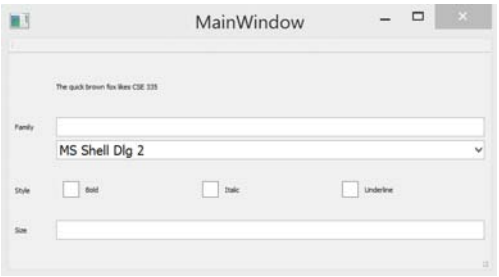
Centralized Collaboration - Mediator Pattern

Alex X. Liu

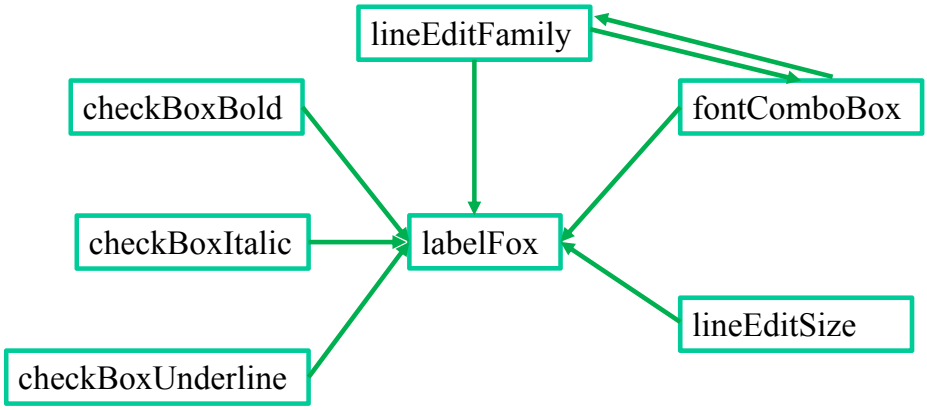
Two Collaboration Models

- OO software=configuration + collaboration
- Distributed Collaboration
  - You know whom you work with and whom you impact.
- Centralized Collaboration
  - You do not know whom you work with and whom you impact.
  - Recall how CIA agents work together? Each only knows the manager.
  - This is called Mediator Pattern.

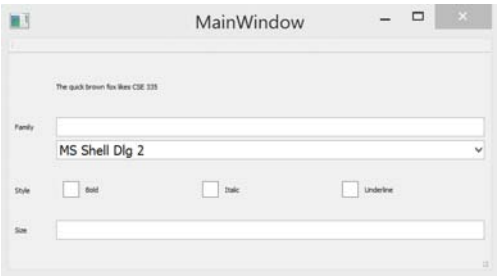
Distributed Collaboration



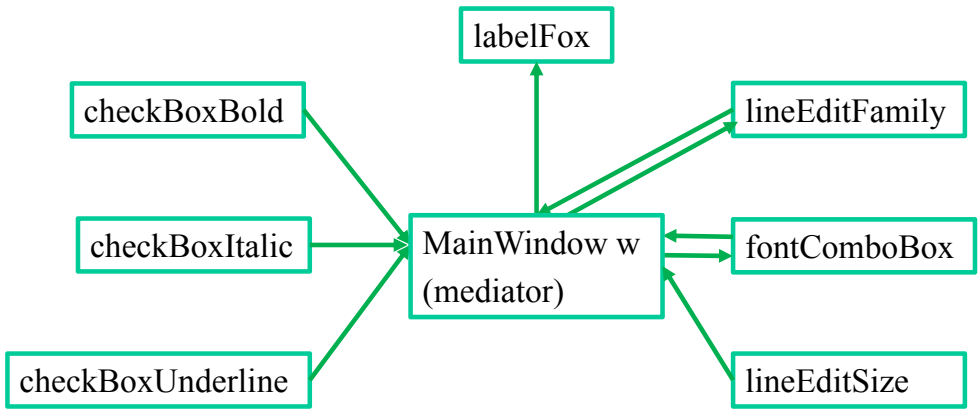
- Weaknesses of Distributed Collaboration
  - Subclasses are tightly coupled, which reduces reusability.
  - Difficult to change system behavior because system behavior is distributed among many objects.



Centralized Collaboration



- Advantages of Centralized Collaboration
  - Classes are loosely coupled, which increases reusability.
    - Each only knows the mediator.
  - Each to change system behavior
    - Just modify the mediator.



alexlineedit.h

```
#ifndef ALEXLINEEDIT_H
#define ALEXLINEEDIT_H
#include <QLineEdit>
class AlexLineEdit: public QLineEdit{
   Q_OBJECT

public:
   AlexLineEdit(const QString& qstring):QLineEdit(qstring){};
   AlexLineEdit(QWidget* qw):QLineEdit(qw){};

signals:
   void iChanged(QObject*); //My own signal

public slots:
   //void myTextChanged(const QString&);
   //For receiving its predefined signal. In its implementation, I emit my own signal.
   void myEditingFinished();
};
#endif // ALEXLINEEDIT_H
```


