Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Grafikus leírónyelvek közös részhalmazának gépi feldolgozása

Önálló laboratórium beszámoló

Készítette
Főglein Simon István

Konzulens
dr. Somogyi Péter

2023. június 4.

Tartalomjegyzék

Bevezető

A dolgozat a Qt és GTK felhasználói felület keretrendszerek felületleíró nyelvei közötti átjárhatóságot mutatja be. Mindkét technológia széleskörűen elterjedt mind a FOSS, mind a kereskedelmi szoftverek körében. A FOSS projektekre jellemző forkolást, továbbfejlesztést segítené a két technológia közötti átjárhatóság. A Qt licencelése jelentősen függ a *The Qt Company*-tól, így ha ők a kizárólagos kereskedelmi licenc mellett döntenek [4], sok projekt bajba kerülhet. A GTK viszont egy tervezése óta szabad szoftverként licencelt GUI keretrendszer, mely alkalmas lehet a Qt helyettesítésére. A Qt, mint keretrendszer pedig alkalmasabb lehet egy komplex projekt megvalósítására, ugyanis az általa biztosított könyvtárak számtalan magasabb absztrakciós szintű osztályt tartalmaznak, ezzel is könnyítve a fejlesztés menetét. Egy fordítóprogram, mely elősegíti a két kezelőfelület-keretrendszer közötti átjárást nagyban segítheti egy projekt más technológiára való átalakítását.

A jelenleg elérhető megoldások nagyon kezdetlegesek, lényegében csak XML → XML és XML → JSON átalakítást tesznek lehetővé, nincs hatékony eljárás a két technológia közötti átjárásra.

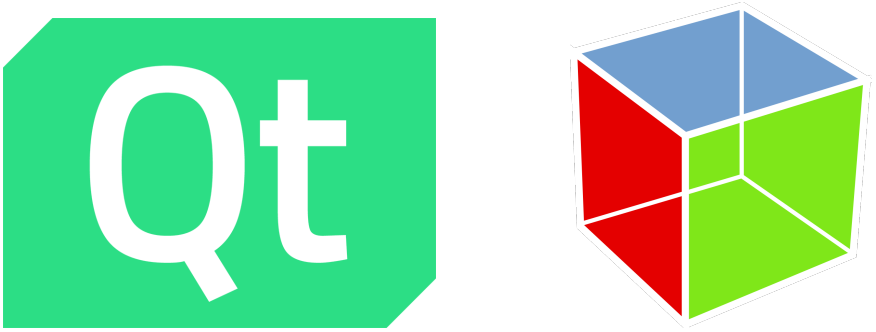
1. fejezet

A Qt és GTK keretrendszerek áttekintése

1.1. Bemutatás, rövid történet

A GTK és a Qt (ejtése mint az angol *cute* szó) széles körben elterjedt GUI eszközkészlet-keretrendszerek. Mindkettővel lehetőség van összetett felhasználói felületek készítésére, a Qt által biztosított osztályok ezen kívül lehetőséget adnak komplex alkalmazások létrehozására is.

A Qt és a GTK története is az 1990-es évekre vezethető vissza. A Qt fejlesztését 1990 nyarán kezdte meg Haavard Nord és Eirik Chambe-Eng, amikor egy ultrahangfelvételek tárolására alkalmas programot fejlesztettek [10]. Később céget alapítottak, 1994-ben megalakult a Quasar Technologies, ami később Trolltech-ként vált ismertté, manapság pedig a The Qt Company nevet viseli. A keretrendszer köré szerveződött cég jól mutatta, hogy a Qt alkotói pénzt szerettek volna keresni a könyvtárral, így a licence nem engedte a szabad terjesztést. Ez először akkor kezdett problémává válni, amikor a KDE – egy népszerű Linux asztali környezet – bebiztosította a helyét a túlnyomórészt szabad szoftverekből álló Linuxos világban.



1.1. ábra. *A Qt és a GTK logója*

A GTK történetének kezdete körülbelül 1996-ra tehető, ekkor kezdte meg ugyanis Peter Mattis a keretrendszer fejlesztését [8]. A cél a GIMP-hez akkor használt Motif GUI eszközkészlet lecserélése volt (a könyvtár eredeti neve, a GIMP ToolKit is innen ered), amit

végül az 1998 nyarán megjelent 1.0-s GIMP verzióval sikerült is véghez vinni.

