IBM Innovate 2012
IBM Rational Rhapsody and Qt Integration for Model-Based GUI Development

Giulio Santoli
Client Technical Professional, IBM Rational
giulio_santoli@it.ibm.com
Session RG-1258

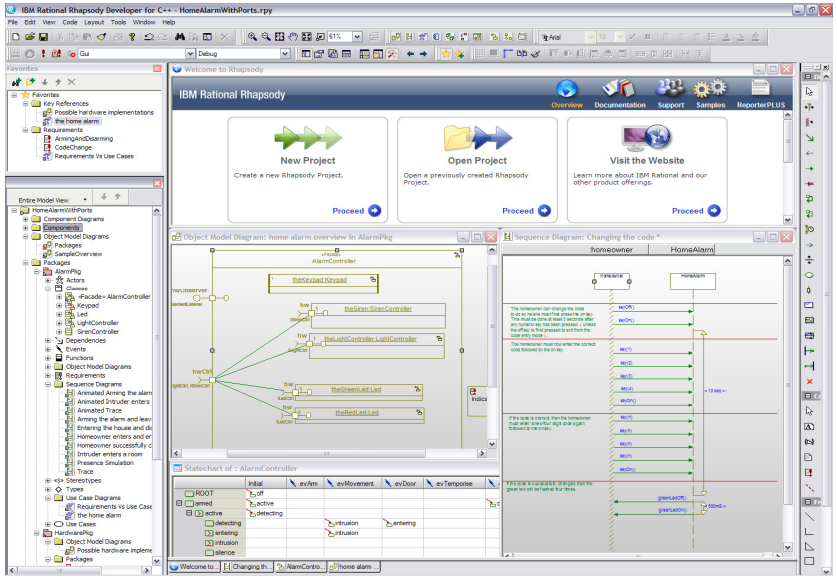
IBM Software
Innovate2012
The Premier Event for Software and Systems Innovation

Next NOW!

Agenda

1. Model-Driven Development with IBM Rational Rhapsody
2. Nokia Qt Framework Overview
3. Rhapsody and Qt Integration Issues
4. Rhapsody Qt Profile Overview
5. Demo
6. Summary

System and Software Engineering with IBM Rational Rhapsody



"Using Rhapsody software improves the quality of the application software that is integral to the series hydraulic hybrid system development process."

