Implementacja aplikacji sieciowych z wykorzystaniem środowiska Qt

1. Wprowadzenie

Wymagania wstępne: wykonanie ćwiczeń „Adresacja IP" oraz „Implementacja aplikacji sieciowych z wykorzystaniem interfejsu gniazd BSD".

Środowisko Qt®, którego producentem jest firma Trolltech® (www.trolltech.com), jest zestawem narzędzi i klas do tworzenia graficznych aplikacji w języku C++. Oprogramowanie to jest dostępne dla wielu systemów operacyjnych, m.in. dla systemów Windows, Mac oraz Unix/Linux. Qt dla systemów Unix/Linux jest bezpłatne, jeśli jest używane do pisania darmowego oprogramowania z otwrtym dostępem do kodu źródłowego; kod źródlowy samego środowiska jest dostępny nieodpłatnie na warunkach licencji publicznej QPL (*Qt Public License*).

Biblioteka Qt jest w pełni obietkowa i pozwla na przenoszenie kodu na różne platformy (wymagana jest rekompilacja). Zawiera ona bogaty zestaw klas, które pozwalają m.in. na obsługę baz danych, sieci i protokołów komputerowych, grafiki 2D i 3D oraz XML. Graficzny interfejs tworzonej aplikacji może być zaprojektowny przy użyciu wizualnego narzędzia o nazwie *Qt Designer*.

Designer jest programem, który pozwala na wizualne tworzenie interfejsu aplikacji oraz na implementację funkcjonalności wszystkich jego elementów. Do dyspozycji programista ma zbiory kontrolek (ang. *widget*), gotowe przykłady okien dialogowych i aplikacyjnych oraz odpowiednie kreatory. Oprócz programu *Designer* istnieją także inne programy do konstruowania interfejsów graficznych aplikacji Qt, są to m.in. *Qt Architect* (qtarch.sourceforge.net) oraz *KDE Studio* (www.thekompany.com/projects/kdestudio).

1.1 Sygnały i odbiorniki

Środowisko Qt wykorzystuje mechanizm komunikacji pomiędzy obiektami o nazwie **sygnały i odbiorniki** (ang. *signals and slots*) – jest on najczęściej używany do połączenia zdarzeń (wywołanych np. przez użytkownika lub przybycie komunikatu) z kodem aplikacji odpowiadającym za obsługę tych zdarzeń. Każdy obiekt w środowisku Qt może emitować i wysłuchiwać dowolną ilość sygnałów; każdy sygnał może mieć dowolną liczbę odbiorników, a każdy odbiornik może reagować na dowolną ilość sygnałów. Połączenia mogą być realizowane pomiędzy sygnałem a odbiornikiem, jeśli ich parametry są tego samego typu; odbiornik może posiadać mniej parametrów niż sygnał (ale nie odwrotnie), wówczas nadmiarowe argumenty sygnału są pomijane.

Aby obiekt mógł korzystać z mechanizmu sygnałów-odbiorników, jego klasa musi dziedziczy z klasy *QObject* (lub z klasy potomnej *QObject*), a w jej definicji należy umieścić makro `Q_OBJECT` (kod z taką definicją jest przetwarzany przez preprocesor, który generuje kod wynikowy, obsługiwany przez kompilator C++). Odbiorniki (ang. *slots*) mogą być deklarowane jako: publiczne (`public slots`), chronione (`protected slots`) i prywatne (`private slots`). Przykład nr 1 prezentuje klasę, w ramach której utworzono odbiornik, sygnał oraz połączenie pomiędzy sygnałem i odbiornikiem dwóch różnych obiektów.

```
class myExample : public QObject
{
   Q_OBJECT                  //makro

public:
   void myFunction();
public slots:
   void mySlot(int iFoo);    //deklaracja odbiornika
signals:
   void mySignal(int iWoo); //deklaracja sygnału
private:
   int iEoo = 1;
}

void myExampl::myFunction()
{
   emit mySignal(iEoo);      //emisja sygnału mySignal
}

int main()
{
   myExample a,b;

   //połączenie sygnału i odbiornika dwóch różnych obiektów
   connect(&a, SIGNAL(mySignal(int)), &b, SLOT(MySlot(int)));
...
}
```

Przykład nr 1. Klasa zawierająca deklarację odbiornika i sygnału oraz połączenie pomiędzy sygnałem i odbiornikiem dwóch różnych obiektów.