Mindkét technológia elterjedésében jelentős szerepe volt annak, hogy a '90-es évek végén nagyobb asztali környezetek kezdték el használni mind a Qt, mind a GTK könyvtárakat. A KDE-projekt a Qt egyik legjelentősebb felhasználója és számos változtatással segítik a keretrendszer fejlődését, míg a GNOME asztali környezet fejlesztése a GTK alakulására van nagy hatással. Fontos különbség azonban a fent már említett licencelés problémája: a Qt egy kereskedelmi forgalomban lévő szoftvercsomag, mely néhány kisebb kivétellel (pl. nyílt forráskódú projektek [1], oktatási célok [3]) csak licencdíj megfizetése ellenében használható. Ez sokaknak nem tetszett a szabad szoftverekben bővelkedő Unixos világban, így ez is motiválta a GTK korai fázisában a fejlesztést, ugyanis a GTK teljesen szabad licenccel rendelkezik, bárki szabadon felhasználhatja, módosíthatja és terjesztheti is.

1.2. Technológiai áttekintés

A Qt elsődleges programozási nyelve a C++, míg a GTK-é a C, bár mindkettőhöz léteznek megoldások más nyelvekkel való együttműködés biztosítására is, mint például Python és C++.

A felhasználói felületek leírásához alapvetően mindkét technológia XML-alapú megoldást használ, bár a Qt esetében lehetőség van a JSON-alapú QML használatára is. A QML (Qt Modeling Language) egy deklaratív felhasználóifelület-leíró nyelv, melyet a Nokia fejlesztett a Qt projekthez 2009 környékén [9]. Előnye az XML-alapú megoldáshoz képest, hogy könnyebben áttekinthető, valamint lehetőséget biztosít JavaScript használatára is, így például a KDE számos alkalmazását folyamatosan portolják – azaz a funkcionalitás megtartása mellett, általában a felhasználók számára láthatatlan módon megváltoztatják a használt könyvtárakat és a kódbázist – át a hagyományos C++ és XML technológia helyett a QML-esre (ugyanakkor a QML részei is elérhetőek C++ kódból, például lehetőség van eseménykezelők regisztrálására is).

A felületleírókban megadott kinézet előállításához a GTK a GTK Builder[1] osztályt használja, míg a Qt a `uic` segédprogram segítségével generál osztályokat a `.ui` fájlokból [2]. Az így generált osztályok a `Ui` névtéren belül érhetőek el. A felület kódbeli inicializálását az alábbi táblázat mutatja be:

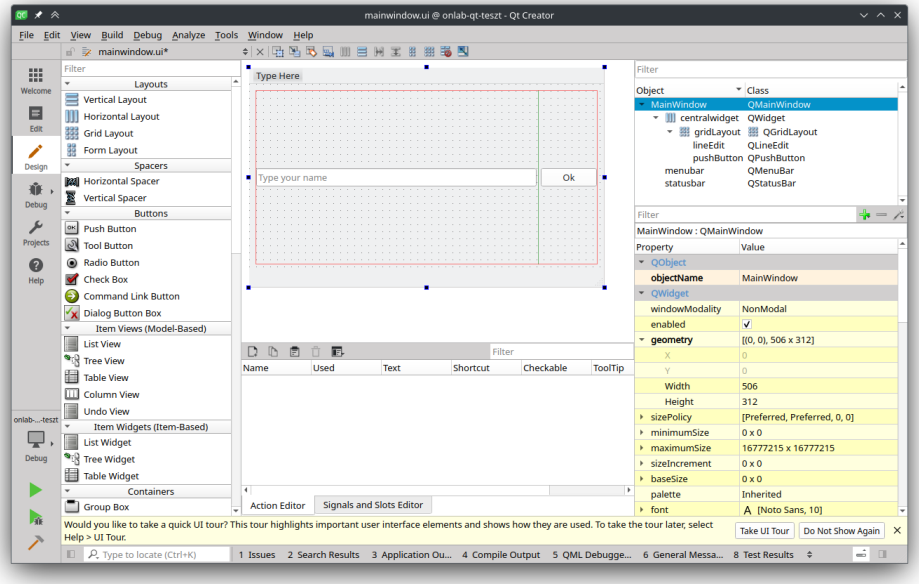
Qt	GTK
`ui(new Ui::Notepad);` `ui->setupUi(this);`	`Gtk::Builder::` `  new_from_file("ui.glade")`

