Seminar "'BSE"'

Winterterm 2010

Code-Cloning from the Victims Point of View

Holger Hans Peter Freyther
freyther@mi.fu-berlin.de

November 17, 2010

Abstract

This summary describes the experience with Code-Clones in the Qtopia Framework of Trolltech ASA, their manifestion and hypothesis on their existence. The paper ends with a look at the research of Cory Kapser and his description of Code Cloning Patterns then apply these to the clones of Qtopia.

Contents

1 Introduction

This summary will take a look at code clones in the Qtopia product. We will start with the definition of code cloning, the Qtopia product, the company behind it and the context in which the code clones were encountered. The second section intends to understand the code clones, the reason why there were found and why they became a problem and proposes an abstraction to avoid these. The third section will introduce the clone patterns identified by Cory Kapser in association with Michael W. Godfrey in "[Cloning Considered Harmful] Considered Harmful" and attempts to apply these to the previously found patterns.

2 Fundamentals

2.1 Code Clone

A code clone is a non trivial amount of redundant code[KG06]. It is believed that 10-15% of the source code in large software systems are code clones [KG04]. They can be created intentionally, e.g. by "copy and pasting", or unintentionally due to language constructs.

2.2 Trolltech ASA

Trolltech ASA was a Norwegian based software company and the producer of the Qt Framework for C++. Founded in 1994, one of its early success was the adoption of Qt by the KDE Free Software Project. The company had its main office located in Oslo, a tools office in Berlin and their Australian office was in Brisbane.

2.2.1 Qtopia in Australia

Trolltech introduced the Qt/Embedded product for Linux in March 2000[Tro00]. This product included its own Windowing System and could be used directly on top of the Linux Framebuffer device. To provide a showcase and help selling this new product the team in Australia created the Q Palmtop Environment (QPE). New applications were written every day.
In the end of 2000, Sharp Japan purchased a license of QPE itself and used it on their Sharp Zaurus SL-5000 PDA device. Trolltech had to turn the demo into a product and picked the name Qtopia for their product. After finishing the deadline to deliver their product Trolltech introduced rules to forbid over time. The evolution of Qtopia included targeting the increasing Linux Phone market.
The Qtopia product relied on the Qt Framework but the teams in Oslo and Brisbane were separated by their distance and their timezones. For many engineers there was no overlap in work time and no direct communication during work hours.

2.2.2 Porting work

The author of this report was contracted by the Taiwanese Openmoko Inc. to port the Qtopia libraries and applications from Qt/Embedded Linux to Qt/X11 and make it run on the Openmoko Freerunner [Ope08]. This work included replacing the QCOP IPC system of Qt/Embedded Linux, adding virtual keyboard infrastructure, dealing with issues related to Qt/X11. The biggest challenge was not the Qt/X11 port itself but to make it run well on the Openmoko Freerunner as there were many problems in the Qtopia product itself in addition to the device integration. This summary will look at problems that were introduced due intentional and unintentional code cloning.

The work was carried out on a tarball release of Qtopia and no version history was available to assist in helping to understand the creation of the clone.

3 Code Clones in Qtopia

This section will focus on some of the code clones encountered in the code. We will begin by introducing the issue that has lead to discover the problems, the attempted changes to the code to create a better abstraction and make a hypothesis about how this clone was created.

3.1 Activating a QWidget

3.1.1 Problem

The X11 Windowing System has the concept of a Window Manager. It is responsible for the policies, this includes - but it is not limited to - the positioning of windows, their stacking and the painting of the window decoration. The Window Manager is realized as a separate process and the behavior on the protocol level is specified by the Inter-Client Communication Conventions Manual (ICCM) of the X Consortium and updated in the NETWM of freedesktop.org.
Qt/Embedded Linux, on the other hand, does not support the concept of a window manager. The windows draw their decoration themselves and the window will be placed on the screen as requested by the window itself without any intervention.
On Qt/Embedded Linux it is enough to call the functions **raise()** and **activateWindow()** to make a window appear on top of all visible windows and activateWindow to make this window have mouse and keyboard focus. This has lead to the raise and activateWindow idiom as can be seen below.

raise, activate idiom

```
1 d->raise();
2 d->activateWindow();
```

On X11 the Window Manager is allowed to ignore both of these requests. This has lead to the problem that windows requiring input or confirmation appear below other windows.

