KARABÜK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

SENIOR PROJECT

CCIDE: COLLABORATIVE C++ IDE

Proje Öğrencisi

2016010205023 – Bedir Kaan TAZE

Öğretim Üyesi:

Dr. Öğr. Üyesi Burhan SELÇUK

2020

- **PROJE KONUSU ÖZETİ**

Teknolojinin gelişmesiyle kullanıcı dostu ürünlerin çıkması giderek kolaylaşmış ve bu kolaylık sayesinde neredeyse her gün yeni bir ürün çıkacak düzeye gelmiştir. Yazılım alanı söz konusu olduğunda ise ürünlerin üretimi çok daha hızlı olmaktadır. Hayatımızı kolaylaştıran yazılımların artması, alternatiflerinin çok sayıda olması yazılım geliştirme takımlarının oluşmasında ve projelerin gerçekleştirilmesinde sıkıntı çıkarmaya başlamıştır. Farklı ürünlerin uyumsuzluğu, platforma özgü yazılımlar gibi engeller projelerin aksamasına, büyük zaman ve maddi kayıplara yol açabilmektedir. Bu çalışmada C++ yazılım dili üzerine çalışma yapan yazılım mühendislerinin efektif çalışabilmeleri hedeflenmiş; kod yazımı, anlık yazılı ve sesli mesajlaşma, versiyon kontrolü ve takım içi kod kontrolünü en efektif ve 3. parti yazılımlara gerek duymadan tek bir ortam üzerinden yapmaları sağlanmıştır.

- **GİRİŞ VE LİTERATÜR ÖZETİ**

Integrated Development Environment (IDE) yazılım ürünlerinin üretilmesini sağlayan yazılıma denir. Bir IDE bir yazılım diline özel ya da birden çok dili üzerine üretilmiş olabilir. IDE kullanan kişiler hazırladıkları kaynak kodları IDE içindeki metin alanına yazarak programlama diline göre derler veya çalıştırırlar. IDE içinde bulunması halinde çeşitli eklentilerle farklı dil desteği, versiyon kontrolü gibi özellikler sağlanabilmektedir.[1]

IDE'ler console/terminal'lerin çıkmasıyla literatüre girmiştir. Bunun sebebi, terminal'den önce programların akış diyagramlarıyla hazırlanıp punch-card'lara işlenmesiyle çalıştırılmasından dolayıdır. Terminaller programlamanın kolaylaştırılması konusunda çığır açmıştır. 1975 yılında Softlab Munich, Maestro I'i geliştirerek ilk IDE'yi hizmete sunmuştur.[2]

- **ARAŞTIRMA PROJESİ KAPSAMI/YÖNTEMLER**

Collaborative C++ IDE, geniş kapsamlı bir geliştirme ortamıdır. Klasik ortamlarda bulunan metin editörü dışında anlık mesajlaşma, sesli mesajlaşma, ortak metin düzenleme, syntax renklendirme, GitHub ile versiyon kontrolü bulunmaktadır ve platformdan bağımsızdır.

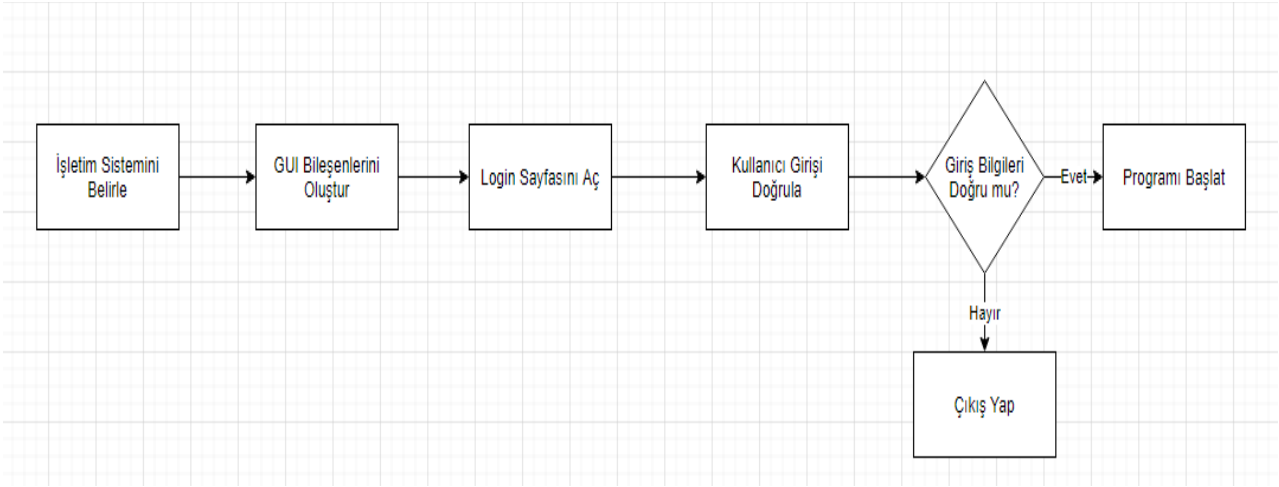
CCIDE geliştirilirken C++ programlama dili ve beraberinde Qt Framework'ü kullanılmıştır. C++ dilinin seçilme nedeni olarak makine diline yakın olması ve pointerlar ile hafıza yönetiminin esnekliği verilebilir. IDE'nin içinde yazılan projeden bağımsız olarak yüksek efektifte çalışması son derece önemlidir. Derleyiciyi çağırmak, derleyici sonuçlarını almak gibi birçok temelinde kompleks olan işlemi yalıtımı (abstraction) en az şekilde tutmak elzemdir.

IDE için görsel programlama işleri Qt Framework ile yapılmıştır. Qt Framework C++ dili üzerine görsel programlama yapılmasını sağlayan bir framework'tür ve platformdan bağımsız olarak sunduğu textbox gibi tüm komponentler istenilen düzeyde bir yalıtkanlıkla kullanıcı kullanıma sunulduğundan tercih edilmiştir. Her işletim sisteminin her versiyonuna ayrı tanımlama yapmak yerine Qt ile tek seferde kod yazılıp tüm platformlara uygun şekilde pencereler oluşturulmaktadır. [3]

Program tasarımında client-server mimarisi tercih edilmiştir. Üye işlemleri, ortak metin düzenleme, sesli mesaj gönderimi ve anlık mesajlaşma operasyonlarının gerçekleştirilmesi için merkezi bir sisteme ihtiyaç duyulmaktadır. Her kullanıcı programı açtığında açtıkları program bir client olarak server ile bağlantı kurar. Üye işlemleri ve sisteme giriş server'da MySQL veritabanında tutulmaktadır. Mesajlaşma için ise TCP/IP'ye uygun olacak şekilde iki parti arasında iletişim kurulmaktadır. Ortak metin düzenleme özelliğinde ise server açılan dosyalarda çakışma olmasını engellemek adına bir moderatör görevi görür, dosyanın son halini client'lar ile paylaşır.

I. Client'ların Başlatılması

CCIDE platform bağımsız bir programdır. Qt Framework'ün yardımı sayesinde her işletim sisteminde çalışabilmektedir. Özet olarak program başlarken önce pencereler işletim sistemine göre oluşturulur. Sonrasında kullanıcılardan giriş yapması istenir. Doğru bilgilerin girilmesi durumunda program başlatılır.