1.1. táblázat. *XML-ben definiált felhasználói felület inicializálása Qt-t és GTK-t használó programokban*

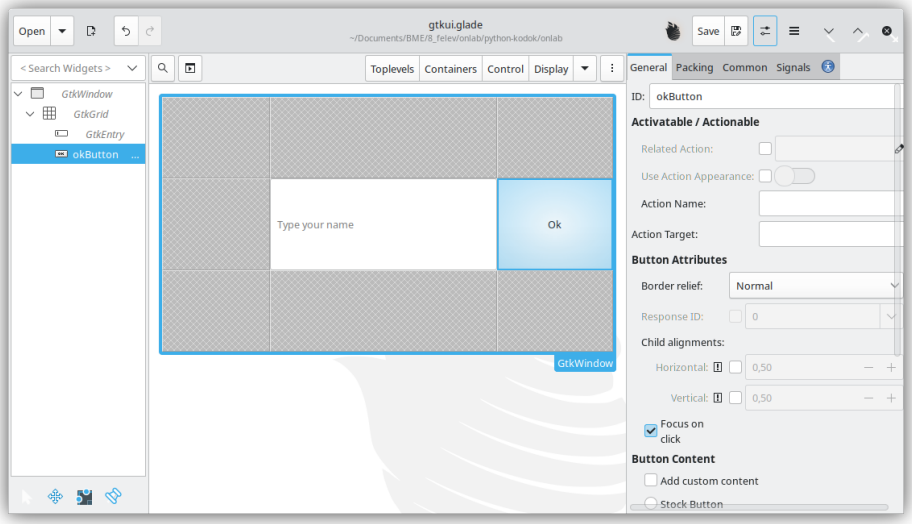
Mivel mindkét szóban forgó keretrendszer lehetőséget biztosít összetett felhasználói felületek létrehozására, és ezek leírása kézzel nagyon körülményes lenne, ezért lehetőség van a felhasználói felületek vizuális összeállítására. Mind a Qt, mind a GTK rendelkezik ilyen programmal: Qt esetében ilyen például a Qt Creator, míg GTK-nál a Glade a legnépszerűbb GUI összeállító program. Bár a *Design* nézet mindkét szerkesztő esetén ugyanazt a célt

[1] `https://docs.gtk.org/gtk3/class.Builder.html`

szolgálja, a két alkalmazás jelentősen eltér egymástól: a Qt Creator egy fejlesztői környezet, mely rendelkezik *Design* nézettel (de emellett lehetőség van pl. C++ forráskód írására, valamint a projekt buildelésére is), a Glade viszont kizárólag a felület összeállításában játszik szerepet. A későbbiekben az ilyen programokkal létrehozott UI-leírók feldolgozására összpontosítunk.



1.2. ábra. *A Qt Creator felhasználói felülete design nézetben*



1.3. ábra. *A Glade felhasználói felülete*

QML

```
QWidget {
    name: "centralWidget"
    QPushButton {
        name: "pushButton"
        geometry: {
            x: 10
            y: 20
            width: 150
            height: 50
        }
        text: "Hello World!"
    }
}
```

Qt XML

```
<widget class="QWidget"
            name="centralWidget">
  <widget class="QPushButton"
          name="pushButton">
    <property name="geometry">
      <rect>
        <x>10</x>
        <y>20</y>
        <width>150</width>
        <height>50</height>
      </rect>
    </property>
    <property name="text">
      <string>Hello World!</string>
    </property>
  </widget>
</widget>
```

GTK XML

```
<object class="GtkFrame">
  <child>
    <object class="GtkButton"
        id="pushButton">
      <property name="label"
          translatable="yes">
          Hello World!
        </property>
        <property name="name">
          pushButton
        </property>
    </object>
  </child>
</object>
```

1.2. táblázat. *A QML, Qt XML és a GTK XML felépítésének összehasonlítása*

2. fejezet

Fordítóprogramok

2.1. Fordítóprogramok bemutatása

A fordítóprogramok olyan számítógépes szoftverek, amelyek egy adott programozási nyelven (forrásnyelv) írt programot képesek egy másik programozási nyelvre (célnyelv), vagy számítógépek által értelmezhető, futtatható gépi kódra átalakítani [5] [6].