Steve Zielinski, Eaton chief engineer for software

Capabilities

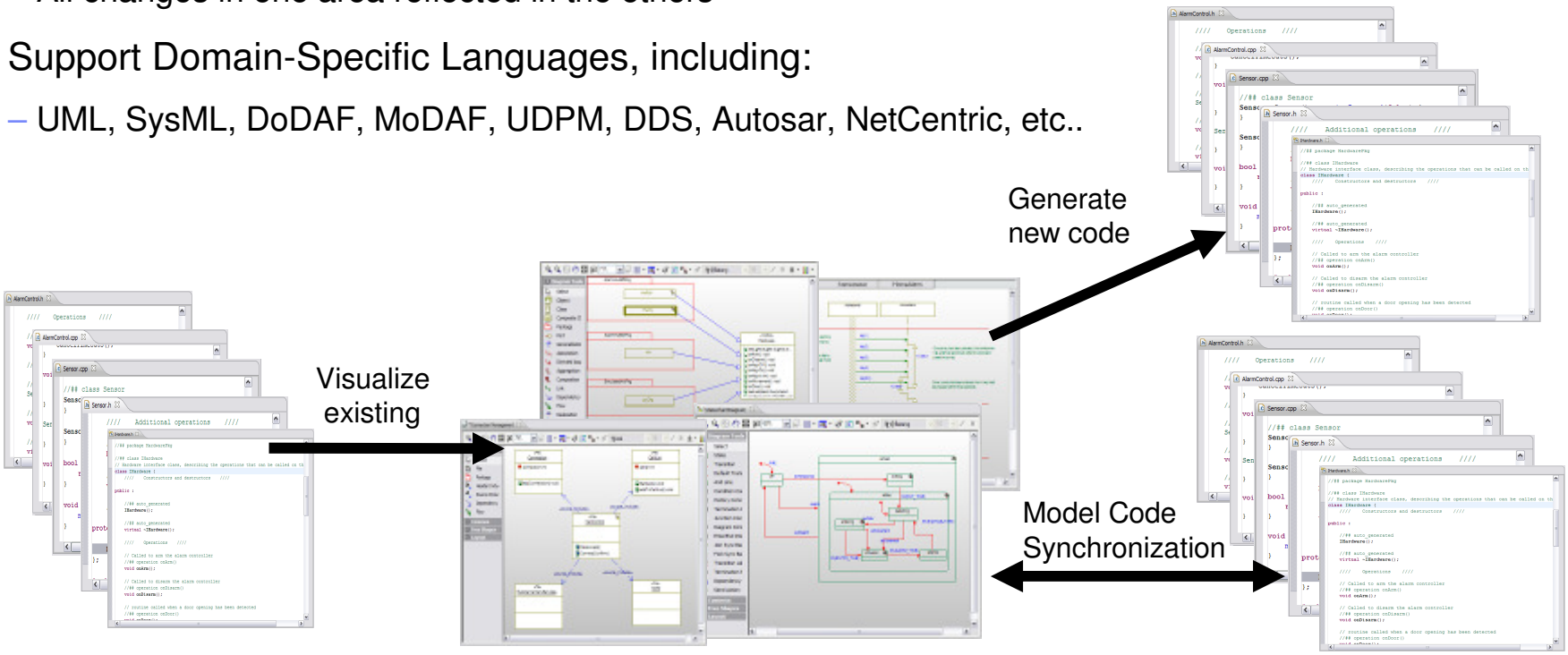
- Requirements-driven analysis and design for technical, embedded or real-time solutions, including those based on *multi-core* architectures
- Rapid design validation and verification with frequent simulation and testing
- Development and deployment of *complete* C, C++, C#, Java and Ada applications

Benefits

- **Build the right product** through non-ambiguous communication and frequent collaboration
- **Eliminate defects early** and increase quality by continually testing the design
- **Reduce development time** by automatically generating applications and documentation
- **Re-use and adapt existing technology** through reverse engineering and product line engineering

Model-Driven Development with IBM Rational Rhapsody

- Build efficient software that exactly matches the design intent
  - Develop C, C++, C#, Java, and Ada applications
- Maintain automated synchronization between model and code
  - Work simultaneously with architecture, software and target
  - All changes in one area reflected in the others
- Support Domain-Specific Languages, including:
  - UML, SysML, DoDAF, MoDAF, UDPM, DDS, Autosar, NetCentric, etc..



Software Model Execution in Animation Mode

Model execution enables iterative development

Requirements test through execution

Correct specification hand-off to software

Nokia Qt Framework Overview

- Nokia Qt is a **cross-platform application** and UI framework with APIs for C++ programming and for rapid UI creation of:
  - mobile applications (Symbian phones and the Nokia N9 smartphone)
  - desktop applications (Microsoft Windows, Mac OS X, and Linux)
- The Qt Framework is based on **modular C++ class library**, including UI, Multithreading, Networking, Multimedia, XML, Database connection and so on.
- The Qt SDK combines the Qt framework with tools such as an IDE, an UI Designer, a Simulator and some other tools.

http://qt.nokia.com/

Rhapsody and Qt Integration: Technical Issues (1/2)

- Qt extends the C++ syntax by introducing some keywords and macros (such as “signal”, “slot”, “Q_OBJECT”) that are handled by the Qt pre-processor at compilation time.
- By default, these keywords are not compatible with the Rhapsody C++ parser and prevents Rhapsody round-trip feature.
- Here follows an example that shows these Qt keywords:

```
#include <QObject>

class Counter : public QObject {

  Q_OBJECT

  public:
      Counter()          { m_value = 0;     }
      int value() const  { return m_value;  }

  public slots:
      void setValue(int value);

  signals:
      void valueChanged(int newValue);

  private:
      int m_value;
};
```

The **Object** class must be the first in the base classes list.