Şekil 1 – CCIDE Açılış Şeması

Client programının başlatılması main fonksiyonunun çağırılması ile başlar. Başlatılan program öncelikle bağlı olduğu bilgisayarın Network ayarlarını okur. Server bağlantısı network ayarlarının okunmasından sonra server ile bir session (oturum) açılır. Network ayarlarının kaydedilmesi ise belirli bi SSID'ye sahip wireless bir network'ün sadece belirli bir bölgede erişilebilir olabileceğinden dolayı gereksiz veya zararlı olabilecek ağları filtrelemek ve doğru konfigürasyon ile server'a bağlanılmasından emin olmak içindir.

```
QNetworkConfigurationManager manager;
if (manager.capabilities() & QNetworkConfigurationManager::NetworkSessionRequired) {
    // Get saved network configuration
```

Şekil 2 – Network Ayarlarının Oluşturulması

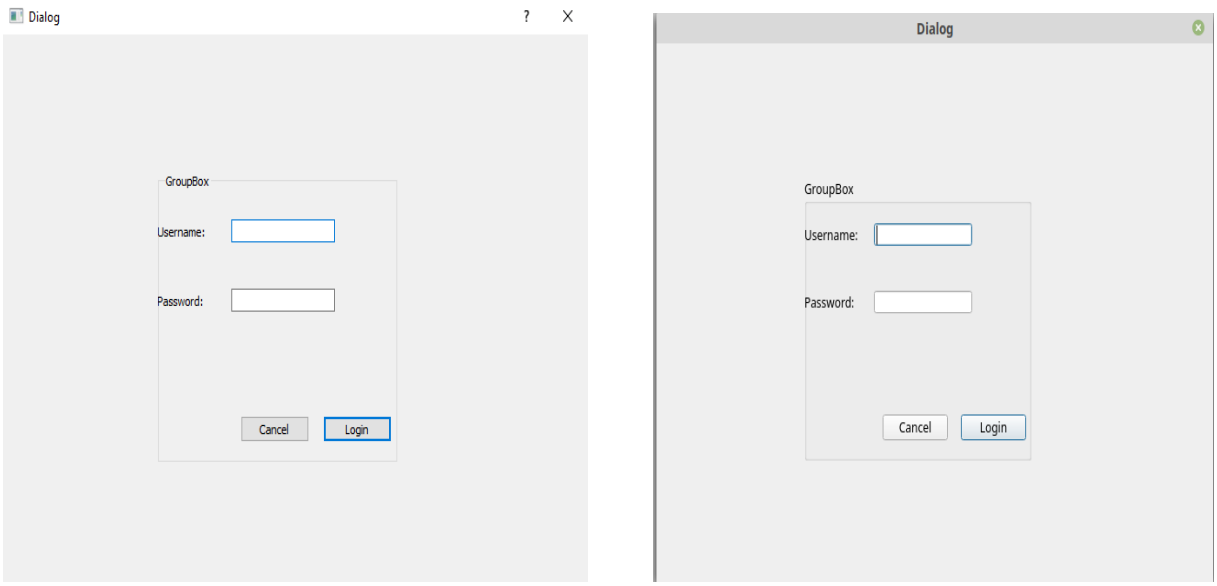
```
QNetworkSession *networkSession = new QNetworkSession(config, &a);
networkSession->open();
networkSession->waitForOpened();

if (networkSession->isOpen()) {
    // Save the used configuration
    QNetworkConfiguration config = networkSession->configuration();
    QString id;
    if (config.type() == QNetworkConfiguration::UserChoice) {
        id = networkSession->sessionProperty(
                QLatin1String("UserChoiceConfiguration")).toString();
    } else {
        id = config.identifier();
    }

    QSettings settings(QSettings::UserScope, QLatin1String("QtProject"));
    settings.beginGroup(QLatin1String("QtNetwork"));
    settings.setValue(QLatin1String("DefaultNetworkConfiguration"), id);
    settings.endGroup();
```

Şekil 3 – Kaydedilen Ayarlarla Session Oluşturulması

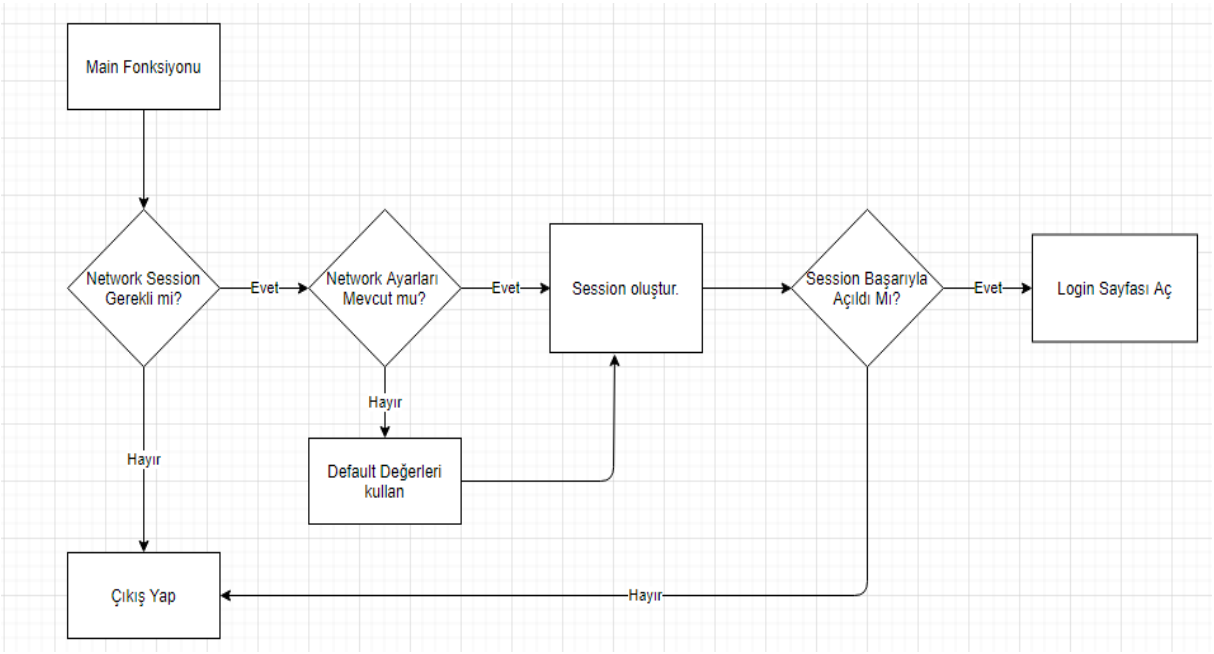
Server ile bağlantı kurulduktan sonra Login ekranı açılarak kullanıcı adı ve şifre istenir. Girilen değerler server'daki MySQL veritabanındaki değerlerle karşılaştırılır.

```
Login login;
login.show();
```