3.1.2 Proposing an Abstraction

The author has attempted to solve this problem with the addition of a static helper function called **raiseAndActivateWindow(QWidget*)** and providing an alternative implementation for Qt/X11. The change in the version history can be seen below.

Adding abstraction

```
1 diff --git a/src/libraries/qtopia/qtopiaapplication.cpp b/src/
    libraries/qtopia/qtopiaapplication.cpp
2 index 9e1d445..5d9f541 100644
```

```
3  --- a/src/libraries/qtopia/qtopiaapplication.cpp
4  +++ b/src/libraries/qtopia/qtopiaapplication.cpp
5  @@ -3264,8 +3267,7 @@ bool QtopiaApplication::
       raiseAppropriateWindow()
6                       if ( w == d->lastraised )
7                           foundlast = true;
8                       if ( foundlast ) {
9  -                        w->raise();
10 -                        w->activateWindow();
11 +                        raiseAndActivateWindow(w);
12                          topsub = w;
13                      }
14                  }
15 @@ -3419,6 +3420,27 @@ static void markQtopiaWindow(QWidget *w)
16      w->setWindowFlags(Qt::FramelessWindowHint|w->windowFlags()
           );
17  }
18
19 +// Emulate a pager here. Qtopia 'knows' when to do these
       things...
20 +static void raiseAndActivateWindow(QWidget *w)
21 +{
22 +    w->raise();
23 +    w->activateWindow();
24 +
25 +    // Play pager now
26 +    static Atom netActiveWindow = XInternAtom(QX11Info::
       display(), "_NET_ACTIVE_WINDOW", False);
27 +
28 +    XEvent event;
29 +    event.xclient.type = ClientMessage;
30 +    event.xclient.display= QX11Info::display();
31 +    event.xclient.window = w->winId();
32 +    event.xclient.message_type = netActiveWindow;
33 +    event.xclient.format = 32;
34 +    event.xclient.data.l[0] = 2;
35 +    event.xclient.data.l[1] = QX11Info::appUserTime();
36 +    event.xclient.data.l[2] = 0;
37 +    XSendEvent(QX11Info::display(), QX11Info::appRootWindow(),
        False, SubstructureRedirectMask|SubstructureNotifyMask, &
       event);
38 +}
39 +
```

```
40  #else
41
42  static void markQtopiaWindow(QWidget *)
43 @@ -3426,6 +3448,12 @@ static void markQtopiaWindow(QWidget *)
44      // Nothing to do here for other platforms.
45  }
46
47 +static void raiseAndActivateWindow(QWidget *w)
48 +{
49 +    w->raise();
50 +    w->activateWindow();
51 +}
52 +
53  #endif
54
55  /*!
```

3.1.3 Analysis

The author of this summary considers this as an unintentional code clone. The structure of the code has already lead to handle all these window placements in the QPEApplication class and only these two calls were duplicated at seven different call sites. The problem was only created by two different concepts of window placing. For X11 the functionality needed to be implemented as a pager.

3.2 Failing SMS/Mail queries

3.2.1 Problem

The author of this summary was exporting the SMS functionality via the DBus IPC System to allow an external application to easily access the SMS Subsystem. Log messages *MailFolder query failed* were noticed by one of the developers. With the usage of the grep utility the source was identified to come from *qtopiamail/qmailstore.cpp*. The author opened the last hit and noticed that there were two possible reasons that this error was hit.

Duplicate logs

```
1 {
2     d->checkComparitors(key);
3
4         QString sql = "SELECT id FROM mailmessages";
5
6     if(!key.isEmpty())
```

```
7           sql += " WHERE " + d->buildWhereClause(key);
8
9       if (!sortKey.isEmpty())
10          sql += " " + d->buildOrderClause(sortKey);
11
12          QSqlQuery query = d->prepare(sql);
13
14          if(query.lastError().type() != QSqlError::NoError)
15                  qLog(Messaging) << "MailFolder query failed";
16
17      if (!key.isEmpty())
18          d->bindWhereData(key,query);
19
20          if(!d->execute(query))
21                  qLog(Messaging) << "MailFolder query failed";
22
23          QMailIdList results;
24
25          while(query.next())
26                  results.append(QMailId(query.value(0).
                       toULongLong()));
27
28          return results;
29  }
```

Changing the log messages in line 15 and 21 and recompiling was still triggering the old log messages in the test case. This was due to the code appearing in another method as well. In this code **QMailStore::queryFolders** and **QMailStore::queryMessages** share almost the same implemention including the bad indention of the code.