Metoda sygnałów-odbiorników może być używana zarówno do obsługi zdarzeń generowanych prze graficzny interfejs użytkownika, jak i zdarzeń związanych z obsługą komunikatów sieciowych. Przykład nr 2 ilustruje wykorzystanie sygnału obiektu klasy *QDns*, o uzyskaniu numeru IP komputera o podanej nazwie domenowej.

```
//utworzenie nowego obiektu i podanie nazwy domenowej do rozwiązania
m_MyDns = new Qdns(lineEditAddress->text());

//połączenie sygnału uzyskania adresu IP z funkcja obsługi
QObject::connect(m_MyDns, SIGNAL(resultsReady()), m_MyObject, SLOT
(ConnectToServer()));

//funkcja obsługi sygnału z klasy obiektu m_MyObject
MyClass::ConnectToServer() { ... }
```

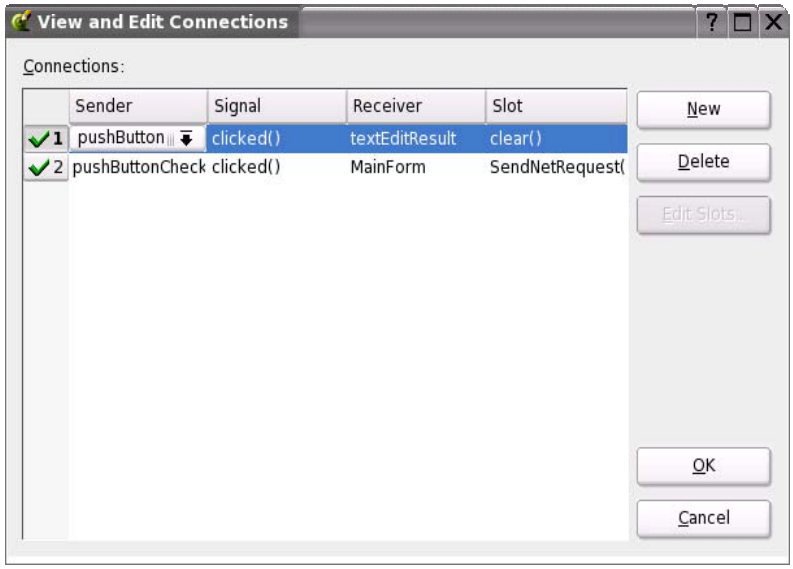
Przykład nr 2. Wykorzystanie sygnału klasy QDns do obsługi zdarzenia uzyskania adresu IP.

1.2 Tworzenie aplikacji przy pomocy *QDesigner*

Utworzenie graficznego interfejsu aplikacji przy pomocy narzędzia QDesigner, wymaga następujących kroków:

- utworzenie nowego projektu – **File | New | C++ Project...** – powoduje to m.in. utworzenie pliku z rozszerzeniem .pro, który jest używany przez narzędzie *qmake*, do generowania pliku *Makefile*;
- utworzenie nowego okna dialogowego – np. **File | New | Dialog...**;

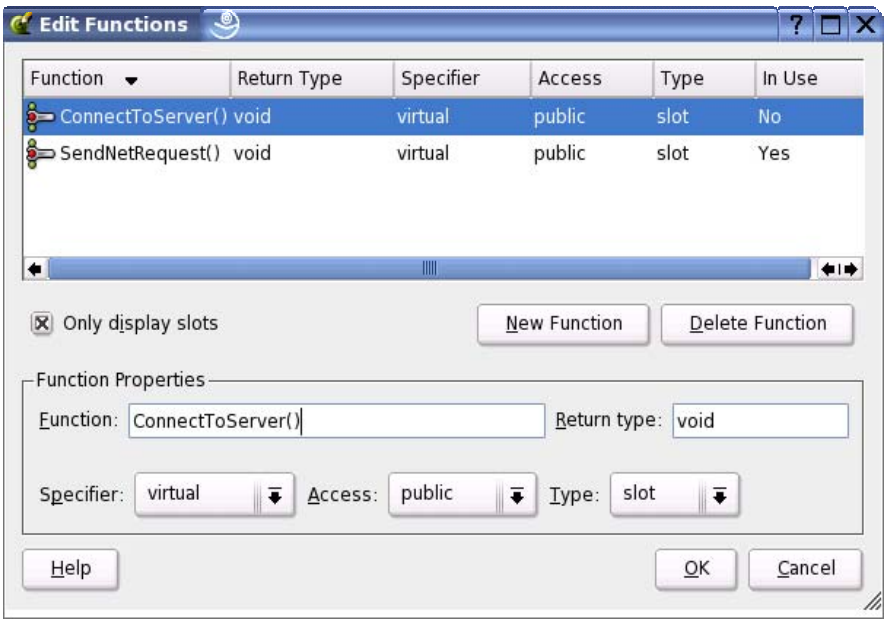
- dodanie do utworzonego okna kontrolek (ang. *widgets*) niezbędnych dla funkcjonalności aplikacji, są one dostępne w okienku **Toolbox**; możliwe jest uruchamianie podglądu wyglądu tworzonej aplikacji przy pomocy komendy: **Preview | Preview Form...**;
- dodanie połączeń sygnał-odbiornik do obsługi zdarzeń związanych z interfejsem graficznym i jego kontrolkami; w aplikacji *QDdesigner* służy do tego specjalne okno uruchamiane poleceniem: **Edit | Connections** – rysunek nr 1;



Rysunek nr 1. Okno umożliwiające edycję połączeń sygnał-odbiornik.

