изменять текущий язык. Обработчик событий по умолчанию для подклассов **QWidget** реагирует на событие QEvent:: LanguageChange, и вызовет эту функцию при необходимости.

Событие LanguageChange выбросится, когда пользователь установит новый перевод (выполнится функция QCoreApplication::installTranslator()). Кроме того, другие компоненты приложения могут заставить виджеты обновить себя, отправив им событие LanguageChange.

9.5. Лабораторная работа № 7 «Работа с графикой и интернационализация»

9.5.1. Цель лабораторной работы

Целью лабораторной работы является знакомство с предоставляемыми MeeGo SDK средствами отображения двухмерной графики и интернационализации.

9.5.2. Введение

В рамках этой лабораторной работы мы создадим приложение, отображающее выбранную пользователем векторную графику (в виде svg-файла) и предоставляющее возможность динамически (в процессе работы программы) менять язык отображения интерфейса. В соответствии этими двумя функциями разделим наше приложение на две части: первая будет включать в себя работу с графикой; вторая — работу со средствами интернационализации. Для демонстрации двухмерной графики с использованием классов QPainter предоставим приложению возможность отображать выбранный файл поверх подложки, нарисованной с помощью этих интерфейсов.

План работы

Создание приложения включает в себя:

1. создание нового приложения;
2. создание проектного файла, продумывание общей структуры проекта;
3. создание ресурсного файла;
4. разработка интерфейса пользователя; описание слотов и событий интерфейса;
5. программирования основной части приложения;
6. интернационализация приложения.

Необходимые знания и навыки

Предполагается, что пользователь выполнил предыдущие лабораторные работы, имеет навыки программирования на языке C++ и прослушал соответствующую лекцию.

Необходимые программные и аппаратные средства

Предполагается, что для работы пользователя установлено MeeGo SDK версии не ниже 1.0, налажен процесс разработки приложений и установлен соответствующий набор библиотек, необходимых для кросскомпиляции.

9.5.3. Инструкция по выполнению работы

9.5.3.1. Создание нового приложения

- Войдите в *Qt Creator*

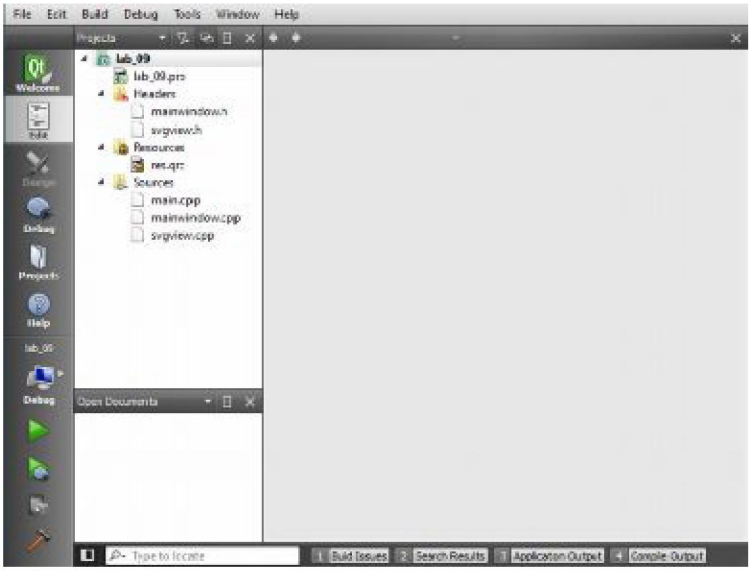

Рис. 9.5.1. Qt Creator.