The **Q_OBJECT** macro is used to enable **Qt meta-object features**, such as dynamic properties, signals, and slots.

Signals are emitted by an object when its internal state has changed in some way that might be interesting to the object's client or owner.
Signal implementation is generated by the Qt preprocessor must not be implemented in the .cpp file.

Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.
A slot is automatically called when a signal connected to it is emitted.

Rhapsody and Qt Integration: Technical Issues (2/2)

- Qt requires the **main thread** to be the primary "**GUI thread**“ as Qt widgets and several related classes don't work in secondary threads. This usually conflicts with the standard Rhapsody configuration, as the main thread is automatically generated as an **OXF thread**.

- To cope Rhapsody with the **Qt meta-object system**, it is needed:
  - Update Rhapsody code generator though Rhapsody Simplifiers APIs
  - Customize Rhapsody properties to handle Qt-specific keywords
  - Customize Rhapsody autogenerated Main to correctly the Qt main thread
  - Customize Rhapsody autogenerated Makefile to invoke Qt command line utilities

Solution: Rhapsody Qt Profile

- Rhapsody Qt Profile is a **custom profile** for Rhapsody in C++ based on stereotype customization and a Java plug-in invoked at code generation time.
- This profile is **not part of the official Rhapsody installation package**.
- It allows you to develop **Qt-compatible code** that also takes advantage of Rhapsody capabilities, such as state-machines with either synchronous or asynchronous events.
- The profile **automatically generates** the Qt project file and customize it accordingly.
- The current implementation supports the following:
  - Qt 4.7.3 for Windows (VS 2008 only) and Linux, with and without Animation (no Tracing).
  - All the .ui files are supposed to be stored in the “gui” directory in the Rhapsody project.
  - No support for Qt message catalogs and resources yet.
  - Only base Qt classes are defined in the profile (QApplication, QObject and QWidget).
  - All the slots are public, regardless their visibility in the model.
  - Qt Mobile Framework is not supported.

Rhapsody Qt Profile Stereotypes

<<QTObject>>

You can use this stereotype to flag those **Classes** that needs to be Qt-compliant, with the Q_OBJECT macro, signals and slots.

<<QTSignal>>

You can use this stereotype to flag those **Operations** that are Qt Signals. They will be declared in the specification file (.h). Use it only for operations belonging to classes stereotyped as <<OTObject>>.

<<QTSlot>>

You can use this stereotype to flag those **Operations, Triggered Operations** or **Receptions** that are Qt Slots. Use it only for operations belonging to classes stereotyped as <<OTObject>>.