2.1.1. Fordítóprogramok típusai

A fordítóprogramoknak számos fajtája van. A legismertebbek azok a fordítók, amelyek magas szintű programozási nyelven írt bemenetekből készítenek gépi vagy más alacsony szintű kimenetet. Ezek közé sorolható többek között a *clang*, *GCC* és a *javac* is. Ezeknek az ellentéte a *decompiler*, mely alacsony szintű (akár futtatható, gépi kódú) bemenetet alakít át magasabb absztrakciós szinten lévő nyelv kódjára. Létezik még továbbá úgynevezett *rewriter* fordító is, mely az eredeti forrásnyelven készít úgy kimenetet, hogy az lényegileg ne változzon (a bemeneti forráskódot és a kimenetet lefordítva ugyanazt a működést kell adnia a programnak). Ezek hasznosak lehetnek például kód refaktorálás vagy olyan API változtatások esetén, amikor egy könyvtár új verziója nem kompatibilis a korábbi verziókkal.

Source-to-source fordítók

Source-to-source fordítóprogramról akkor beszélhetünk, ha az adott fordító két – nagyjából megegyező absztrakciós szinten lévő – programozási nyelv között fordít. A fordítóprogramok ezen típusának története körülbelül az 1970-es évek második felére vezethető vissza. Ebben az időben ugyanis nem volt még jelen egy általánosan elfogadott szabvány a processzorok utasításkészletére, sőt, a technológia fejlődésével az utasításkészletek rohamosan változtak, így például az Intel egyes termékei sem voltak kompatibilisek egy korábbi architektúrára írt programmal. Ezt azonban érthető okokból igyekeztek elkerülni, a 16 bites 8086-os CPU-t például már úgy hozták forgalomba, hogy az elméletben kompatibilis volt a 8 bites 8080-ra írt programokkal. A kompatibilitást source-to-source fordítókkal valósították meg, azaz a két processzor utasításkészlete között megfeleltetéseket hoztak létre, és a fordítóprogram ennek megfelelően alakította át a szoftvert, hogy a más architektúrájú

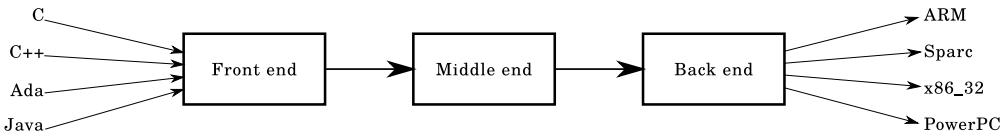
számítógépen is fusson [11].

2.2. Fordítóprogramok architektúrája

A fordítóprogramok általában három részre bonthatók:

- Front end
- Middle end
- Back end

A front end felelős a kód szintaktikai elemzéséért. Amennyiben a kód szintaktikailag helyes, akkor következik egy belső reprezentáció (*intermediate representation, IR*) létrehozása, amellyel majd a fordító a további műveletek során dolgozik. Bár a frontend feladatát akár egy program is elláthatná, a modularitás és a gyorsaság érdekében manapság általában három külön részre osztják: a lexer, mely a forráskód tokenizációjáért, azaz azért felel, hogy a kapott forráskódban szereplő elemeket osztályozza (pl. változó, operátor stb.). Szintén a frontend feladata a szintaktikai elemzés, tehát annak meghatározása, hogy a kód megfelel-e a nyelvtan szabályainak, valamint a szemantikai analízis, mely során többek között típusellenőrzést végez, valamint ellenőrzi, hogy a használni kívánt változókat deklaráltuk-e.



2.1. ábra. *Háromfázisú fordítóprogram-architektúra*

A middle end feladata az optimalizálás, melyet már a frontend által létrehozott belső reprezentáción végez. Az optimalizálás célja a teljesítmény maximalizálása.

A back end végzi a CPU-specifikus optimalizációkat, valamint ennek a modulnak a feladata a végleges kód generálása is.