- Создайте новый проект (“File->New File or Project”)
- В открывшемся диалоге (Рис. 9.5.2) выберите пункт “Mobile Qt Application”

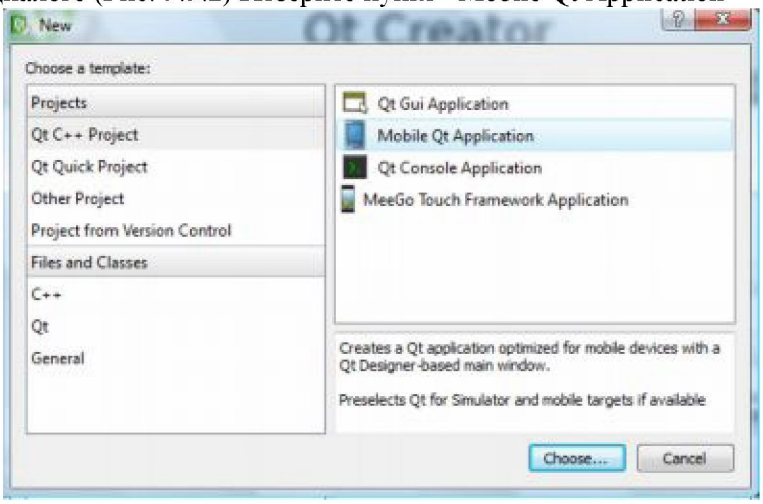

Рис. 9.5.2. Создание нового приложения.

- Перейдите к следующему шагу, нажав кнопку “Choose”

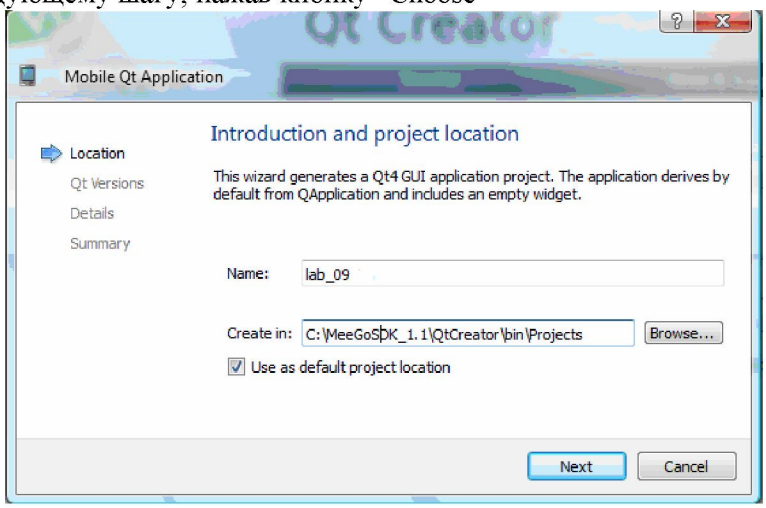

Рис. 9.5.3. Выбор имени и расположения проекта.

- Введите название проекта (“lab_09”) и выберете его расположение
- Перейдите к следующему шагу, нажав кнопку “Next”
- Укажите целевые платформы для которых будет осуществляться кросскомпиляция (в данном случае это **meego-handset**, Рис. 9.5.4)

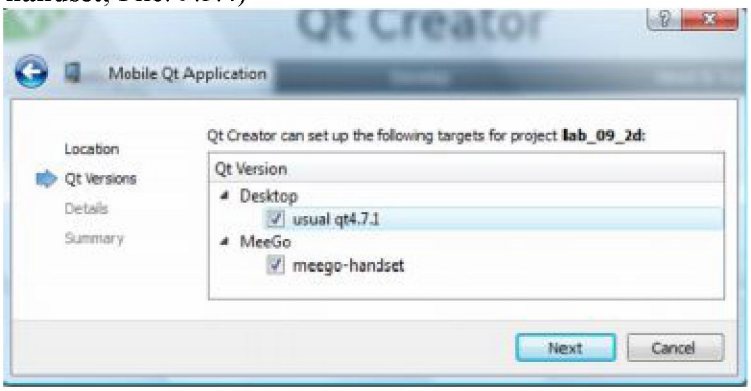

Рис. 9.5.4. Выбор целевых платформ.