- być może konieczne będzie dodanie nowego odbiornika (ang. *slot*), do obsługi zdarzeń – można to wykonać wybierając polecenie **Edit Slots...** w oknie dialogowym obsługi połączeń (rysunek nr 1); wyświetlone zostanie wówczas kolejne okno do automatycznego dodawania funkcji i odbiorników – rysunek nr 2;
- dodanie kodu obsługi zdarzeń i funkcji aplikacji – implementacji należy dokonać w pliku uruchamianym z okna **Project Overview** (nazwa pliku jest następująca: *nazwa_okna_dialogowego.ui.h*);
- ostatecznie należy utworzyć plik z główną funkcją (ang. *main)* aplikacji – służy do tego kreator uruchamiany poleceniami: **File | New | C++ Main File...**; utworzony plik nie musi podlegać żadnym zmianą;
- po zapisani wszystkich zmian, należy skompilować cały projekt; w tym celu wykorzystuje się narzędzie *qmake*, które generuje odpowiedni plik *Makefile* (na podstawie pliku projektu .pro) dla programu *make*; kompilacja wymaga więc następujących komend:

```
qmake -o Makefile ./projekt.pro
make
```



Rysunek nr 2. Okno edycji funkcji i odbiorników.

1.3 Klasy biblioteki Qt do obsługi sieci i protokołów komputerowych

Bogaty zestaw klas dostarczany w ramach środowiska Qt zawiera także klasy, które ułatwiają programistom tworzenie aplikacji sieciowych. Wśród tych klas znajdują się rozwiązania upraszczające korzystanie z gniazd komunikacji sieciowej i systemu DNS oraz do sprawnej implementacji protokołów sieciowych. Poniżej znajduje się lista najistotniejszych klas związanych z sieciami komputerowymi:

- *QSocket*: klasa umożliwia realizację buforowanych połączeń TCP;
- *QSocketDevice*: klasa – niezależnej od platformy – obsługi interfejsu gniazd; pozwala na realizację połączeń TCP i UDP oraz obsługuje protokoły IP w wersjach 4 i 6;
- *QServerSocket*: klasa obsługi nadchodzących połączeń TCP, przeznaczona do zastosowania przy implementacji serwerów;
- *QSocketNotifier*: klasa pomocnicza do obsługi gniazd; pozwala na asynchroniczną (nieblokującą) obsługę gniazd, poprzez udostępnianie sygnału informującego o zdarzeniu związanym ze wskazanym gniazdem;
- *QNetworkProtocol*: klasa ułatwiająca implementację protokołów sieciowych; klasa ta dostarcza zbiór metod adekwatnych do podstawowych operacji protokołów – do metod tych należy dodać funkcjonalność właściwą dla implementowanego protokołu;
- *QNetworkOperation*: klasa opisująca stan operacji protokołu sieciowego;
- *QDns*: klasa asynchronicznych zapytań DNS (rozwiązywanie nazw domenowych w adresy IP).

Oprócz wymienionych klas przydatne w implementacjach sieciowych mogą być także m.in. klasy: *QFtp*, *QHttp*, *QHttpHeader, QHttpRequestHeader*, *QHttpResponseHeader* oraz *QUrl*, *QUrlDrag*, *QUrlInfo*, *QurlOperator*.

2 Organizacja, wymagany sprzęt, oprogramowanie

1. Zadanie wykonywane jest pojedynczo przez wszystkich studentów (lub w parach);
2. sprzęt: 2 komputery;
3. oprogramowanie: biblioteka Qt, *Qt Designer*, narzędzia *qmake* i *make* oraz kompilator g*cc*.

3 Zadania

1. Na jednym z komputerów należy uruchomić serwer *daytime*; zadanie polega na napisaniu graficznego programu sieciowego – klienta – przy użyciu biblioteki Qt (i narzędzia *Qt Designer*), który uruchomiony na drugim komputerze odczyta informacje z uruchomionego serwera. Należy użyć trybu połączeniowego, korzystając z klasy *QSocketDevice*.
2. Wzbogacić powyższy program o obsługę nazw domenowych przy pomocy klasy *QDns*.

4 Pytania sprawdzające

1. Dlaczego do implementacji graficznych aplikacji sieciowych zaleca się używania połączeń nieblokujących?
2. W jakim kontekście zostanie wykonany kod odbiornika, jeśli sygnał został nadany przez wątek?
3. Czy możliwe jest zastąpienie funkcji systemowej **`select()`**, w aplikacji sieciowej, rozwiązaniem środowiska Qt bez utraty funkcjonalności? Jeśli tak, to jakim i w jaki sposób?

5 Literatura

1. Internet: www.trolltech.com/qt , dokumentacja http://doc.trolltech.com/3.3/index.html , whitepapers http://www.trolltech.com/products/whitepapers.html .
2. Neil M., Stones R.,: *Zaawansowane programowanie w systemie Linux*, Helion, 2000.