Duplicate logs

```
1   {
2       d->checkComparitors(key);
3
4           QString sql = "SELECT id FROM mailfolders";
5
6       if (!key.isEmpty())
7           sql += " WHERE " + d->buildWhereClause(key);
8
9       if (!sortKey.isEmpty())
10          sql += " " + d->buildOrderClause(sortKey);
11
12          QSqlQuery query = d->prepare(sql);
```

```
13
14          if(query.lastError().type() != QSqlError::NoError)
15                  qLog(Messaging) << "MailFolder query failed";
16
17      if(!key.isEmpty())
18          d->bindWhereData(key,query);
19
20          if(!d->execute(query))
21                  qLog(Messaging) << "MailFolder query failed";
22
23          QMailIdList results;
24
25          while(query.next())
26                  results.append(QMailId(query.value(0).
                        toULongLong()));
27
28          return results;
29  }
```

3.2.2 Abstraction

There are two ways to improve here. The first would be to add a special **qError** method to always include the function name, line-number and file in the log message to uniquely identify the place of the error. Futhermore, it should also print the kind of error that occurred.
To avoid this clone a C++ template or a preprocessor macro could be used and the table be passed as parameter or template specialisation.
It is also important to note that it is potentially dangerous to format a SQL statement can have security implications.

3.2.3 Analysis

The two methods appeared below each other, solve the same problem and have the same indention. Judging from the log messages, it appears that the **queryFolders** method was copied and renamed to **queryMessages**. The differences between these two methods are the different table and the static type of the two parameters passed to the function. These are not compatible with the type system of C++. The solution for these problems would be the introduction of a macro or the usage of a C++ template. It is hard to guess why the author of the code decided not to use any of these possibilities. He might have been inexperienced with C++ templates or for reasons in the environment not willing to invest the time for doing this abstraction.

3.3 Crash with the PIN Dialog

3.3.1 Problem

Qtopia allows to do the development on a normal desktop machine and use a Phone Simulator to simulate the GSM usage. When attempting to use the code on the target device and not entering the SIM Pin fast enough the application would run into an assertion and then quit. The assertion was inside the Qtopia libraries and it was not obvious why it was hit when running on the device. One of the differences between the desktop and the target device was that a different plugin was used to provide the telephony services. The assertion was due to the implementation of a **QSimInfo***, searching in the *src* directory only revealed one implementation with the name **QModemSimInfo**. Instrumenting the code did not show any effect on the device. After some search, a code clone was revealed in the *devices/ficgta01* folder. The diff below shows the difference in the code clone. The removed lines are the code that was missing from the cloned code, the added lines show code that is still in the clone or was added for it.

```
1  --- modem_qtopia.impl    2010-11-12 19:30:09.830729660 +0100
2  +++ modem_fic.impl  2010-11-12 19:29:41.581670870 +0100
3  @@ -4,7 +4,6 @@
4        QModemService *service;
5        QTimer *checkTimer;
6        int count;
7  -     bool simPinRequired;
8    };
9
10   QModemSimInfo::QModemSimInfo( QModemService *service )
11 @@ -16,14 +15,10 @@
12       d->checkTimer->setSingleShot( true );
13       connect( d->checkTimer, SIGNAL(timeout()), this, SLOT(
             requestIdentity()) );
14       d->count = 0;
15 -     d->simPinRequired = false;
16 +
17
18       // Perform an initial AT+CIMI request to get the SIM
             identity.
19       QTimer::singleShot( 0, this, SLOT(requestIdentity()) );
20 -
21 -     // Hook onto the posted event of the service to determine
22 -     // the current sim pin status
23 -     connect( service, SIGNAL(posted(QString)), this, SLOT(
       serviceItemPosted(QString)) );
24   }
```

```
25
26  QModemSimInfo::~QModemSimInfo()
27 @@ -63,27 +58,18 @@
28                 d->count++;
29             } else {
30                 d->count = 0;
31 -               // If not waiting for SIM pin to be entered by the
       user
32 -               if ( !d->simPinRequired ) {
33                     // post a message to modem service to stop SIM
                           PIN polling
34                     d->service->post( "simnotinserted" );
35                     emit notInserted();
36                 }
37 -           }
38 -           // If we got a definite "not inserted" error, then
       emit notInserted().
39 -           if ( result.resultCode() == QAtResult::SimNotInserted
       )
40 +           // If we got a definite "not inserted" or "sim failure
       " error,
41 +           // then emit notInserted().
42 +           if ( result.resultCode() == QAtResult::SimNotInserted
43 +             || result.resultCode() == QAtResult::SimFailure)
44                 emit notInserted();
45         }
46  }
47
48 -void QModemSimInfo::serviceItemPosted( const QString &item )
49 -{
50 -    if ( item == "simpinrequired" )
51 -        d->simPinRequired = true;
52 -    else if ( item == "simpinentered" )
53 -        d->simPinRequired = false;
54 -}
55 -
56  // Extract the identity information from the content of an AT+
       CIMI response.
57  // It is possible that we got multiple lines, including some
       unsolicited
58  // notifications from the modem that are not yet recognised.
       Skip over
```