- Оставляем нетронутыми названия генерируемых классов и снимаем галочку с Generate form (Рис. 9.5.5) — при разработке приложения **не будет** использован **Qt Designer**)

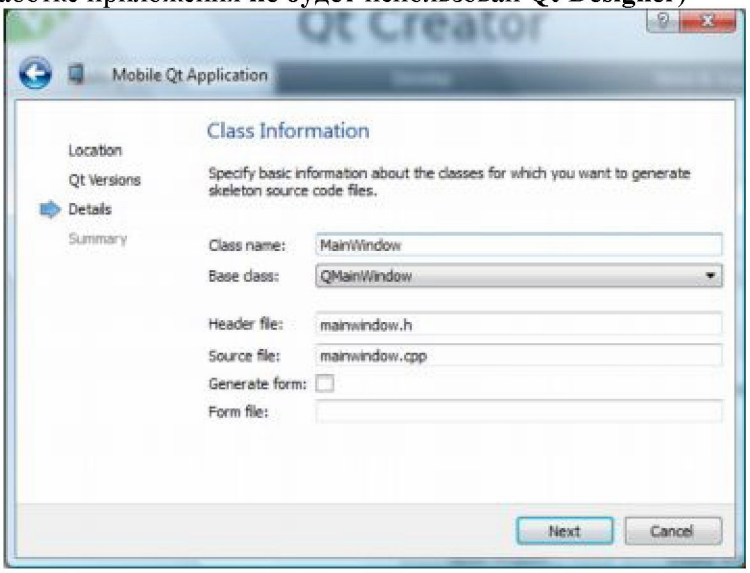

Рис. 9.5.5. Генерируемые классы.

- Добавьте проект под управление системой контроля версий, установленной на ПК — в данном случае, это GIT (Рис. 9.5.6)

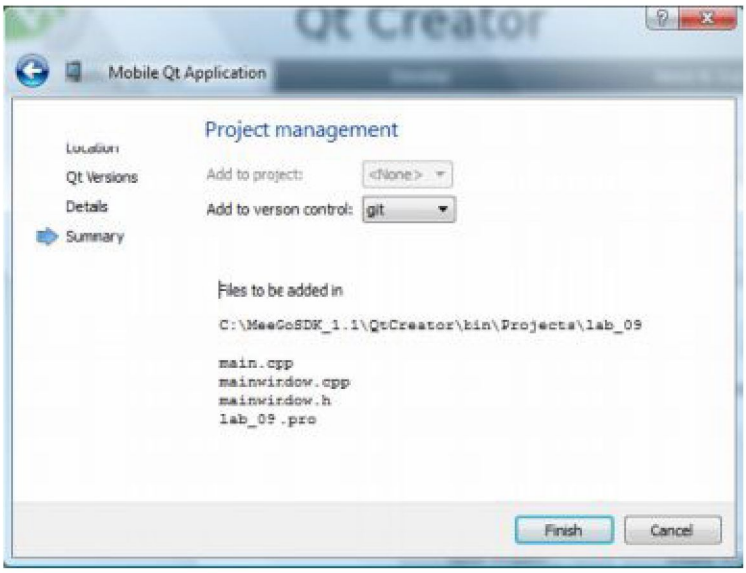

Рис. 9.5.6. Контроль версий.

9.5.3.2. Подготовка проектного файла

Для того чтобы наше приложение скомпилировалось необходимо правильно описать проектный файл (*.pro). Для этого отредактируем сгенерированный автоматически файл **lab_09.pro**.

```
QT += core gui svg
TARGET = lab_09
TEMPLATE = app
target.path=/usr/local/bin

INSTALLS = target
SOURCES += main.cpp\
                        mainwindow.cpp \
                        svgview.cpp
HEADERS  += mainwindow.h \
                         svgview.h
RESOURCES += res.qrc
TRANSLATIONS = lab09_en.ts \
                          lab09_ru.ts
```

Обратим внимание на строки **TRANSLATIONS** (забегая вперёд, данная строка определяет используемые в приложении языковые файлы) и **RESOURCES** (определяет ресурсный файл, в котором указываются все ресурсы, компилируемые в исполняемый файл). Кроме того, обратим внимание на строку “QT=core gui svg” — в ней мы добавили необходимые библиотеки для работы с интерфейсом пользователя и векторной графикой в формате SVG.