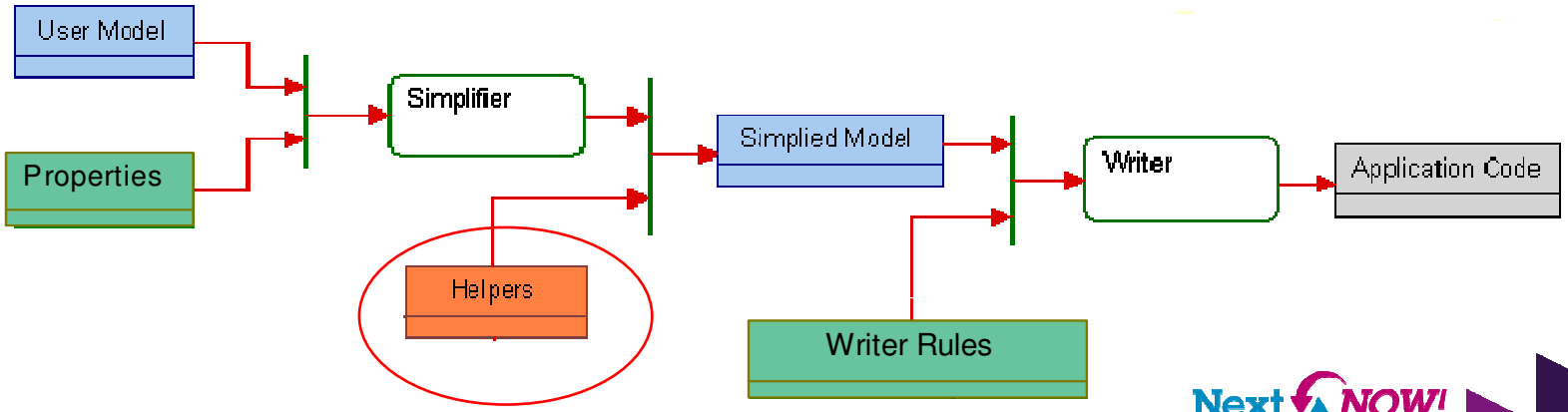
<<QTConfiguration>>

You can use this stereotype to flag those **Configurations** that you want to generate Qt-related programs. The default Makefile and Main are generated accordingly.

Custom Simplifiers in Rhapsody Qt Profile

- Rhapsody Qt Profile uses **Custom Simplifiers** to "hook" pre-defined extension points at code generation time.
- Custom simplifiers manipulate the code model using standard Rhapsody APIs.
- For each element the simplify property can specify:
  - *Default*: follow out-of-the-box simplification
  - *None*: suppress the simplification
  - *Copy*: copy the application element to the code model as is
  - *User*: replace out of the box one with a new user defined one
  - *User Post Default*: invoke a custom simplifier after the out of the box one

C_CG	
Attribute	
Simplify	Default
Class	
Simplify	Default
Configuration	
SimplifyMainFiles	Default
SimplifyMakeFile	Default
Event	
Simplify	Default
Generalization	
Simplify	Default
Operation	
Simplify	Default



Rhapsody Qt Profile Usage Overview

- To create you classes and set the <<QTObject>> stereotype to those of them you want to be Qt-enabled.
- When you stereotype a class with <<QTObject>> do the following:
  - Create a generalization dependency to QObject or QWidget classes provided by the profile. If you want to generalize a different class, copy one of the above ones in your project and rename at your will.
  - Create a constructor that initializes the parent Qt class, i.e. MyWindget(QWidget*parent)
  - Specify the includes you need in the Properties, i.e. <QWidget> or just <QtGui>
  - Apply the <<QTSignal>> and <<QTSlot>> stereotypes accordingly
- Apply the <<QTConfiguration>> stereotype to the Rhapsdody configuration you want to generate the Qt-enabled program.
- If you need user interface files (.ui) generated with Qt Designer, put them in the “gui” directory in your project: they will be automatically included in the build. Just Remember to include their correspondent header file names, as they are supposed to be generated during the Qt buil process.

Code Example for a simple Reactive Class

- Generated code example with the Rhapsody Qt Profile:

```
//## class MyCountdownGUI
class MyCountdownGUI : public QWidget, public OMReactive, private Ui::QtCountdown {

    Q_OBJECT

    ////    Slots    ////
    public:
        Q_SLOT void tgReset();
        Q_SLOT void tgStart();
        Q_SLOT void tgTimeout();
    ////    Constructors and destructors    ////

public :

    //## operation MyCountdownGUI(QWidget)
    MyCountdownGUI(QWidget* parent = 0, IOxfActive* theActiveContext = 0);

    ////    Operations    ////

    //## operation startTimer(long)
    void startTimer(long time);

    ////    Additional operations    ////

    //## auto_generated
    virtual bool startBehavior();

protected :

    //## auto_generated
    void initStatechart();

    ////    Attributes    ////

    int counter;        //## attribute counter
```