Şekil 4 – Login Ekranının Başlatılması



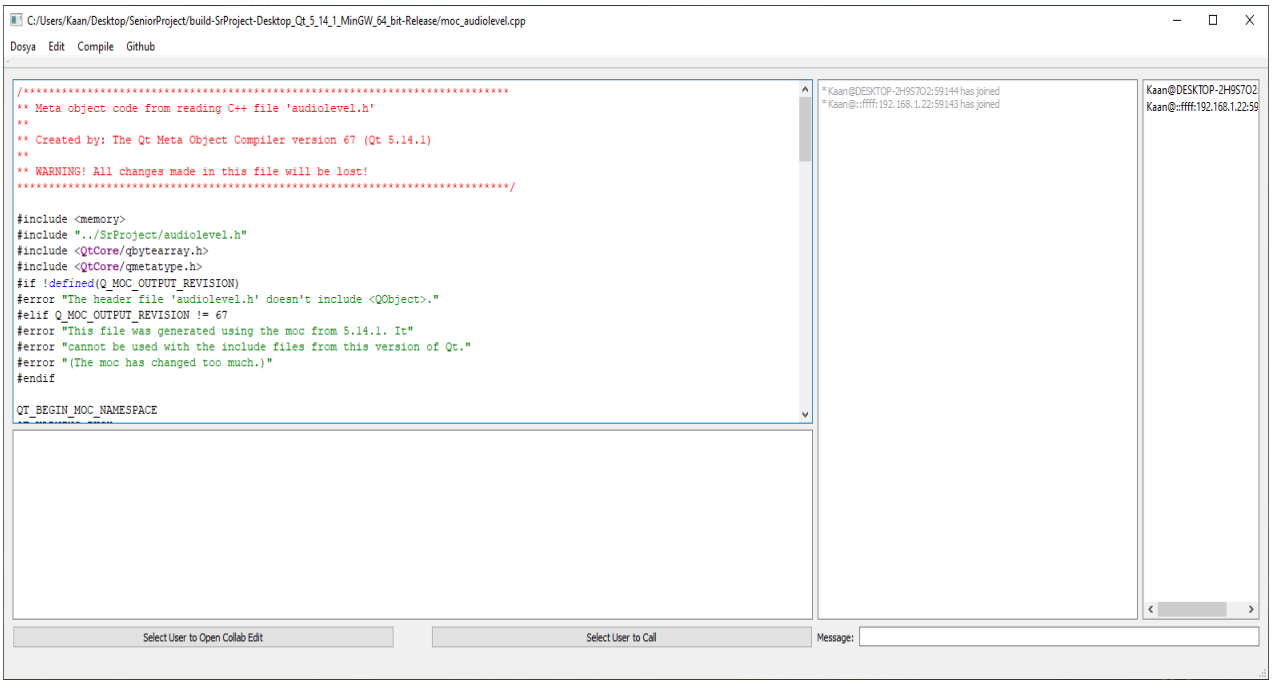
Şekil 5 – Login Ekranının Windows ve Linux Ortamlarında Görünümü



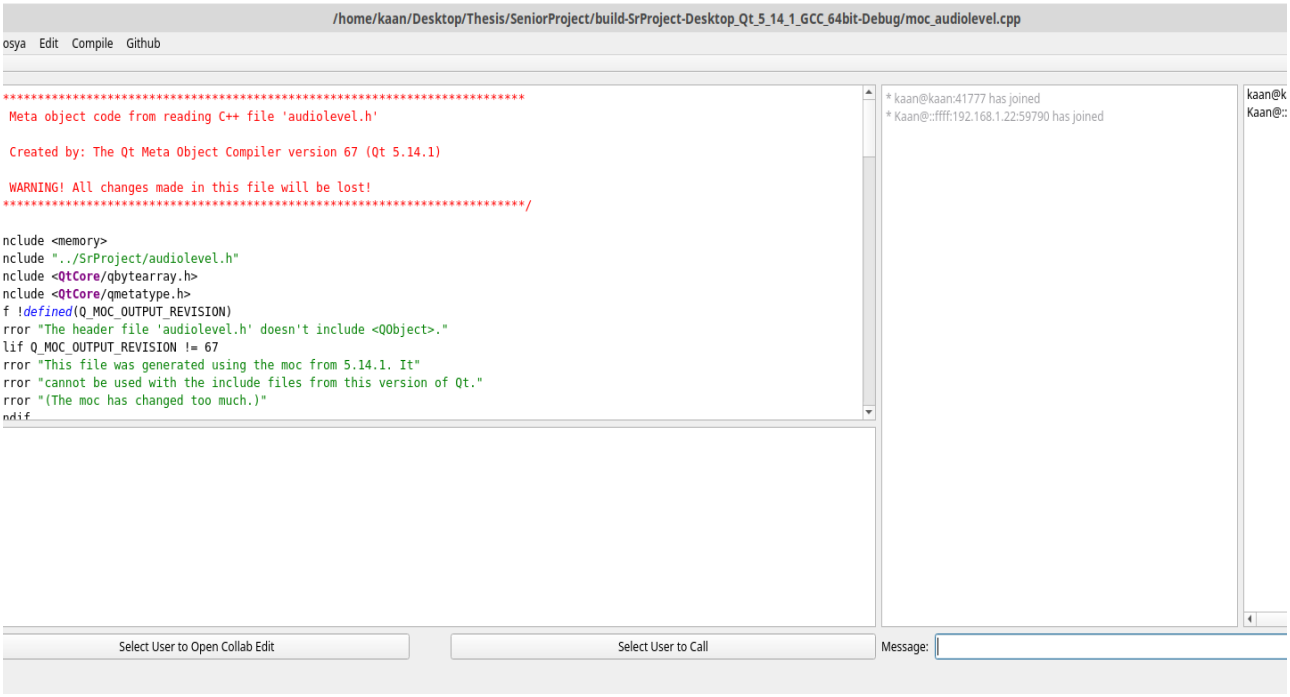
Şekil 6 Server ile Oturum Açılması

```
if(db.open()){
    QString command = "SELECT * FROM user WHERE username = '" + enteredUsername + "' AND password = '" + enteredPass + "'";
    QSqlQuery query;
    if(query.exec("SELECT * FROM 'user'")){
        if(query.size()>0){
            Login::hide();
            mainWindow = new MainWindow(this);
            mainWindow->show();
        }
        else {
             QMessageBox::information(this,"Login Failed", "Username or Password is wrong");
        }
```