В строках **SOURCES** и **HEADERS** фактически описана структура нашего приложения:

- в главном файле “main.cpp” осуществляется загрузка приложения и подключение нужного перевода;
- файл “mainwindow.cpp” описывает главное окно программы, а также слоты и события;
- “svgview.cpp” описывает класс виджета в котором будет отображаться svg-графика.

9.5.3.3. Подготовка ресурсного файла

В ресурсном файле мы указываем изображения, переводы, иконки и т. д. (все ресурсы), которые скомпилирует в конечный исполняемый файл. Для нашего проекта мы хотим подготовить файл содержащий:

- svg-изображение, появляющееся по умолчанию при запуске программы (файл “test.svg” — предварительно поместим его в созданную подпапку “files/images/” проекта);
- логотип MeeGo, которое мы загрузим используя QPainter (файл “files/images/meego_logo.svg”);
- и объектные файлы, содержащие перевод на русский и английский (“files/translations/lab09[_ru|_en].qm”).

Ресурсные файлы в Qt имеют расширение *.qrc, являются правильными xml файлами и могут быть созданы либо вручную, либо через *“File->New File->Qt->Qt Resource File”*. Если файл добавлялся и редактировался вручную, то необходимо не забыть добавить его под управление используемой системы контроля версий (для GIT — *git add res.qrc*).

Ресурсный файл нашего проекта (res.qrc) будет иметь вид:

```
<RCC>
    <qresource prefix="/">
        <file>files/images/test.svg</file>
        <file>files/images/meego_logo.jpeg</file>
        <file>files/translations/lab09_ru.qm</file>
        <file>files/translations/lab09_en.qm</file>
    </qresource>
</RCC>
```

9.5.3.4. Разработка интерфейса пользователя, описание слотов и событий

В проекте интерфейс будет создаваться вручную, без использования визуальных средств, предоставляемых Qt Designer. Это связано с тем, что несмотря на внешнюю простоту, работа с Qt Designer требует уверенного знания принципов и концепций лежащих в основе организации графического интерфейса средствами Qt. Рассмотрение данных концепций лежит вне предмета данной работы.

Интерфейс пользователя будет включать выпадающее меню, с использованием которого будет происходить работа с приложением. Для этих целей в файле “mainwindow.cpp” в конструкторе **MainWindow::MainWindow()** добавим строки:

```
this->fileMenu = new QMenu(tr("&File"), this);
        openAction = fileMenu->addAction(tr("&Open..."));
        openAction->setShortcut(QKeySequence("Ctrl+O"));
        quitAction = fileMenu->addAction(tr("E&xit"));
        quitAction->setShortcuts(QKeySequence::Quit);

this->viewMenu = new QMenu(tr("&View"), this);
        m_backgroundAction = viewMenu->addAction(tr("&Background"));
        m_backgroundAction->setEnabled(false);
        m_backgroundAction->setCheckable(true);
        m_backgroundAction->setChecked(false);
        m_outlineAction = viewMenu->addAction(tr("O&utline"));
        m_outlineAction->setEnabled(false);
        m_outlineAction->setCheckable(true);
        m_outlineAction->setChecked(true);

 this->windowBkgMenu = new QMenu(tr("&Background"), this);
        m_chessAction = windowBkgMenu->addAction(tr("&Chess"));
        m_chessAction->setCheckable(true);
        m_chessAction->setChecked(true);
        m_meegoAction = windowBkgMenu->addAction(tr("&MeeGo"));
        m_meegoAction->setCheckable(true);
        QActionGroup *bkgGroup = new QActionGroup(this);
            bkgGroup->addAction(m_chessAction);
            bkgGroup->addAction(m_meegoAction);
```

```
this->langMenu = new QMenu(tr("&Language"), this);
       m_engAction = langMenu->addAction(tr("&English"));
       m_engAction->setCheckable(true);
       m_engAction->setChecked(true);
       m_rusAction = langMenu->addAction(tr("&Russian"));
       m_rusAction->setCheckable(true);
       QActionGroup *langGroup = new QActionGroup(this);
           langGroup->addAction(m_rusAction);
           langGroup->addAction(m_engAction);

       menuBar()->addMenu(this->fileMenu);
       menuBar()->addMenu(this->viewMenu);
       menuBar()->addMenu(this->windowBkgMenu);
       menuBar()->addMenu(this->langMenu);

       setCentralWidget(m_view);
       setWindowTitle(tr("SVG Viewer"));
```