3.3.2 Abstraction

The first approach was to remove the clone and use the standard **QModemSimInfo** implementation again. This was done under the assumption that all fixes have been migrated back to the standard implementation.
A developer from Trolltech highlighted that this assumption was wrong and that the addition of a clause in the if statement was not in the standard implementation. The second implementation added a quirk enum to the QModemSimInfo, a method to enable quirks, the extra condition and the configuration to enable the quirk.

Add Abstraction

```
1  diff --git a/devices/ficgta01/src/plugins/phonevendors/ficgta01
     /vendor_ficgta01.cpp b/devices/ficgta01/src/plugins/
     phonevendors/ficgta01/vendor_ficgta01.cpp
2  index 41819e5..f174393 100644
3  --- a/devices/ficgta01/src/plugins/phonevendors/ficgta01/
     vendor_ficgta01.cpp
4  +++ b/devices/ficgta01/src/plugins/phonevendors/ficgta01/
     vendor_ficgta01.cpp
5  @@ -560,8 +560,11 @@ void Ficgta01ModemService::initialize()
6       if ( !supports<QBandSelection>() )
7           addInterface( new Ficgta01BandSelection( this ) );
8
9  -    if ( !supports<QSimInfo>() )
10 -        addInterface( new QModemSimInfo( this ) );
11 +    if ( !supports<QSimInfo>() ) {
12 +        QModemSimInfo* simInfo = new QModemSimInfo( this );
13 +        simInfo->setSimNotInsertedReason( QModemSimInfo::
     Reason_SimFailure );
14 +        addInterface( simInfo );
15 +    }
16
17      if ( !callProvider() )
18          setCallProvider( new Ficgta01CallProvider( this ) );
19 diff --git a/src/libraries/qtopiaphonemodem/qmodemsiminfo.cpp b
     /src/libraries/qtopiaphonemodem/qmodemsiminfo.cpp
20 index 69c8ec1..9fb959f 100644
21 --- a/src/libraries/qtopiaphonemodem/qmodemsiminfo.cpp
22 +++ b/src/libraries/qtopiaphonemodem/qmodemsiminfo.cpp
23 @@ -51,6 +51,7 @@ public:
24      QTimer *checkTimer;
25      int count;
26      bool simPinRequired;
```

```
27 +     unsigned reasons : 1;
28  };
29
30  /*!
31 @@ -66,6 +67,7 @@ QModemSimInfo::QModemSimInfo( QModemService *
       service )
32       connect( d->checkTimer, SIGNAL(timeout()), this, SLOT(
           requestIdentity()) );
33       d->count = 0;
34       d->simPinRequired = false;
35 +     d->reasons = 0;
36
37       // Perform an initial AT+CIMI request to get the SIM
           identity.
38       QTimer::singleShot( 0, this, SLOT(requestIdentity()) );
39 @@ -140,7 +142,8 @@ void QModemSimInfo::cimi( bool ok, const
       QAtResult& result )
40                 }
41             }
42             // If we got a definite "not inserted" error, then
               emit notInserted().
43 -           if ( result.resultCode() == QAtResult::SimNotInserted
       )
44 +           if ( result.resultCode() == QAtResult::SimNotInserted
45 +               || (d->reasons & Reason_SimFailure && result.
       resultCode() == QAtResult::SimFailure))
46                 emit notInserted();
47       }
48  }
49 @@ -153,6 +156,11 @@ void QModemSimInfo::serviceItemPosted(
       const QString &item )
50           d->simPinRequired = false;
51  }
52
53 +void QModemSimInfo::setSimNotInsertedReason(enum
       SimNotInsertedReasons reason)
54 +{
55 +    d->reasons |= reason;
56 +}
57 +
58  // Extract the identity information from the content of an AT+
       CIMI response.
59  // It is possible that we got multiple lines, including some
```

```
    unsolicited
60  // notifications from the modem that are not yet recognised.
    Skip over
61 diff --git a/src/libraries/qtopiaphonemodem/qmodemsiminfo.h b/
   src/libraries/qtopiaphonemodem/qmodemsiminfo.h
62 index 34c556a..01b8100 100644
63 --- a/src/libraries/qtopiaphonemodem/qmodemsiminfo.h
64 +++ b/src/libraries/qtopiaphonemodem/qmodemsiminfo.h
65 @@ -32,9 +32,15 @@ class QTOPIAPHONEMODEM_EXPORT QModemSimInfo
   : public QSimInfo
66  {
67      Q_OBJECT
68  public:
69 +    enum SimNotInsertedReasons {
70 +        Reason_SimFailure = 0x1
71 +    };
72 +
73      explicit QModemSimInfo( QModemService *service );
74      ~QModemSimInfo();
75
76 +    void setSimNotInsertedReason(enum SimNotInsertedReasons);
77 +
78  protected slots:
79      void simInserted();
80      void simRemoved();
```