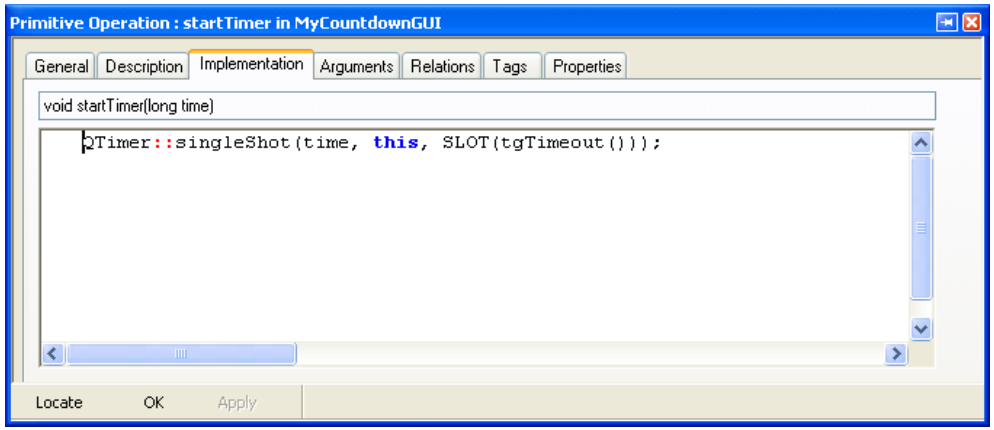
Rhapsody base class for classes with state machines

QTProfile generated slots and macros

This method starts the Rhapsody state machine implementation

Asynchronous or Synchronous Slots in Rhapsody State Machine (1/2)

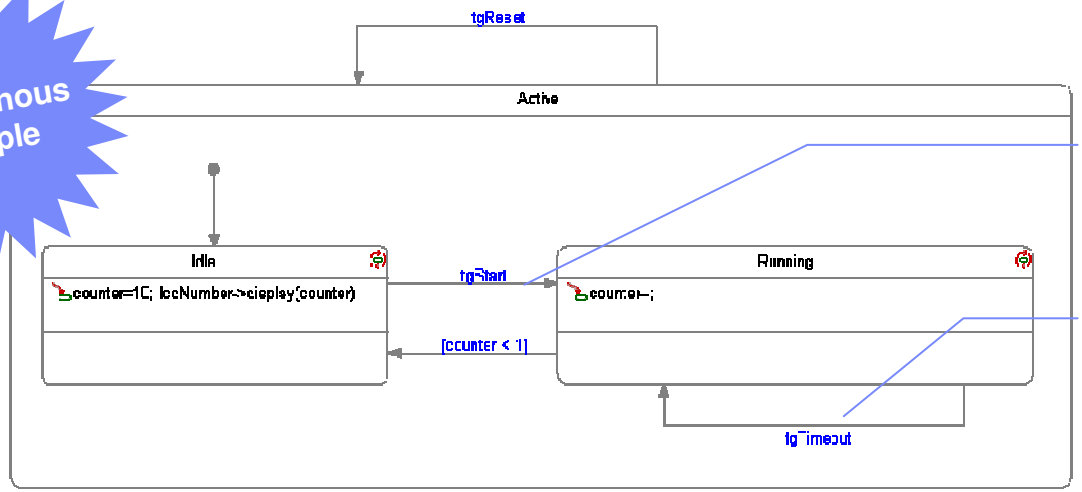
- **Synchronous State Machines**
  - In the Synchronous implementation, if the Class in Rhapsody is not Active, the Main Thread is the Qt GUI Thread, it's not possible to use Rhapsody Asynchronous Events e and no OXF Threads are running.
  - In this configuration, it's not possible to use the Rhapsody timer but it's needed to use the Qt one. Here follows a simple example:



- **Asynchronous State Machines**
  - If the Class in Rhapsody is Active, it runs its own OXF Thread and it's possible to use Asynchronous Events (Receptions) and the Rhapsody timeout.