Переменные **langMenu, windowBkgMenu, viewMenu** и другие, описаны в заголовочном файле “mainwindow.h”. Метод **QMenu::addAction** добавляет соответствующее действие в выпадающий пункт меню. Вызовы **openAction->setShortcut(QKeySequence("Ctrl+O"))** добавляют клавиатурные сокращения для пунктов меню. Вызовы методов **setEnabled, setCheckable** и **setChecked** устанавливают доступность элементов меню и их исходные состояния. Вызовы методов **addMenu** прикрепляют описанные ранее пункты меню; **setWindowTitle** — устанавливает заголовок текущего окна.

Далее необходимо добавить описанным элементам меню обработчики с помощью системы слотов и сигналов Qt. Для этого в конструкторе допишем следующие строки:

```
connect(quitAction, SIGNAL(triggered()), qApp, SLOT(quit()));
connect(openAction, SIGNAL(triggered()), this, SLOT(openFile()));
connect(m_backgroundAction, SIGNAL(toggled(bool)), m_view,
SLOT(setViewBackground(bool)));
connect(m_outlineAction, SIGNAL(toggled(bool)), m_view,
SLOT(setViewOutline(bool)));
connect(bkgGroup, SIGNAL(triggered(QAction*)),
             this,           SLOT(setWindowBackground(QAction*)));
```

Теперь необходимо описать соответствующие обработчики событий. Первым, в файле “mainwindow.cpp” опишем стандартный диалог открытия svg-файлов:

```
void MainWindow::openFile(const QString &path)
{
    QString fileName;

    if (path.isNull())
        fileName = QFileDialog::getOpenFileName(this, tr("Open SVG File"),
                m_currentPath, "SVG files (*.svg *.svgz *.svg.gz)");
    else
        fileName = path;

    if (!fileName.isEmpty()) {
        QFile file(fileName);

        if (!file.exists()) {
            QMessageBox::critical(this,
                                                  tr("Open SVG File"),
                                                  tr("Could not open file
'%1'.").arg(fileName));
```

```
            m_outlineAction->setEnabled(false);
            m_backgroundAction->setEnabled(false);
            return;
        }

        m_view->openFile(file);
        m_currentPath = fileName;
        setWindowTitle(tr("%1 - SVGViewer").arg(m_currentPath));

        m_outlineAction->setEnabled(true);
        m_backgroundAction->setEnabled(true);
        resize(m_view->sizeHint() + QSize(80, 80 + menuBar()->height()));
    }
}
```

Далее добавим выбор подложки.

```
void MainWindow::setWindowBackground(QAction *action) {
    if (action == m_chessAction){
        m_view->setBkgPaint(0);//chess
    } else if(action == m_meegoAction){
        m_view->setBkgPaint(1);//meego
    }
}
```

Эта функция в зависимости от действия будет вызывать с соответствующим параметром метод **setBkgPaint** класса **SvgView**.