3. fejezet

Saját munka bemutatása

A fordítóprogramok nagyon komplex szoftverek. A legismertebb C/C++ fordítóprogram talán a GCC (GNU Compiler Collection), amely óriási kódbázissal rendelkezik, 2019-ben körülbelül 15 millió sort tartalmaztak a forrásállományai [7]. Természetesen egy GCC szintű fordítóprogram számos olyan funkcióval rendelkezik, mely a több, mint 36 évnyi fejlesztésből és a projekt léptékéből adódik, ilyen például a többféle programozási nyelv támogatása, valamint a fejlett optimalizációs megoldások. Ezen funkciók közül jó néhányat egy felhasználóifelület-leíró nyelveket támogató fordítóprogramnak nem szükséges biztosítania, és mivel az Önálló laboratórium tárgy választott témájának keretében a fordítóprogramok felépítésének és készítésének, valamint a programozási nyelvek konstrukcióinak mélyebb megismerése állt a középpontban, így elsősorban ezekre fektettem nagy hangsúlyt az irodalomkutatás során, továbbá ilyen területekre fogok összpontosítani a fordítóprogramom implementálása alatt is.

3.1. Feladat specifikációja

Az Önálló laboratórium projekt keretében egy úgynevezett *source-to-source* fordítót készítek. Amint azt a 2.1. fejezetben ismertettem, az ilyen fordítók forrásfájlok közötti fordításra alkalmasak, azaz a bemenetként kapott fájlokból nem futtatható kódot generálnak, hanem egy másik nyelvre fordítják át azt. A projektfeladat keretében létrehozott fordító az 1. fejezetben és az 1.2. táblázatban bemutatott keretrendszerek felületleíró nyelveinek egy részhalmazát képes a QML, Qt XML és GTK XML között átalakítani.

Támogatott formátumok

A fentiek értelmében a program a tárgyalt két keretrendszer XML- és JSON-alapú nyelvei között teszi lehetővé a fordítást. Ennek megfelelően a bemeneti fájloknak az adott platform specifikációit kielégítően kell felépülniük, a kiterjesztésüknek pedig egyértelműen meghatározhatóvá kell tenni a forrásfájl típusát (azaz a `.ui` (Qt XML), `.qml` és `.glade` (GTK XML) hármas közül kell kikerülnie). A program ez alapján tudja eldönteni, hogy melyik értelmező (parser) modult kell meghívnia.

3.1.1. Közös részhalmaz meghatározása

Mivel a feladat során a hangsúly a fordítóprogramok felépítésének és működésének megismerésén van, továbbá az ismertetett keretrendszerek rengeteg felhasználásiterület-specifikus komponenssel rendelkeznek, melyeknek gyakran nincs is pontos megfelelője a másik keretrendszerben, ezért a fordítandó komponenseket az alapvető felhasználói felületi elemekre korlátoztam. Így végül egyszerű konténerek, gombok és beviteli mezők fordítására van lehetőség a programban.

A megfelelő részhalmaz kiválasztása a vártnál nagyobb kihívást jelentett, ugyanis sok felületi elem teljesen más tulajdonságokkal van ellátva a másik könyvtárhoz képest. Ilyen például, hogy GTK-ban alapvetően nem adható meg egy gomb pontos pozíciója és mérete, hanem kitölti a szülőkomponensben rendelkezésre álló helyet. Emiatt a gombok méretének beállítására csak akkor van lehetőség, ha a fordítás a két Qt-fájltípus, a `.ui` és a `.qml` között valósul meg (a fordítás iránya felcserélhető).

A programban lehetőség van a bemeneti fájl struktúrájának vizuális megjelenítésére is. Ehhez kimeneti fájlként egy `.svg` kiterjesztésű fájlt kell megadni. Ekkor a program az első két fázisban (ld. következő alfejezet) ugyanúgy viselkedik, mint általános esetben, de a kódgenerálási lépés során felületleíró fájl helyett egy vektorgrafikus állományt készít, mely egy irányított gráf formájában mutatja be a bemeneti fájl felépítését.

3.2. A program felépítése és működése

Ahogy arról már a 2.2 fejezetben is szó volt, a fordítóprogramok általában háromrétegűek, és a rétegek jól elkülöníthetőek. A munkám során törekedtem ezen architektúra követésére: külön egységbe kerültek a forrásnyelvelemző, a belső reprezentáció és a kódgenerálás forrásfájljai. Ezeken túl még röviden kitérek arra is, hogy a felhasználók hogyan használhatják a programot a parancssoron keresztül.