Asynchronous or Synchronous Slots in Rhapsody State Machine (2/2)

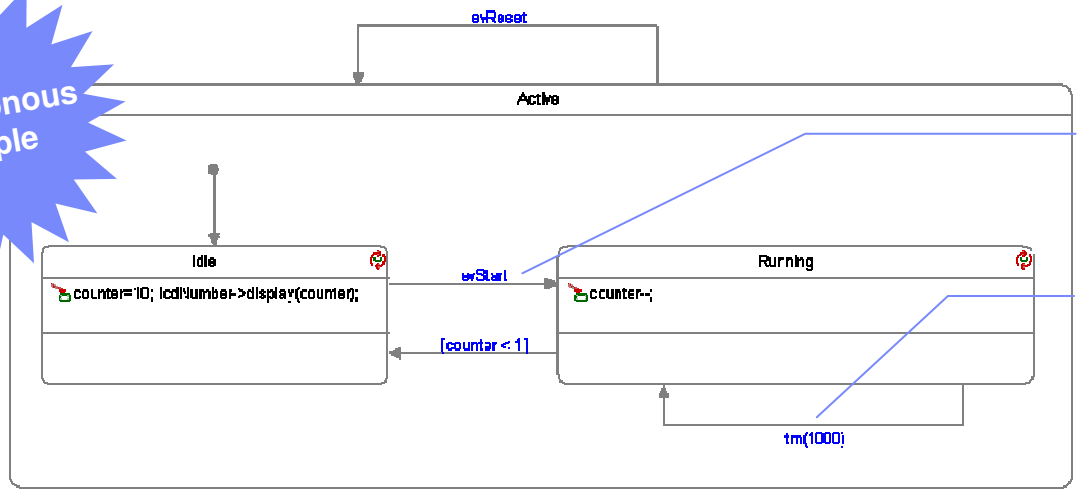
Synchronous Example



In synchronous mode, only **Triggered Operations** can be used.

In synchronous mode, it is needed to use **Qt timeout**.