9.5.3.5. Основная часть приложения

Итак, перейдём к основной части приложения. Главная функция (файл "**main.cpp**") будет выглядеть следующим образом:

```
int main(int argc, char **argv)
{
    Q_INIT_RESOURCE(res);
    QApplication app(argc, argv);

    MainWindow window;
    if (argc == 2)
        window.openFile(argv[1]);
    else
        window.openFile(":/files/images/test.svg");
    window.show();

    return app.exec();
}
```

Вызов макроса **Q_INIT_RESOURCES** инициализирует ресурсы; ветвление "**if (argc == 2)**" позволяет запускать программу из консоли, с указанием в качестве параметра файла, который необходимо открыть. По умолчанию используется хранящийся в ресурсах svg-файл ":/files/images/test.svg" (**":/"** означает ссылку на объект внутри ресурсного файла).

В файле svgview.cpp опишем конструктор виджета:

```
SvgView::SvgView(QWidget *parent)
    : QGraphicsView(parent)
    , m_svgItem(0)
    , m_backgroundItem(0)
    , m_outlineItem(0)
{
```

```
    setScene(new QGraphicsScene(this));
    setViewportUpdateMode(FullViewportUpdate);
    setBkgPaint(0);//set chess background
}
```

Наш виджет наследуется от класса **QGraphicsView** (смотреть заголовочный файл “svgview.h” в приложении к лабораторной работе), таким образом он наследует методы **setScene** — который прикрепляет новую графическую сцену к виджету, **setViewportUpdateMode**, устанавливающий способ обновления видимой области и прочие (полное описание методов можно найти в документации к Qt). Вызов метода **setBkgPaint** позволяет установить подложку по умолчанию. Он будет иметь следующий вид:

```
void SvgView::setBkgPaint(int idx) {
    QPixmap tilePixmap(64, 64);
    switch (idx) {
        case 0 : {//chess
                tilePixmap.fill(Qt::white);
                QPainter tilePainter(&tilePixmap);
                QColor color(220, 220, 220);
                tilePainter.fillRect(0, 0, 32, 32, color);
                tilePainter.fillRect(32, 32, 32, 32, color);
                tilePainter.end();
                break;
        }
        case 1 : {//meego
                tilePixmap.fill(Qt::white);
                QPainter tilePainter(&tilePixmap);
                QColor color(220, 220, 220);
                QImage *meego_img = new
QImage(":/files/images/meego_logo.jpeg");
                tilePainter.drawImage(0,10, meego_img-
>scaled(QSize(64,32),Qt::KeepAspectRatio));
                tilePainter.end();
                break;
        }
    }
    setBackgroundBrush(tilePixmap);
}
```

Здесь в каждом случае мы рисуем на тайлах (повторяющихся изображениях), размером 64x64 пикселей. Вызов метода **fill** с параметром **Qt::white** очищает подложку. Вызов конструктора **QPainter tilePainter(&tilePixmap)** позволяет создать новый объект, который осуществляет рисование на тайле. Метод **fillRect** заливают указанный прямоугольник; **drawImage** — рисует заданное изображение. Вызов метода **end** применяет изменения.

В следующей функции инициируется рисование подложки:

```
void SvgView::drawBackground(QPainter *p)
{
    p->save();
    p->resetTransform();
    p->drawTiledPixmap(viewport()->rect(), backgroundBrush().texture());
    p->restore();
}
```

Далее описываются слоты, которые позволяют пользователю установить окантовку и подложку для svg-файла.

```
void SvgView::setViewBackground(bool enable)
{
    if (!m_backgroundItem)
          return;
```

```
    m_backgroundItem->setVisible(enable);
}

void SvgView::setViewOutline(bool enable)
{
    if (!m_outlineItem)
        return;

    m_outlineItem->setVisible(enable);
}
```