alexfontcombobox.h

```
#ifndef ALEXFONTCOMBOBOX_H
#define ALEXFONTCOMBOBOX_H
#include <QFontComboBox>

class AlexFontComboBox: public QFontComboBox{
   Q_OBJECT

public:
   AlexFontComboBox(QWidget* qw):QFontComboBox(qw){};

signals:
   void iChanged(QObject*); //My own signal

public slots:
   //For receiving its predefined signal. In its implementation, I emit my own signal.
   void myCurrentFontChanged(const QFont&);
};

#endif // ALEXFONTCOMBOBOX_H
```

alexcheckbox.h

```
#ifndef ALEXCHECKBOX_H
#define ALEXCHECKBOX_H
#include <QCheckBox>

class AlexCheckBox: public QCheckBox{
   Q_OBJECT

public:
   AlexCheckBox(QWidget* qw):QCheckBox(qw){};
   AlexCheckBox(QString qs):QCheckBox(qs){};

signals:
   void iChanged(QObject*); //My own signal

public slots:
   //For receiving its predefined signal. In its implementation, I emit my own signal.
   void myStateChanged(int);
};
#endif // ALEXCHECKBOX_H
```


mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QFont>
namespace Ui {class MainWindow;}

class MainWindow : public QMainWindow{
   Q_OBJECT
public:
   explicit MainWindow(QWidget *parent = 0);
   ~MainWindow();
public slots:
   void actByYourChange(QObject*);
private:
   Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

alexlineedit.cpp, alexfontcombobox.cpp, alexcheckbox.cpp

```
#include "alexlineedit.h"
void AlexLineEdit::myEditingFinished(){
   emit iChanged(this);
}
```

```
#include "alexfontcombobox.h"
void AlexFontComboBox::myCurrentFontChanged(const QFont&){
   emit iChanged(this);
}
```

```
#include "alexcheckbox.h"
void AlexCheckBox::myStateChanged(int){
   emit iChanged(this);
}
```


mainwindow.cpp

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow){
   ui->setupUi(this);
   //Note: Use SIGNAL(editingFinished()), not SIGNAL(textChanged(QString)), otherwise, it will cause circular dependency.
   //connect(ui->lineEditFamily,SIGNAL(textChanged(QString)),ui->lineEditFamily,SLOT(myTextChanged(QString)));
   connect(ui->lineEditFamily,SIGNAL(editingFinished()),ui->lineEditFamily,SLOT(myEditingFinished()));
   connect(ui->lineEditFamily,SIGNAL(iChanged(QObject*)),this,SLOT(actByYourChange(QObject*)));

   connect(ui->fontComboBox,SIGNAL(currentFontChanged(QFont)),
           ui->fontComboBox,SLOT(myCurrentFontChanged(QFont)));
   connect(ui->fontComboBox,SIGNAL(iChanged(QObject*)),this,SLOT(actByYourChange(QObject*)));

   connect(ui->checkBoxBold,SIGNAL(stateChanged(int)),ui->checkBoxBold,SLOT(myStateChanged(int)));
   connect(ui->checkBoxBold,SIGNAL(iChanged(QObject*)),this,SLOT(actByYourChange(QObject*)));

   connect(ui->checkBoxItalic,SIGNAL(stateChanged(int)),ui->checkBoxItalic,SLOT(myStateChanged(int)));
   connect(ui->checkBoxItalic,SIGNAL(iChanged(QObject*)),this,SLOT(actByYourChange(QObject*)));

   connect(ui->checkBoxUnderline,SIGNAL(stateChanged(int)),ui->checkBoxUnderline,SLOT(myStateChanged(int)));
   connect(ui->checkBoxUnderline,SIGNAL(iChanged(QObject*)),this,SLOT(actByYourChange(QObject*)));

   connect(ui->lineEditSize,SIGNAL(editingFinished()),ui->lineEditSize,SLOT(myEditingFinished()));
   connect(ui->lineEditSize,SIGNAL(iChanged(QObject*)),this,SLOT(actByYourChange(QObject*)));
}

MainWindow::~MainWindow(){delete ui;}
```

mainwindow.cpp

```
void MainWindow::actByYourChange(QObject* senderObj){
  if(senderObj==ui->lineEditFamily){
     ui->labelFox->setFont(QFont(ui->lineEditFamily->text()));
     ui->fontComboBox->setCurrentFont(QFont(ui->lineEditFamily->text()));
  }else if(senderObj==ui->fontComboBox){
     ui->labelFox->setFont(ui->fontComboBox->currentFont());
     ui->lineEditFamily->setText(ui->fontComboBox->currentFont().family());
  }else if(senderObj==ui->checkBoxBold){
     QFont font=ui->labelFox->font();
     font.setBold(ui->checkBoxBold->isChecked());
     ui->labelFox->setFont(font);
  }else if(senderObj==ui->checkBoxItalic){
     QFont font=ui->labelFox->font();
     font.setItalic(ui->checkBoxItalic->isChecked());
     ui->labelFox->setFont(font);
  }else if(senderObj==ui->checkBoxUnderline){
     QFont font=ui->labelFox->font();
     font.setUnderline(ui->checkBoxUnderline->isChecked());
     ui->labelFox->setFont(font);
  }else if(senderObj==ui->lineEditSize){
     QFont font=ui->labelFox->font();
     font.setPointSize(ui->lineEditSize->text().toInt());
     ui->labelFox->setFont(font);
  }
}
```