Asynchronous Example

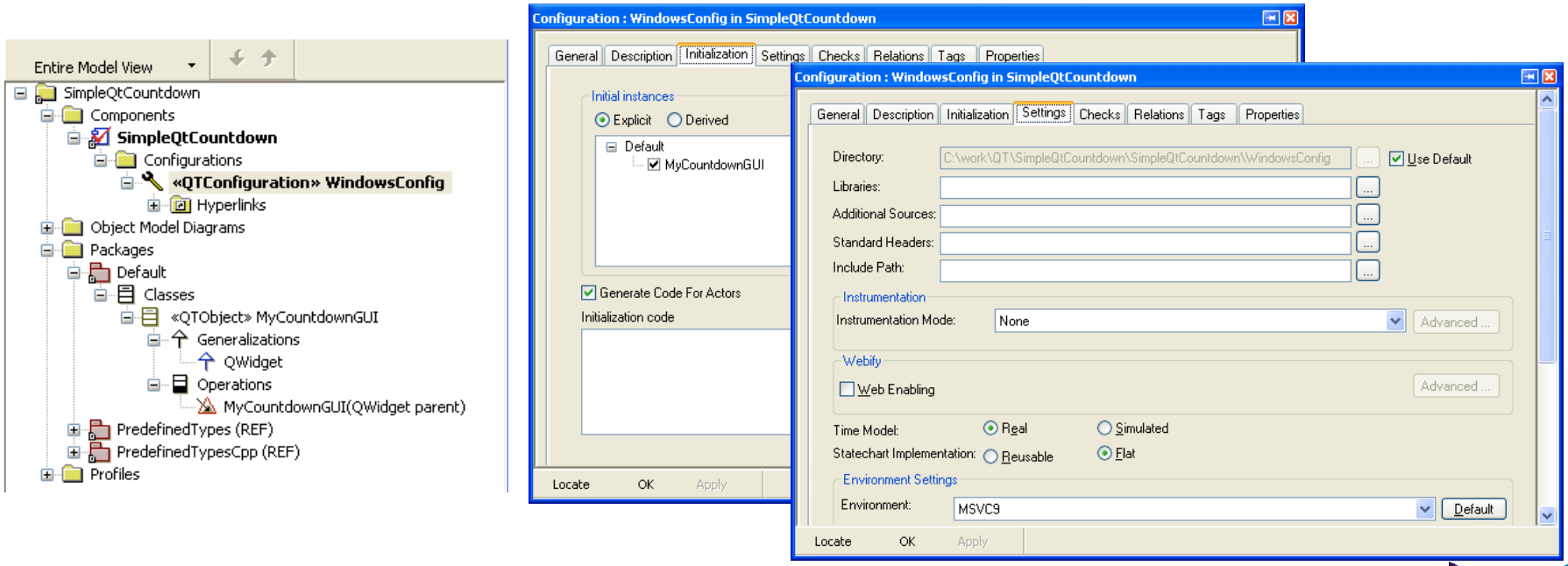


In asynchronous mode, both **Receptions** and **Triggered Operations** can be used.

In asynchronous mode, it possible to use OXF timeout.

QTProfile Main Class v.s. Default Main Class (1/2)

- The QTProfile introduces some differences to the generated main class, to make it compatible with Qt and make the main thread the **primary GUI thread**.
- Here follows the differences with the default standard main class:
  - It automatically creates a *QApplication* object and starts it: this makes the main thread the Qt GUI thread too.
  - It does not start the OXF main loop: if you want to use Rhapsody asynchronous events you need at least an active class in your model.



QTProfile Main Class v.s. Default Main Class (2/2)

MainDefaultComponent.cpp [Beyond Compare]

File Edit Search View Tools Help

C:\...\DefaultConfig\MainDefaultComponent.cpp 7/27/2011 5:27:14 PM

```
#include <QApplication>
//## auto_generated
#include "MainDefaultComponent.h"
//## auto_generated
#include "Hello.h"
int main(int argc, char* argv[]) {
    int status = 0;
    QApplication app(argc, argv);
    if(OXF::initialize())
        {
            Hello * p_Hello;
            p_Hello = new Hello;
            //#[ configuration DefaultComponent::DefaultConfig
            //#]
            app.exec();

            delete p_Hello;
            status = 0;
        }
    else
        {
            status = 1;
        }
    return status;
}

/*********************************************************************
    File Path    : DefaultComponent\DefaultConfig\MainDefaultCompor
```

With the Qt Profile

C:\...\DefaultConfig\MainDefaultComponent.cpp 7/27/2011 7:46:36 PM

```
//## auto_generated
#include "MainDefaultComponent.h"
//## auto_generated
#include "Hello.h"
int main(int argc, char* argv[]) {
    int status = 0;

    if(OXF::initialize())
        {
            Hello * p_Hello;
            p_Hello = new Hello;
            //#[ configuration DefaultComponent::DefaultConfig
            //#]
            OXF::start();

            delete p_Hello;
            status = 0;
        }
    else
        {
            status = 1;
        }
    return status;
}

/*********************************************************************
    File Path    : DefaultComponent\DefaultConfig\MainDefaultCompor
```

Without the Qt Profile

3 Section(s) Different Load time: 0.00 sec

www.ibm.com/software/rational

Summary

- IBM Rational Rhapsody is a great tool for **Model-Drive Development** of C, C++, C#, Java and Ada applications.
- Nokia Qt is a well known and adopted C++ Framework to build **cross-platform** GUI applications.
- Thanks to **Rhapsody flexibility**, it has been possible to develop a custom Rhapsody profile to benefit of both Rhapsody and Qt and leverage Model-Driven Development for GUI Applications development.

www.ibm.com/software/rational

Daily iPod Touch giveaway

- Complete your session surveys online each day at a conference kiosk or on your Innovate 2012 Portal!
- Each day that you complete all of that day's session surveys, your name will be entered to win the daily IPOD touch!
- On Wednesday be sure to complete your full conference evaluation to receive your free conference t-shirt!

iPod Touch giveaway.

sponsored by AllianceTech

Acknowledgements and disclaimers

Availability: References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

www.ibm.com/software/rational