Şekil 7 – Kullanıcı Bilgilerinin Kontrolü



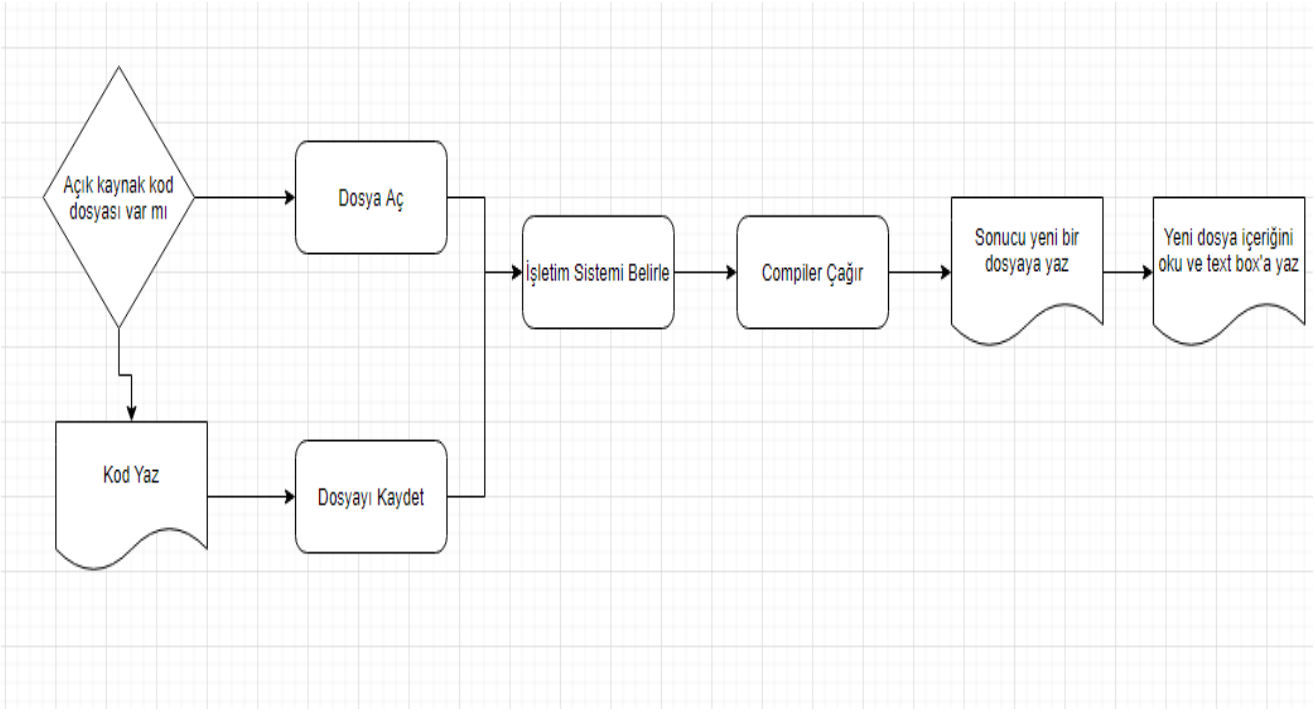
Şekil 8 – Windows Client



Şekil 9 – Linux Client

II. Kodun Derlenmesi

CCIDE içinde compiler barındırmayan bir IDE'dir. Düzgün çalışması için işletim sistemine bir adet compiler yüklenmesi gerekmektedir. Kullanıcılar textbox'a kodlarını yazdıktan sonra dosya işlemleri ile local olarak kaydedilir. İşletim istemi belirlenerek hangi komutların çalıştırılacağı belirlenir. Çalıştırılacak sistem komutları Windows işletim sistemleri ve LINUX/UNIX tabanlı sistemlerde farklılık göstermektedir. Sistem çağrısından sonra derlenmiş programın çıktısı başka bir dosya içine kaydedilir. Dosya işlemleri ile program çıktısı okunur ve CCIDE içinde program çıktısının gösterileceği salt okunur textbox içinde gösterilir.



```
void MainWindow::on_actionCompile_Code_triggered()
{
    QProcess compile;
    QString fileNameOutput = QFileInfo(currentFile).fileName().mid(0, QFileInfo(currentFile).fileName().indexOf("."));
    QString path = QFileInfo(currentFile).filePath().mid(0, QFileInfo(currentFile).filePath().indexOf("."));
#ifdef Q_OS_WIN
        compile.startDetached("powershell", QStringList() << "g++ " + QFileInfo(currentFile).fileName() +" -o " + fileNameOutput);
        if (compile.error()){
            ui->textEdit_3->setText(compile.errorString());
        }
        compile.kill();
        compile.execute("powershell " + path + ".exe | Out-File ./output");
#else
        compile.start("g++ " + QFileInfo(currentFile).fileName() +" -o " + fileNameOutput);
        if (compile.error()){
            ui->textEdit_3->setText(compile.errorString());
        }
        compile.kill();
        compile.execute(path + ".exe > ./output");
#endif
    on_actionOnCompileFinished_triggered();
```

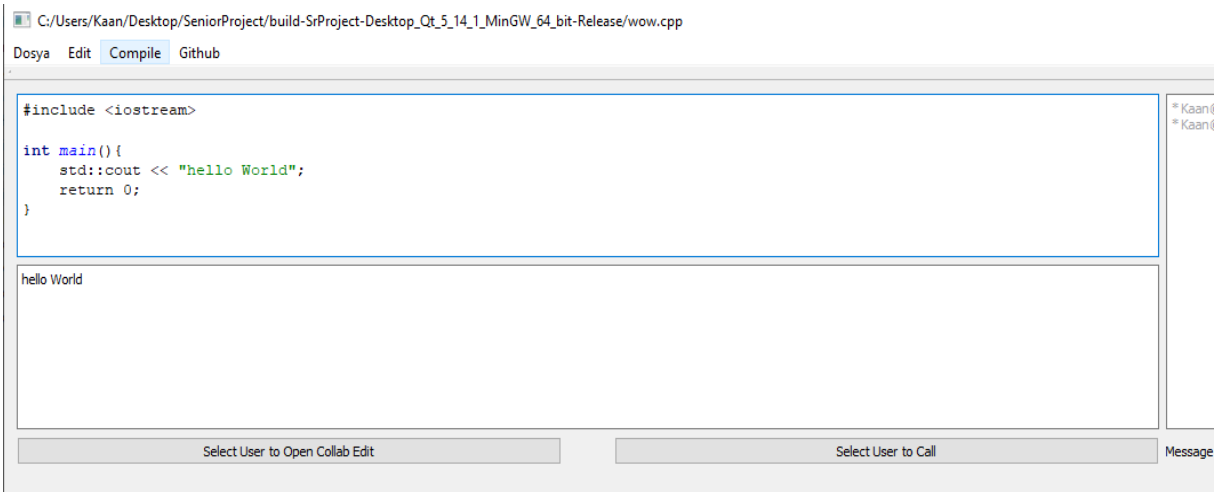
Şekil 10 – Kod Derlenmesi için Compiler Çağrımı

Şekil 10'da görüldüğü üzere önce dosyanın konumu belirlenmiş ve dosya adı çağıralacak sistem komutuna göre düzenlenmiştir. Sonrasında ise macro ile işletim sistemine göre çalıştırılacak kodlar yazılmıştır. Compile adında bir process üretilmiş, bu process kendi içinde iki adet thread ile hem kodu derleyip hem de derlenmiş kodun çıktısını almaktadır.

```
QTextStream in(&outputFile);
QString text = in.readAll();
ui->textEdit_3->setPlainText(text);
outputFile.close();
```

Şekil 11 – Çıtkının Ekrana Yazılması

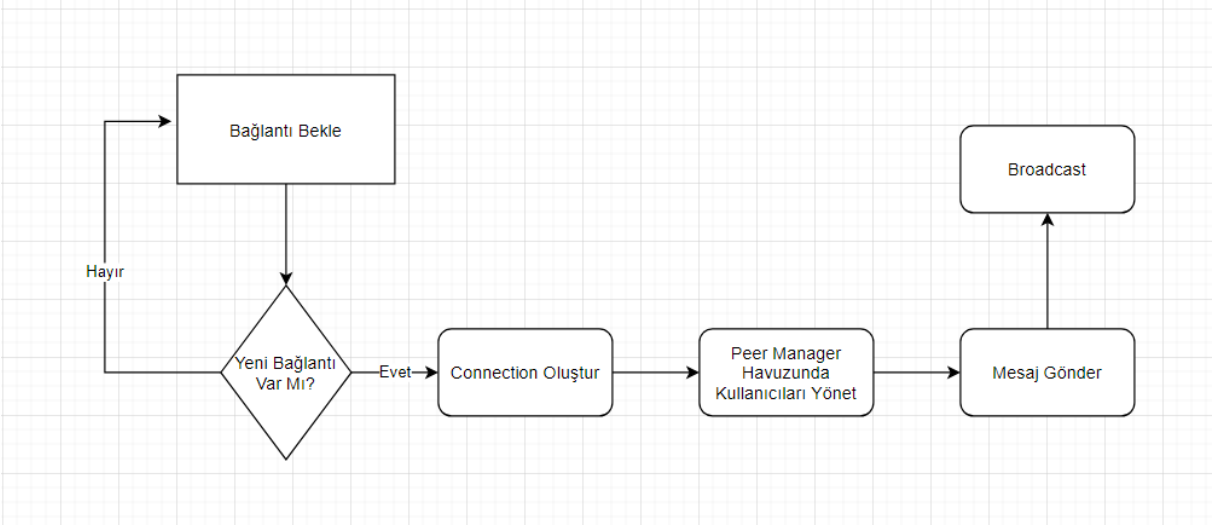
Derlenen program outputFile adında bir dosyaya kaydedilmekte ve dosya işlemleriyle text box'lara yazılmaktadır.