3.3.3 Analysis

Qtopia allowed the device integration to change the configuration, e.g. to provide more specialized implementations for certain interfaces. This system has been utilized to use the code clone instead of the standard implementation. The question is why this approach was chosen over inheritance with reimplementation or adding the additional clause like it was done above. One possible explanation is that the addition of new methods is not allowed during a minor release. This is a policy used on the Qt product to assure binary compability between minor releases and was applied by the Australian team to Qtopia. The question of why no tries were made to at least use inheritance and limit the code cloning to a single function can not be answered.
This issue shows the need for an approach to manage code clones. Somewhere in the development the new functinality was added to the standard implementation but the code clones were not updated.

3.4 Device Integration

3.4.1 Problem

This section is a bit different. There was no big issue that was caused by code cloning but after the experience of using the device integration system to avoid the creation of an abstraction, the author of this summary decided to actively look for code clones among the different supported devices. The intent of the device abstraction is to access the underlying hardware. This can include code for managing the sound system, driving the LEDs, querying the battery status. One more complex example would be to interface with a GSM modem and handle modem specific quirks.
The Linux kernel is already providing this abstraction for various hardware classes including the display, the backlight, the LEDs, the sound system, the battery status, the input event framework. The actual device integration should only map from the hardware name to the high level functionality of the framework, e.g. this could be to use the blue hardware LED for signalling incoming mail. In practice, there is no shared code to access any of these Linux kernel frameworks, instead there are copies of the code and the device names were changed in them.

3.4.2 Abstraction

The device abstraction could provide classes to access the ALSA sound system, the linux input event framework, backlight, LEDs and battery status. The maintainers of the device integration could then instantiate these classes with the proper hardware names for the device. This would greatly reduce the code clones in the *device* directory.

4 Code Clone Patterns

The previous section has highlighted the use of intentional and unintentional code clones in the Qtopia product. It has indicated that no system to manage these clones was in place and that this has lead to to runtime errors. This can be considered as a "bad smell" of software development [KG06] and one possible reaction to it would be to band the usage of intentional code clones.
Cory Kapser and Michael W. Godfrey argue that considering code cloning harmful should be considered harmful [KG06] and argue that some clones can have a positive impact on the software system. In their work, they have introduced the concept of clone patterns and for each identified pattern, they dicuss the possible benefit, the problems, possible long term issues and a proposal on how to manage these clones. They have identified eight patterns and divided them into different groups *Forking*, *Templating* and *Customization* based on the high level motivation. The following section will shortly introduce these groups, the patterns and will discuss how these patterns were used in Qtopia.