И наконец, опишем метод **openFile**, в котором осуществляется вывод svg-графики из указанного файла:

```
void SvgView::openFile(const QFile &file)
{
    if (!file.exists())
        return;

    QGraphicsScene *s = scene();

    bool drawBackground = (m_backgroundItem ? m_backgroundItem->isVisible() :
false);
    bool drawOutline = (m_outlineItem ? m_outlineItem->isVisible() : true);

    s->clear();
    resetTransform();

    m_svgItem = new QGraphicsSvgItem(file.fileName());
    m_svgItem->setFlags(QGraphicsItem::ItemClipsToShape);
    m_svgItem->setCacheMode(QGraphicsItem::NoCache);
    m_svgItem->setZValue(0);

    m_backgroundItem = new QGraphicsRectItem(m_svgItem->boundingRect());
    m_backgroundItem->setBrush(Qt::white);
    m_backgroundItem->setPen(Qt::NoPen);
    m_backgroundItem->setVisible(drawBackground);
    m_backgroundItem->setZValue(-1);

    m_outlineItem = new QGraphicsRectItem(m_svgItem->boundingRect());
    QPen outline(Qt::black, 2, Qt::DashLine);
    outline.setCosmetic(true);
    m_outlineItem->setPen(outline);
    m_outlineItem->setBrush(Qt::NoBrush);
    m_outlineItem->setVisible(drawOutline);
    m_outlineItem->setZValue(1);

    s->addItem(m_backgroundItem);
    s->addItem(m_svgItem);
    s->addItem(m_outlineItem);

    s->setSceneRect(m_outlineItem->boundingRect().adjusted(-10, -10, 10, 10));
}
```

9.5.3.6. Интернационализация приложения

В учебных целях в приложении реализуется перевод на русский и английские языки.

Для того чтобы осуществить перевод приложения необходимо выполнить следующие шаги:

- Перейти в корень проекта.

```
> cd /home/user/qt/projects/lab_09
```

- Вызвать утилиту **lupdate** с указанием проектного файла.

```
> lupdate ./lab_09.pro
```

В результате вызова данной утилиты мы получим набор из двух файлов: **lab09_en.ts** и **lab09_ru.ts**, которые будут содержать маркированные вызовами функции **tr()** и макроса **QT_TR*_NOOP()** литерные тексты, содержавшиеся исходном C++ коде.

- Осуществить перевод маркированного литерного текста — для этого необходимо либо отредактировать получившийся .TS файл (который содержит правильный XML), либо использовать утилиту **Qt Linguist** (Рис. 9.5.6). Мы воспользуемся утилитой.

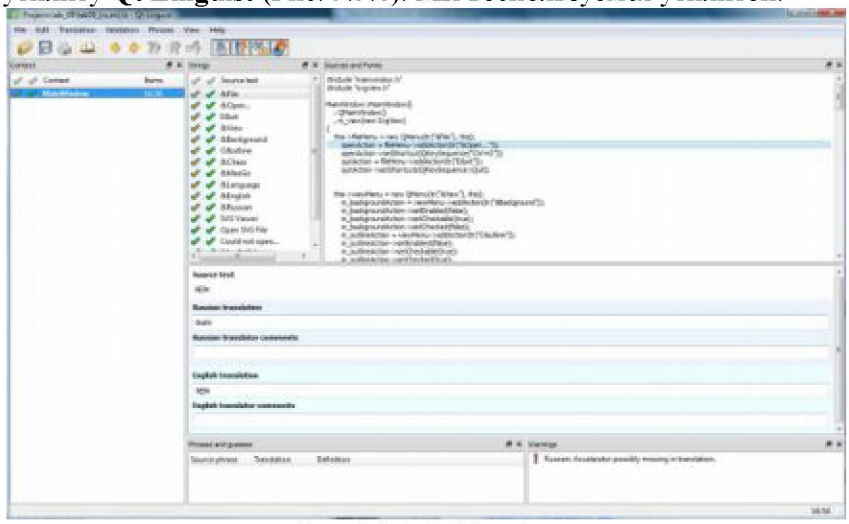

Рис. 9.5.6. Qt Linguist.