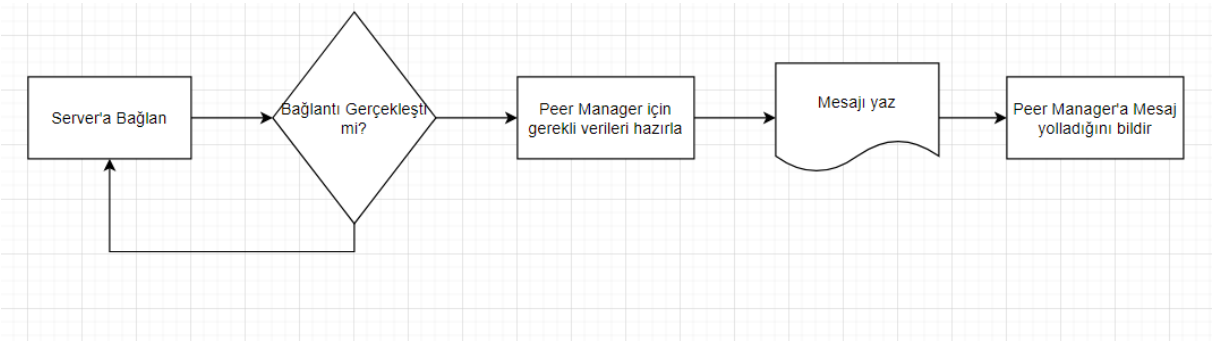
Şekil 12 – Hello World Programının Derlenmesi

III. Anlık Mesajlaşma

Anlık mesajlaşma Transmission Control Protocol/Internet Protocol üzerine inşa edilmiştir. Başlatılan her client programı server'a bağlanırken server içinde bir havuza abone olurlar. Server sürekli dinleme halindedir ve socketDestriptor denilen tanımlayıcıları dikkate alarak yeni bir "Connection" oluşturur. Server'da her client'in operasyonlarıyla ilgilenen PeerManager, her bir client'ın server'ın hangi portuna bağlandığını, server içindeki kullanıcı adını ve yazdığı mesajların broadcast edilmesini sağlar. Online olan tüm kullanıcılar bir tarafta listelenirken bir başka pencerede anlık olarak mesajalaşabilmektedirler.



Şekil 13 - Server'ın mesajları işlemesi



Şekil 14 – Client'ın Mesaj Yollaması

```
Server::Server(QObject *parent)
    : QTcpServer(parent)
{
    listen(QHostAddress::Any);
}

void Server::incomingConnection(qintptr socketDescriptor)
{
    Connection *connection = new Connection(socketDescriptor, this);
    emit newConnection(connection);
}
```

Şekil 15 – Server Implementasyonu

Şekil 15'te görüleceği üzere Server oluşturulduğu andan itibaren gelecek bağlantıları dinlemeye başlamaktadır. Gelen bağlantı olmasında durumunda bağlantı objesi oluşturulur.

```
static const int TransferTimeout = 30 * 1000;
static const int PongTimeout = 60 * 1000;
static const int PingInterval = 5 * 1000;

/*
 * Protocol is defined as follows, using the CBOR Data Definition Language:
 *
 *  protocol    = [
 *     greeting,        ; must start with a greeting command
 *     * command        ; zero or more regular commands after
 *  ]
 *  command     = plaintext / ping / pong / greeting
 *  plaintext   = { 0 => text }
 *  ping        = { 1 => null }
 *  pong        = { 2 => null }
 *  greeting    = { 3 => text }
 */

Connection::Connection(QObject *parent)
    : QTcpSocket(parent), writer(this) {...}

Connection::Connection(qintptr socketDescriptor, QObject *parent)
    : Connection(parent) {...}
```

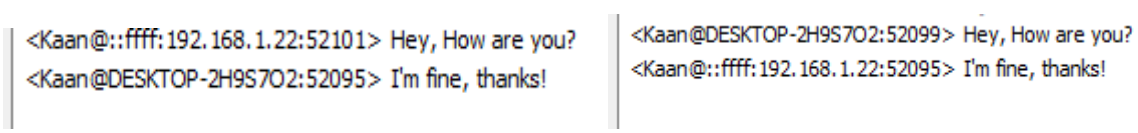
Şekil 16 – İletişim protokolü connection objesinde tanımlanmıştır

Şekil 16'da görüldüğü üzere tüm mesajlaşma tabanlı komunikasyonlar CBOR Veri Tanımlama dilinde tanımlandığı gibi yapılmaktadır. Her istekte bir greeting komutu ve sıfır veya daha fazla ek komut içermektedir. Gelen veri parse edilerek durumuna uygun işlem yapılır.

```
void PeerManager::sendBroadcastDatagram()
{
    QByteArray datagram;
    {
        QCborStreamWriter writer(&datagram);
        writer.startArray(2);
        writer.append(username);
        writer.append(serverPort);
        writer.endArray();
    }

    bool validBroadcastAddresses = true;
    for (const QHostAddress &address : qAsConst(broadcastAddresses)) {
        if (broadcastSocket.writeDatagram(datagram, address,
                                          broadcastPort) == -1)
            validBroadcastAddresses = false;
    }

    if (!validBroadcastAddresses)
        updateAddresses();
}

void PeerManager::readBroadcastDatagram()
{
    while (broadcastSocket.hasPendingDatagrams()) { ...}
}
```