4.1 Forking

The *forking* pattern can be useful when an existing solution can be used to bootstrap the work of something closely related and the cost of adding an abstraction to the code would be too high or would risk breaking the already existing solution. One key is the assumption that the base and the fork will evolve differently. This approach was used in the Linux kernel with the creation of ext3 and ext4, in both cases the filesystem of the previous generation was forked and then evolved into a new version.
The forking patterns are *hardware variation* to support a new version of the hardware, *platform variation* to support a new platform or operating system and *experimental variation* to experiment with the code without breaking existing users.

4.1.1 Qtopia and Forking

The *QSimInfo* issue could be considered an application of the hardware variation pattern. With the help of the device integration a fork of the *QModemSimInfo* class was created. The question is if there was an intent to make the clones evolve differently.

4.2 Templating

For the *templating* pattern there is already an existing solution that can be adapted to satisfy the new desired behavior. In most of these cases, the abstraction needed would be the right parametrization and one of their examples is to make a method that is working with *floats* work on *shorts* as well. The identified patterns include *boiler-plating due to language in-expressiveness*, e.g where code reuse is not easily possible, *API Library protocols*, e.g the code needed to setup a socket in C or *general language or algorithmic idioms*, e.g checking for memory allocation failures in C. In general, all of the clones should evolve in a similar way.
One of the dangers is that e.g. with a new version of the language or the API many clones need to be updated and finding all of them could be difficult.

4.2.1 Qtopia and Templating

There are two examples that match the criteria of templating. The window placement issue could be considered an occurence of the API Library protocols pattern and the *QMailStore* issue could be either the general language or algorithmic idiom or the *boiler-plating due to language in-expressiveness* due the C++ template support being difficult to use.

4.3 Customization

Customization is used in case there is already code that is solving a very similar problem but for various reasons, e.g. being afraid about the impact on system stability, someone

else owning this method, not being able to change the existing code, the code can not be changed to handle the additional requirements. The two identified patterns are *bug workaround* and *replicate and specialize*. In the case of the a workaround, one should attempt to remove the workaround when the original bug is resolved, in case of *replicate and specialize* one should try to remove the original code in case the new method is able to handle both required behaviors.

4.3.1 Qtopia and Customization

The cloning of the *QModemSimInfo* class can not be considered an occurence of the *bug workaround* as the bug is with the used GSM modem and not with the code itself, but it could be considered a usage of the *replicate and specialize* pattern due to the added conditionals in the code. The clone in the *QMailStore* class could be considered a case of *replicate and specialize* due the changing of the method parameters and the changing of the table name to query.

5 Summary

The developers behind Qtopia have used unintentional and intentional code cloning throughout the development. In some cases, the intentional clones were not updated and the outdated versions have caused runtime failures. When these code clones are in a *device* specific folder, they are creating the issue for the user of the Qtopia product and without access to the source control history, it is harder to find what needs to be changed in the clones. It appears crucial to either avoid code clones or to have a system in place that will identify them and warn when these clones become outdated.
In the case of Qtopia the shown code clones were harmful, Kapser in association with Godfrey argues that code clones can be beneficial as well and have found evidence in the Linux source code. What other factors are important to have beneficial code clones and avoid the harmful ones? Is this a matter of tool support? Is this solely an attitude of the team developing the system?

References

[KG04] Cory Kapser and Michael W. Godfrey. Aiding comprehension of cloning through categorization. In *Proceedings of the Principles of Software Evolution, 7th International Workshop*, pages 85–94, Washington, DC, USA, 2004. IEEE Computer Society.

[KG06] Cory Kapser and Michael W. Godfrey. "cloning considered harmful" considered harmful. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pages 19–28, Washington, DC, USA, 2006. IEEE Computer Society.

[Ope08] Openmoko Inc. *http://wiki.openmoko.org/wiki/QtopiaOnX11*, 2008. Wiki pointing to the work.

[Tro00] Trolltech ASA. *http://web.archive.org/web/20000511204217/www.trolltech.com/*, 2000. Link to the Announcement from the Web Archive.