owner.
- **Threads**: threads are groups of posts. Users with the correct permissions can create a thread in a specific section and have other users add posts or subscribe to it. Threads can be edited and deleted by the original owner.
- **Sections**: sections are content containers intended to group threads related to the same subject. Forum administrators and moderators can create sections and give users permissions to view or edit them.
- **Attachments**: users with the correct permissions can upload files and attach any number of them to one of their posts.

Figure 2.2: Content hierarchy example.

#### 4..2.3 Content tracking system

- **Creation data**: user-created content (posts, threads,attachments, etc) will have some data specific to its creation that can be extended by forum administrators. Basic predefined data will consist of creation date and time. It will be possible to run statistical queries on content creation data.
- **Subscriptions**: users and moderators will be able to subscribe to specific sections, threads or user to track their contents. They will receive real-time notifications upon addition/editing of tracked content.

Figure 2.3: Subscription/notification architecture example.

## 4..3 Example use cases

In software and systems engineering, a **use case** is a list of steps, typically defining interactions between one or more actors and a system, to achieve a goal.

### 4..3.1 Mobile game forum

A company developed a popular mobile game, with a wide audience. The company uses the **veeForum framework** to give users a place to discuss game strategy, give feedback on the quality of their product and receive technical support.

#### 4..3.1.1 Actors

- Game developers.
- Game players.
- Forum management team.
- Technical support team.
- Feedback (PR) team.

#### 4..3.1.2 Pre-conditions

- Release of a popular product with a wide audience.
- Game users need to register on the forum.

#### 4..3.1.3 Flow of events

- Installation and configuration of a veeForum-enabled forum system by the forum management team.
- Creation of the sections and permission hierarchies by the forum management team and the developers.
- Registration and content creation by the game developers and game players.

#### 4..3.1.4 Post-conditions

- Game players will be able to share their strategies and thoughts on the product.
- The technical support team will find all technical issues grouped in a convenient way and will be able to track individual issues. Technical support members will be able to communicate with each other in a private section.
- The feedback team will be able to track user suggestions and forward potential product improvements to the developer team.

Figure 2.4: Basic forum usage use case diagram.

Figure 2.5: Technical support and feedback use case diagram.

#### 4..3.2 Local city GNU/Linux usergroup forum

Some GNU/Linux users from the same city decide to start a local usergroup to discuss the GNU/Linux ecosystem and make new friends. In spirit with the open-source nature of the system, collaboration is extremely important. They require to easily assign specific permissions to users and groups to allow the forum to grow and be well-organized.

##### 4..3.2.1 Actors

- Usergroup creators.
- Usergroup members.
- External visitors.

##### 4..3.2.2 Pre-conditions

- Interest in a local GNU/Linux usergroup.
- Availability of people willing to collaborate.

##### 4..3.2.3 Flow of events

- Installation and configuration of a veeForum-enabled forum system by the usergroup creators.
- Creation of the initial sections and permission hierarchies by the usergroup creators.
- Registration of usergroup members and external visitors.
- The usergroup creators give other usergroup members permissions to create and manage sections and users, starting a chain of collaborative forum content development.
- Usergroup members and external visitors contribute and make use of the content.

##### 4..3.2.4 Post-conditions

- Local city usergroup members will be able to get to know and speak to each other.
- Usergroups members willing to contribute will be able to easily manage sections and write posts/articles.
- External visitors will be able to make use of the public content.

Figure 2.6: Usergroup forum use case diagram.

## 4..4 Non-functional requirements

Functional requirements are supported by **non-functional requirements** (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability).

### 4..4.1 Performance

The system will be designed from the ground-up with emphasis on performance. As the forum may have huge amounts of contents and concurrent usage after its deployment, optimizing is a must.

When possible, functions will be implemented **directly in the database**, for maximum performance.

Web backend functions will also be carefully **optimized both for memory and speed**.

### 4..4.2 Reliability

The system will have to be reliable and keep working in case of errors.

Database queries and functions will be executed in **safe wrappers** that catch and handle errors carefully.

### 4..4.3 Security

veeForum needs to guarantee privacy and security for users and administrator of the system.

Well-tested and well-received **security idioms** and **encryption algorithms** will have to be used throughout the implementation of the whole system.

### 4..4.4 Maintainability and portability

Being an open-source project, **maintainability**, **extensibility** and **portability** are key.

The code layer will be carefully designed and organized to allow easy maintenance, bug-fixing and feature addition.

To ensure maximum portability, the product will be designed to work on the most popular **GNU/Linux** distributions and will be thoroughly tested on different platforms.

# 5. Analysis models

## 5..1 Activity diagrams

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organisational processes (i.e. workflows). Activity diagrams show the overall flow of control.

The following diagram shows the steps that the **forum management team** must take in order to setup and initialize a veeForum-enabled forum.

Figure 2.7: Forum setup and initialization activity diagram.

The following diagram shows the steps required for **user registration**

Figure 2.8: User registration activity diagram.

The following diagram shows the steps required for **user authentication**

Figure 2.9: User authentication activity diagram.

The following diagrams shows the steps that the **forum users** must take in order to add content to the forum system.

Figure 2.10: Content creation activity diagram - 0.

Figure 2.11: Content creation activity diagram - 1.

The following diagram shows the steps required for **post content editing**

Figure 2.12: Post content editing activity diagram.

The following diagram shows the steps required for **attachment creation**

Figure 2.13: Attachment creation activity diagram.

The following diagram shows the steps required for **content subscription**

Figure 2.14: Content subscription activity diagram.

The following diagram shows the steps that the **forum system** must take in order to validate a permission bitset for a specific action.

Figure 2.15: Recursive permission validation algorithm.

## 5..2 Class diagrams

**Class diagrams** are created using UML.

The **Unified Modeling Language** (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

It offers a way to visualize a system's architectural blueprints in a diagram, including elements such as:

- Any activities (jobs).
- Individual components of the system.
- And how they can interact with other software components.
- How the system will run.
- How entities interact with others (components and interfaces).
- External user interface.

Figure 2.16: Database entities UML diagram.

Figure 2.17: Server backend UML diagram.

## 5..3 Sequence diagrams

The following diagram shows the interaction between **forum users**, the **subscription broker** and the **content management** systen in order to manage subscriptions and generate notifications.

Figure 2.18: Subscription/notification system sequence diagram.

The following diagram shows the interaction between the backend and the database.

Figure 2.19: Backend-database interaction sequence diagram.

The following diagram shows the interaction between the backend and database stored procedures.

Figure 2.20: Backend-stored procedures interaction sequence diagram.

The following diagram shows the interaction between the authentication system and the database.

Figure 2.21: Authentication system-database interaction sequence diagram.

## 5..4 Deployment diagram

A **deployment diagram** in the Unified Modeling Language models the physical deployment of artifacts on nodes.

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have subnodes, which appear as nested boxes. Device nodes are physical computing resources with processing memory and services to execute software, such as typical computers or mobile phones.

Figure 2.22: Deployment diagram for veeForum.