Şekil 17 -Peer Manager Operasyonları

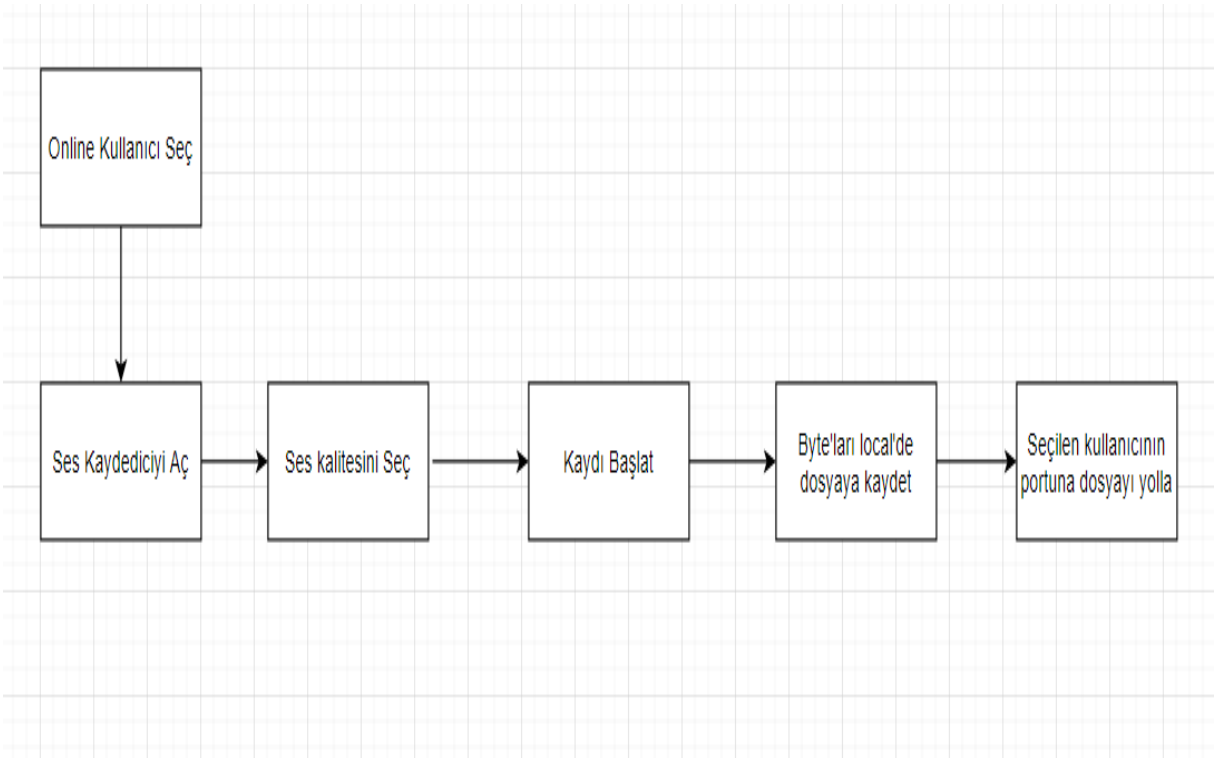
Şekil 17'de Peer Manager'ın yaptığı en önemli iki iş gözükmektedir. Gelen ve giden datagramları okur ve broadcasting işlemini yapar.



Şekil 18 – 52101 ve 52095 portunda iki kişinin mesajlaşması

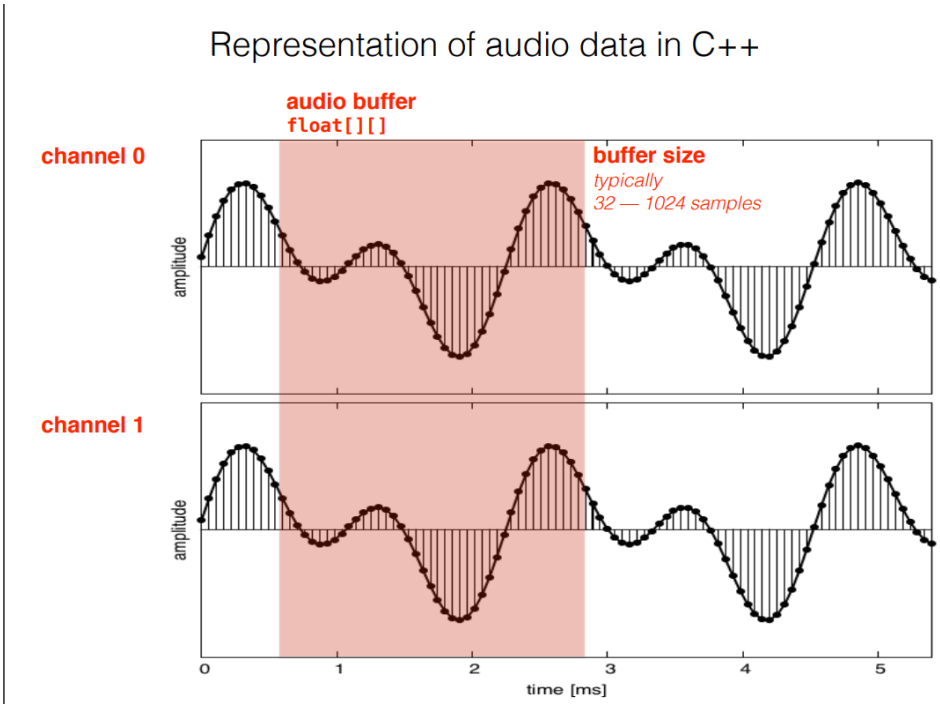
IV. Sesli Mesajlaşma

CCIDE, kullanıcılara sesli mesaj ile iletişim kurma imkanı sunmaktadır. Bu özelliğin gerçekleştrilirmesi için çeşitli yöntemler kullanılabilir. Bilgisayara bağlı mikrofona bağlanıp gelen ses verilerini sırayla çok kısa bir gecikme ile User Datagram Protocol (UDP) kullanarak karşı kullanıcının portuna server üzerinden yollanıp, karşı kullanıcıdan gelen byte'ları sırayla oynatması istenebilir [4]. Literatürde standart bu şekilde olmasına rağmen mevcut server'ın bu özelliği efektif kullanacağı özelliklerde olmaması, kullanıcıların asıl önceliğinin anlık sesli iletişimde olmaması ve anlık özelliğinin çok maliyetli olması sebebiyle farklı bir yol izlenmiştir. CCIDE sesli mesajlaşma özelliğini ses verilerini bir dosyaya kaydedip bu dosyayı TCP ile server üzerinden karşı kullanıcıya yollamaktadır. Bu aradaki gecikmeyi biraz arttırsa da en düşük maliyetle en kaliteli düzeyde sesli iletişimi sağlamaktadır.



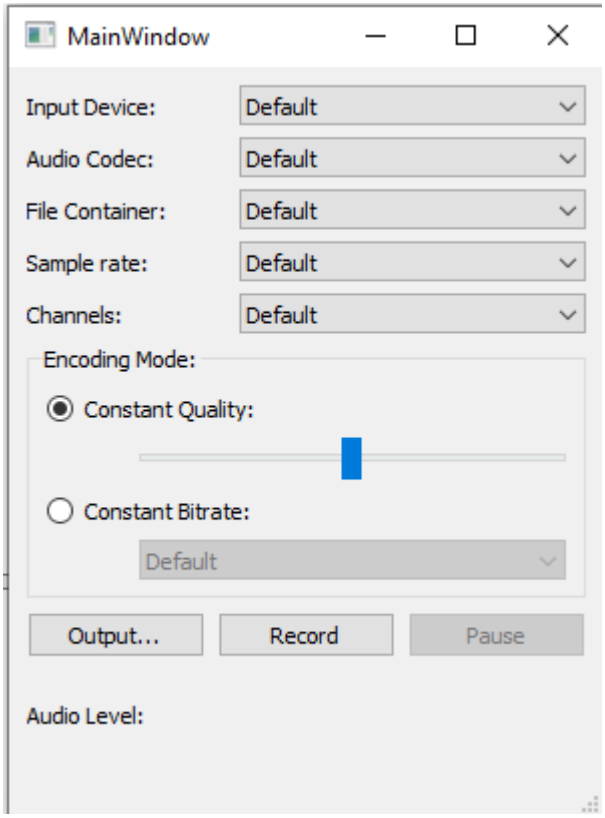
- **Ses Kaydı**

  Gündelik hayatta duyduğumuz tüm sesler birer analog sinyaldir ve bunları bilgisayar üzerinde gösterebilmek ve üzerinde işlemler yapmak için bu sinyaller dijital sinyallere çevrilmek zorundadır.