3.2.1. Parancssori interfész

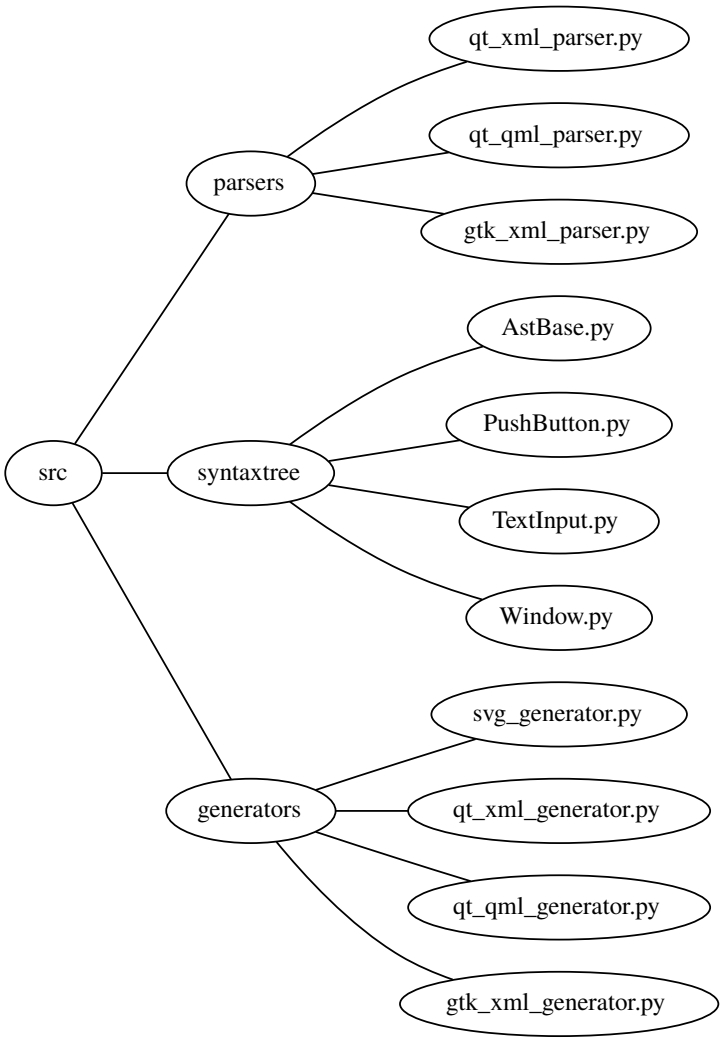
A felhasználók parancssoron keresztül indíthatják el a programot, itt van lehetőség a forrás- és kimeneti fájlok elérési útjának megadására. A megadott paramétereket a Python által biztosított `ArgumentParser` osztály segítségével dolgozzuk fel. A megadott paraméterek helyességét is ellenőrizzük, ez látható az alábbi kódrészleten:

```
1   <item row="1" column="0">
2     <widget class="QTimeEdit" name="timeEdit"/>
3   </item>
```

3.1. lista. *Hibaüzenetet előidéző kódrészlet*

3.2.2. Front end

A front end feladata, hogy ellenőrizze és értelmezze a kapott forrásfájlokat. A projektem során ide tartozó kódrészek a *parsers* mappában kaptak helyet.



3.1. ábra. *A program architektúrája*

3.2.3. Middle end

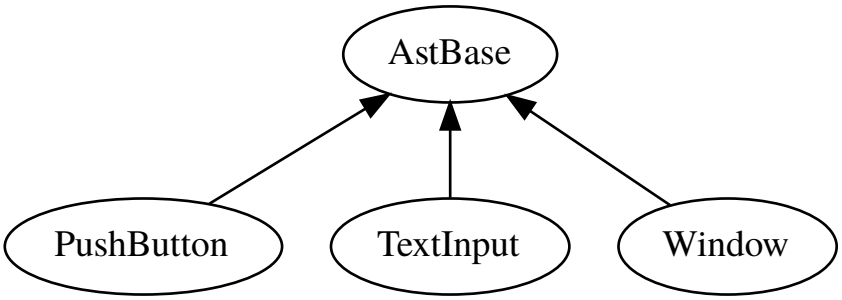
Esetemben a middle end feladata egy egységes reprezentáció biztosítása a felhasználói felület elemei számára, hogy azok a forrás- és célplatformtól függetlenül legyenek kezelhetőek a programon belül.