- В Qt Linguist необходимо открыть имеющиеся .TS файлы и осуществить перевод каждой фразы.
- Вызвать утилиту **lrelease** для того чтобы получить легковесный файл с сообщениями (.QM файл). Данный файл пригоден только для конечного использования.
- Скопируем получившиеся .QM файлы в пути, указанные в ресурсном файле (“./files/translations/lab09_en.qm” и “./files/translations/lab09_ru.qm”).

Подключим переведённые файлы, для этого в главном файле “main.cpp” добавим строки

```
QTranslator appTranslator;
appTranslator.load("lab09_en.qm",":/files/translations/");
app.installTranslator(&appTranslator);
window.setTranslator(&appTranslator);
```

Метод **QTranslator::load()** загружает перевод; **QCoreApplication::installTranslator()** устанавливает выбранный переводчик; метод **MainWindow::setTranslator()** устанавливает перевод для главного окна.

- Добавим метод **MainWindow::retranslateUI()**, вызываемый когда пользователь инициирует переключение языка. В данном методе мы поочерёдно будем переназначать название каждого ранее маркированного элемента. В файл “mainwindow.cpp” добавим:

```
void MainWindow::retranslateUI() {

   this->fileMenu->setTitle(tr("&File"));
   this->viewMenu->setTitle(tr("&View"));
   this->windowBkgMenu->setTitle(tr("&Background"));
   this->langMenu->setTitle(tr("&Language"));
```

```
    setWindowTitle(tr("SVG Viewer"));
    this->m_backgroundAction->setText(tr("B&ackground"));
    this->m_outlineAction->setText(tr("O&utline"));
    this->m_chessAction->setText(tr("&Chess"));
    this->m_meegoAction->setText(tr("&MeeGo"));
    this->m_engAction->setText(tr("&English"));
    this->m_rusAction->setText(tr("&Russian"));
    this->openAction->setText(tr("&Open..."));
    this->quitAction->setText(tr("E&xit"));

}
```

- Соединим соответствующие слоты-события, связанные с переключением языка. Для этого добавим в файл “mainwindow.cpp” описание слота **setWindowLang**

```
void MainWindow::setWindowLang(QAction *action)
{
    if (action == m_engAction) {
        this->translator->load("lab09_en.qm",":/files/translations/");
    } else if(action == m_rusAction) {
        this->translator->load("lab09_ru.qm",":/files/translations/");
    }
    retranslateUI();
}
```

и соединим элементы в конструкторе

```
connect(langGroup, SIGNAL(triggered(QAction*)),
            this,          SLOT(setWindowLang(QAction*)));
```

9.6. Выводы

В этой лекции мы провели обзор возможностей MeeGo API для программирования 3D графики — были рассмотрели основные реализации OpenGL ES (1.X и 2.X), и их различия. Кроме того, изучили средства работы с векторной графикой (по средствам SVG формата) и простейшего рисования с помощью QPainter. Также были рассмотрены основные способы интернационализации, используемые в MeeGo API и их сценарии использования.

При выполнении лабораторной работе № 7 читатели могут получить практические навыки работы с векторной и двухмерной графикой, а также со средствами интернационализации, предоставляемыми MeeGo API.

9.7. Контрольные вопросы

1) Основным классом для работы с 2D графикой в MeeGo API является класс:
    1. QPainter
    2. QSvg
    3. QWidget
    4. QImage
2) Три основных класса, предназначенных для работы с двухмерной графикой в MeeGo API:
    1. QSvg -> QWidget -> QImage
    2. QWidget -> QPainter-> QPaintDevice
    3. QPainter -> QPaintEngine -> QPaintDevice
3) В качестве устройства вывода **не может быть использован** экземпляр класса:
    1. QWidget
    2. QImage