Şekil 19 – Analog sinyalin C++ dilinde gösterimi[5]

Şekil 19'da görüleceği üzere analog sinyal olan sesleri algılayan bir mikrofondan sinüzoid dalgadan belirli aralıklarla örnekler alınmaktadır. Bu örnekler her geldiklerinde direkt olarak işleme sokulmazlar, bir buffer içine konurlar. Aksi halde işlemci sayının çokluğundan dolayı işlem yapamayacaktır.



Şekil 20 – Recorder Görünümü

Sesini kaydetmek isteyen kullanıcı mikrofon cihazını, örnekleme değerini seçmekte sonrasında Record tuşuna basarak kaydetme işlemini aktif hale getirmektedir.

```
m_audioRecorder->setEncodingSettings(settings, QVideoEncoderSettings(), container);
m_audioRecorder->record();
```

Şekil 21 – Ayarların Kaydedilip Kayıt İşleminin Başlaması

```
// This function returns the maximum possible sample value for a given audio format
qreal getPeakValue(const QAudioFormat& format) {...}

// returns the audio level for each channel
QVector<qreal> getBufferLevels(const QAudioBuffer& buffer) {...}

template <class T>
QVector<qreal> getBufferLevels(const T *buffer, int frames, int channels) {...}

void AudioRecorder::processBuffer(const QAudioBuffer& buffer) {...}
```

Şekil 22 - Alınan Ses Verilerinin İşlenmesi

Seçilen ses formatına göre en yüksek örnekleme değeri alınarak kaydedilen tüm kanallarda ses verisi dosyaya kaydedilir.

- **Kaydedilen Dosyanın Transferi**

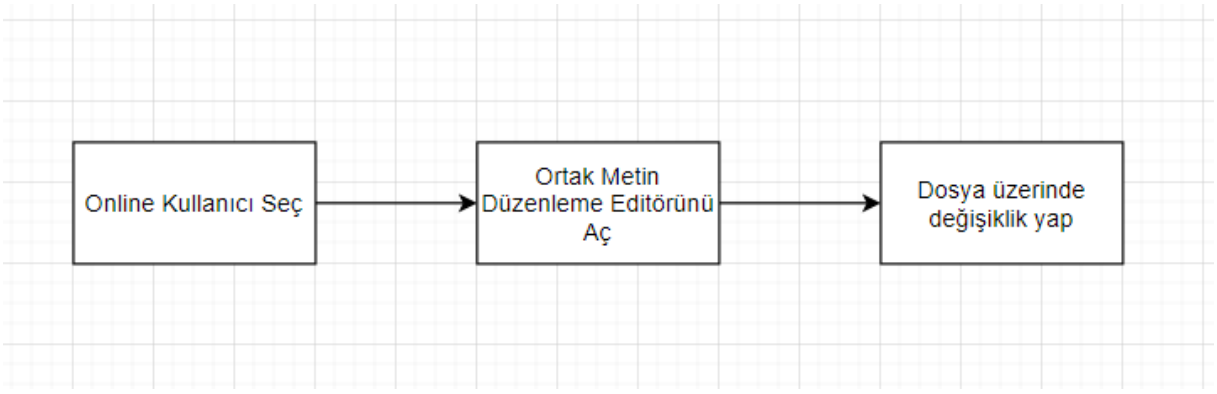
```
void SendThread::run()
{
    QTcpSocket client;
    if (!client.setSocketDescriptor(socketDescriptor))
    {
        qDebug() << client.error();
        return;
    }
    qDebug() << "Thread : Connected";

    //send File
    QFile inputFile(FILENAME);
    QByteArray read;
    inputFile.open(QIODevice::ReadOnly);
    while(1) {...}
    inputFile.close();
    client.disconnectFromHost();
    client.waitForDisconnected();
    qDebug() << "Thread : File transfer completed";
}
```

Şekil 23 – TCP Soketi ile Dosya Transeri

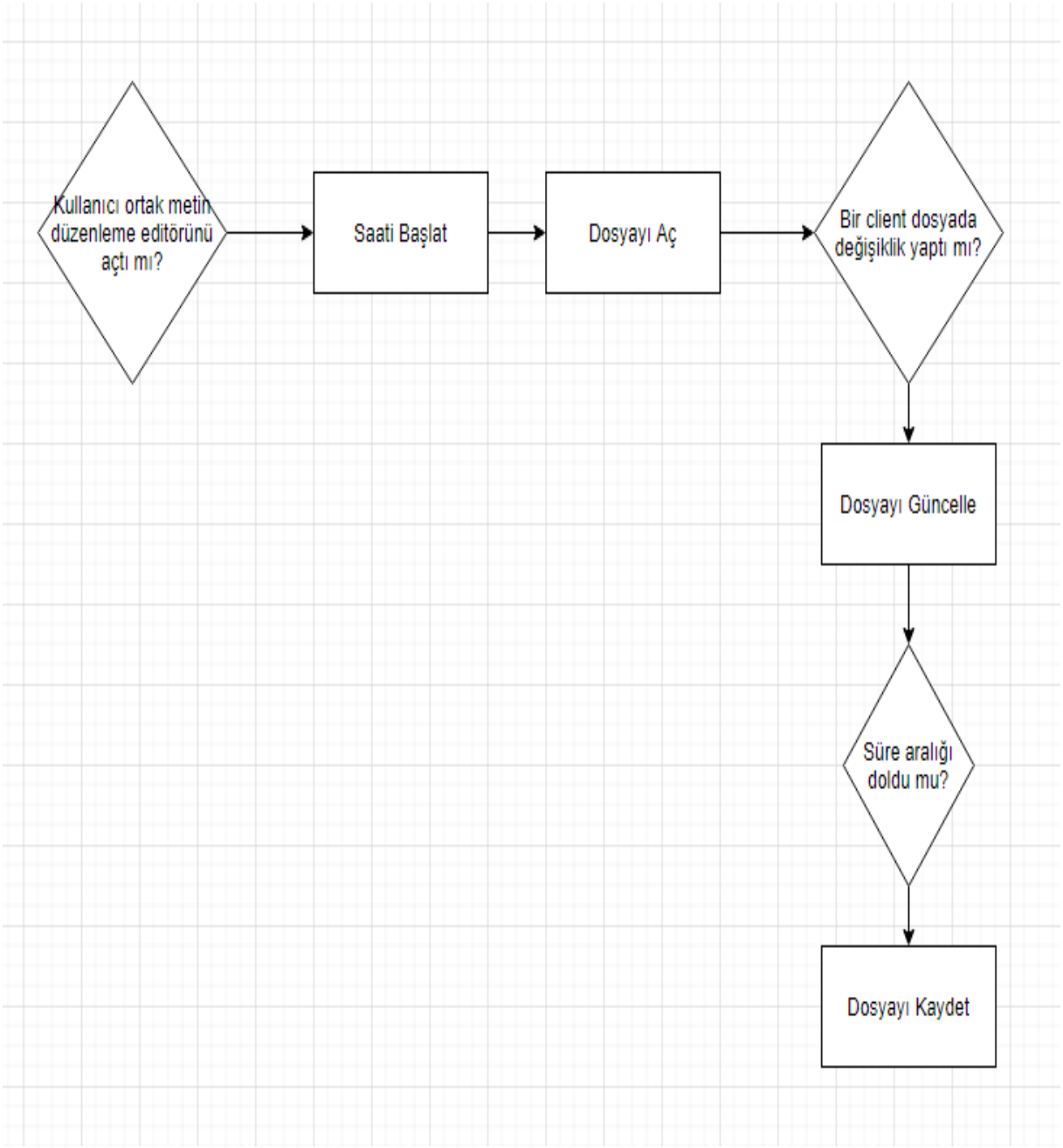
Ses dosyasının kaydedilmesinin ardından yeni bir thread oluşturulur ve gönderilmesi istenen kullanıcının Socket Descriptor'u gösterilerek ses dosyasının tüm byte'ları sırayla yollanır.