3.2.4. Back end

Kódgenerálás

3.2.5. Hibakezelés

A program implementálása során nagy hangsúlyt fektettem a hibakezelésre. Ez igaz mind a felhasználói parancsok értelmezésére, mind a program belső működésére, ezzel is könnyítve a későbbi bővítést és növelve a hibakeresés hatékonyságát.



3.2. ábra. *A belső reprezentáció felépítése*

```
1   <item row="1" column="0">
2     <widget class="QTimeEdit" name="timeEdit"/>
3   </item>
```

3.2. lista. *Hibaüzenetet előidéző kódrészlet*

3.3. Jelenlegi limitációk, továbbfejlesztési lehetőségek

A program jelenleg a Qt és a GTK eszközkészletének csak egy kis részét fedi le, így könnyű olyan funkciót vagy felhasználóifelületi elemet találni, amelynek implementálásával érdemes lehet foglalkozni a jövőben.

Mivel mindkét könyvtár többféle megoldást kínál a vezérlőelemek felhasználói felületen való elrendezésére, ezért kézenfekvőnek tűnhet a különböző konténerosztályok feldolgozásának támogatását kitűzni következő mérföldkőként. Ehhez a motivációt növeli, hogy az összetettebb felhasználói felületek mind használnak ilyen osztályokat, általában többet is, egymásba ágyazva. A program jelenleg nem támogatja teljeskörűen az ilyen felületeket: GTK és Qt közötti fordításnál például bizonyos bemenetek esetén előfordulhat, hogy az elemek nem az elvárt pozícióba kerülnek. Ennek legfőbb oka, hogy a GTK-s vezérlők a Qt-nál látottakkal ellentétben alapértelmezetten kitöltik a számukra rendelkezésre álló helyet, és ezt csak különböző konténerek bevezetésével lehet felülírni, ami Qt-ban látottakhoz képest sokkal összetettebb logikát igényel a programban.

A másik fő irány, amiben a program továbbfejlesztésre szorul, az az elérhető vezérlők számának növelése. Ahogy arra a 3.1.1 alfejezetben kitértem, a program jelenlegi állapotában csak néhány alapvető felhasználóifelületi elemet támogat, ami jelentősen korlátozza a felhasználási körét. Új vezérlők hozzáadása a program jelenlegi felépítése mellett már sokkal könnyebb, ugyanis már sok, az új elemek kezeléséhez szükséges megoldásra szükség volt a jelenlegi állapot eléréséhez is, ezek pedig a program moduláris mivoltából adódóan könnyen újrafelhasználhatóak. Ilyen megoldásnak tekinthető például a belső reprezentáció alaposztálya, az `AstBase`, melyből egy új osztályt leszármaztatva vehetjük fel a kívánt felületi elemet a programba. Emellett rendelkezésre állnak már egyes adatstruktúrák és fabejáró függvények a jövőbeni munka megkönnyítésére.

Irodalomjegyzék

[1] Qt Group. *Qt for Open Source Development*, May 2023. URL: https://www.qt.io/download-open-source.

[2] Qt. *Getting Started Programming with Qt Widgets*, May 2023. URL: https://doc.qt.io/qt-5/qtwidgets-tutorials-notepad-example.html.

[3] Qt Group. *Qt Educational license for students and teachers*, May 2023. URL: https://www.qt.io/qt-educational-license.

[4] Olaf Schmidt-Wischhöfer. *Qt, Open Source and corona.* KDE, April 2020. URL: https://mail.kde.org/pipermail/kde-community/2020q2/006098.html.

[5] Wikipedia. *Compiler*, May 2023. URL: https://en.wikipedia.org/wiki/Compiler.

[6] Wikipedia. *Fordítóprogram*, May 2023. URL: `https://hu.wikipedia.org/wiki/Ford%C3%ADt%C3%B3program`.

[7] Wikipedia. *GNU Compiler Collection*, May 2023. URL: https://en.wikipedia.org/wiki/GNU_Compiler_Collection.

[8] Wikipedia. *GTK*, May 2023. URL: https://en.wikipedia.org/wiki/GTK.

[9] Wikipedia. *QML*, May 2023. URL: https://en.wikipedia.org/wiki/QML.

[10] Wikipedia. *Qt (Software)*, May 2023. URL: https://en.wikipedia.org/wiki/Qt_(software).

[11] Wikipedia. *Source-to-source compiler*, May 2023. URL: https://en.wikipedia.org/wiki/Sourcetosource_compiler.