V. Ortak Metin Düzenleme

CCIDE küçük boyutlu yazılım geliştirme takımlarının daha efektif bir şekilde kod kontrolü (code review) yapmalarını sağlamak için ortak metin düzenleme (collaborative text edition) özelliğini kulanıcılara sunmaktadır.[6] Ortak metin düzenleme literatürde öncelikle Operational Transformation [7] denilen bir sistem kullanılarak ya da daha yeni ve daha efektif bir yöntem olan CRDT (Conflict-free Replicated Data Type) [8] [9] denilen bir veri tipi kullanılarak server üzerinde atomik okuma-yazma yapılarak yapılmaktadır. Server'ın bulunduğu sistem atomik işlemleri desteklemediğinden server tabanlı başka bir yol izlenmiştir. Bu yöntemde server ortak metin düzenlemesi kullancağı dosya için bir saat tutar. Client'ların yaptığı değişikler server'a yollanır. Server tüm bu değişiklikleri oluşturduğu saatin belirli bir aralığına göre kendi içindeki dosyayı değiştirir. Bu değişiklik yapılan dosyanın en güncel hali zaman aralığı sonunda tüm client'lar ile paylaşılır. Dosya üzerinde veri çakışmalarını engellemek adına client'lar ortak metin düzenleme anında lokallerinde dosyayı değiştirmezler. Tüm işlemler server nezdinde halledilir. Ortak metin düzenleme bittiği zaman server client'taki dosyayı değiştirir.



Şekil 24 – Client'ın Ortak Metin Düzenlemesi



Şekil 25 – Server'ın ortak metin düzenleme için yaptığı işlemler

```
CollabEditor::CollabEditor(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::CollabEditor)
{
    ui->setupUi(this);

    setupEditor();
    //Timer
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(updateDocument()));
    timer->start(60000);

}
```

Şekil 26 – Ortak Metin Düzenleme Editörü / Server

Şekil 26'da görüleceği gibi bir adet sayaç oluşturulmuştur ve her zaman aralığının dolmasıyla updateDocument metodu çağırılarak server içindeki kullanıcıların ortak inceledikleri güncellenir ve kullanıcılar bu güncellenmiş versiyonu görürler.

```
void CollabEditor::on_collabTextWidget_textChanged()
{
    if(currentFile == "")
        return;
    QFile file(currentFile);
    if(!file.open(QIODevice::WriteOnly | QFile::Text)){ ...}
    QTextStream out(&file);
    QString text = ui->collabTextWidget->toPlainText();
    out << text;
    file.close();
}
```

Şekil 27 – Local'de Yapılan Değişikler

Şekil 27'de görülen metodda ise kullanıcıların her değişikliği kayıt altına alınıp bufferlanarak sisteme gönderilir.

VI. Versiyon Kontrolü

CCIDE versiyon kontrolü için aynı derleyicide olduğu gibi 3. Parti bir yazılıma ihtiyaç duymaktadır. Kullanıcıların bilgisayarında Git programının kurulu olması ve hesaplarına giriş yapmış olması gerekmektedir. Kullanıcıların yaptığı tüm değişikliler 2 butonla öncelikle commit edilip sonrasında GitHub hesaplarındaki repositorylerine "push"lanır. [10]

VII. Diğer Özellikler

CCIDE ana özelliklerinin yanı sıra syntax highlightning denilen C++ kodlarının renklendirilmesi ve bazı kod fikirlerinin çalışıp çalışmadığını çözmek için Python Interpreter'a sahiptir. Kod renklendirmesi Regular Expressionlar'la sağlanmıştır. Python kodlarının çalışması ise C++ kodları ile aynı şekildedir. Yapılan bir sistem çağrısı ile Python kodu çalıştırılır, çıktısı program içinde yazdırılır.

```
    {
QT_MOC_LITERAL(0, 0, 6), // "Server"
QT_MOC_LITERAL(1, 7, 13), // "newConnection"
QT_MOC_LITERAL(2, 21, 0), // ""
QT_MOC_LITERAL(3, 22, 11), // "Connection*"
QT_MOC_LITERAL(4, 34, 10) // "connection"

    },
    "Server\0newConnection\0\0Connection*\0"
    "connection"
};
#undef QT_MOC_LITERAL

static const uint qt_meta_data_Server[] = {

 // content:
       8,       // revision
       0,       // classname
       0,    0, // classinfo
       1,   14, // methods
       0,    0, // properties
```

Şekil 28 – Regular Expression İle Tanınan Kelimelerin renklendirilmesi

- **ARAŞTIRMA PROJESİNİN SONUCU VE KATKISI**

Bu proje sonucunda yazılım geliştirme süreçlerinde 3. Parti yazılımların maliyeti ne kadar arttırabileceği, bir Integrated Development Environment'ın nasıl olması gerektiği, işletim sistemi düzeyinde sistem çağrıları, ortak metin düzenleme temelleri ve network temellerinin ortaklaşa (aynı hafıza alanı üzerinde) nasıl kullanılması gerektiği öğrenilmiştir. Her biri ayrı bir program/ürün olabilecek büyüklükte olan özellikler CCIDE içinde bir araya gelmiş ve işletme düzeyinde kaliteden ödün vermeden düşük maliyetlerle kısa sürede güçlü bir ürün ortaya çıkarabilmekte ve şirketleri yazılım mühendisi başına aylık yüzlerce dolar maliyetten kurtarabilmektedir.

- **KAYNAKLAR**

[1] https://en.wikipedia.org/wiki/Integrated_development_environment

[2] https://www.computerwoche.de/a/interaktives-programmieren-als-systems-schlager,1205421

[3] https://www.qt.io/

Collaborative Real-Time Editor, https://en.wikipedia.org/wiki/Collaborative_real-time_editor

[4] https://en.wikipedia.org/wiki/Voice_over_IP

[5] https://timur.audio

[6] Kleppmann M., "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable and Maintainable Systems" 6th Edition, Chapter 5, Page 165

[7] Sun C., Ellis C. "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements", ACM Conference on Computer Supported Cooperative Work(CSCW), 1998

[8] Conflict-free Replicated Data Type, https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type

[9] Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski. A comprehensive study of Convergent and Cummutative Replicated Data Types. [Research Report] RR-7506, Inria – Centre ParisRocquencourt; INRIA. 2011, pp.50. ffinria-00555588f

[10] https://github.com/about