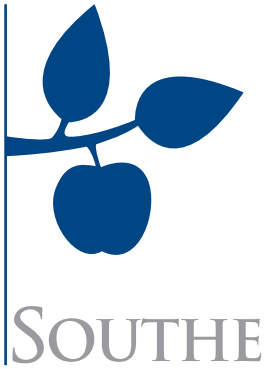
Bachelor



University of Southern Denmark

Extended user interface and simplification of the interactions with RobWork

Anders Ellinge
aelli14@student.sdu.dk

Mathias Elbæk Gregersen
magre14@student.sdu.dk

This page is intentionally left blank

Informations

Institution:	University of Southern Denmark – The Technical Faculty
Course code:	RB-BAP6-U1
Project title:	Extended user interface and simplification of the interactions with RobWork
Supervisors:	Lars-Peter Ellekilde, lpe@mmmi.sdu.dk Thomas Nicky Thulesen, tnt@mmmi.sdu.dk
Project period:	1. of february – Sometime in 2017
Number of pages:	9 pages
Character count:	48,000 (excl. references, appendices etc.)
Signatures:	

Anders Ellinge
aelli14@student.sdu.dk

Mathias Elbæk Gregersen
magre14@student.sdu.dk

Abstract

Your abstract goes here... ...

Contents

Foreword **II**

List of Abbreviations and Symbols **III**

1 Introduction **1**

2 Chapter 1 **2**
2.1 We write shit . . . 2

3 RobWork library and its functionalities **3**
3.1 WorkCells . . . 3
3.2 Frames . . . 3
3.3 Objects . . . 3
3.4 Devices . . . 4
3.5 States and State Structures . . . 5
3.6 Namespacing in RobWork . . . 5
3.7 Typical use of RobWorkStudio and WorkCell-files . . . 5

4 Qt **6**
4.1 Qt Class Hierarchy and Object Model . . . 6
4.1.1 The Meta-Object System . . . 6
4.2 QWidgets . . . 7
4.2.1 QMainWindow . . . 7
4.3 QLayout . . . 8
4.4 Signal and Slots . . . 8
4.4.1 Difference between events and Signal/Slots . . . 8
4.5 Subclassing . . . 8
4.6 Plug-in . . . 8

5 Describe functionality the user needs, and sort in order of importance **9**

Foreword

- The foreword page should describe the aim of the thesis, and its various research stages, and present the partners, funding and circumstances involved in the thesis project. The forewords should also include words of gratitude, addressed to people who have been incremental in your thesis-writing process. The supervisor, the second examiner, and the technical supervisor can be mentioned as well.

List of Abbreviations and Symbols

RW RobWork
RWS RobWorkStudio
moc Meta-Object Compiler

1 Introduction

- Describe the background and motivation of your thesis
- Describe in detail the objectives of your thesis (i.e. what is your aim to achieve), and on what basis the scope of your research area has been chosen
- At the end of the introduction, you might want to give an overview about the structure of your thesis

RobWork is a collection of C++ libraries for simulation and control of robot systems. RobWork is used for research and education as well as for practical robot applications. It has been noted that there exists different tedious processes of editing the environment of the RobWork simulation, involving reconfiguring files, unintuitive user interaction or reloading software. If some of these processes can be shortened or made more intuitive, the user experience of RobWork would greatly benefit from it. An example of an above-mentioned process: Adding a specific object to the environment of a simulated robot requires altering an .XML file and reloading RobWork. To specify goals for this bachelor project, the following features to implement is stated in order of importance:

- A user interface to initialize and load objects into the environment of RobWork with a simple drag and drop-like feature.
- Changing the configurations of the simulated robot in an intuitive matter, involving cursor interaction pulling the robot into the desired position.
- Adding tools or end effector to a simulated robot in a drag and drop-like feature. The tool should then be able to snap unto various parts of the robot.

With skills and experience within software development, object oriented C++ and various programming oriented skills, the authors of this project will try to solve and implement the goals stated. The first step of the project would be to become familiar with RobWork and learn how the source code works. The next step is to design the software for the first feature and disclose a possible solution. The third step is to implement the first feature and test it. The success criteria for the project is a finished, tested and working feature for RobWork of the before mentioned first goal set for this project. If this criterion is met early, the project would continue in similar manner with the second and third feature.

2 Chapter 1

2.1 We write shit

shit shit shit shit

3 RobWork library and its functionalities

This chapter is a general introduction to the RobWork library and the most commonly used data structures and functionalities within.

3.1 WorkCells

A WorkCell is the basis structure in RobWork. The WorkCell can be thought of as a box containing all of the other building blocks and information needed to represent an environment. The WorkCell most commonly contains Frames, Objects, Devices which are used to represent the different items in the environment. The WorkCell also contains a State Structure used to describe how the items in the environment are related.

3.2 Frames

One of the most common data structures from the RobWork library is a frame. A frame is the basic building block in the RobWork library, representing (in the case of RobWork) a local 3D cartesian coordinate system.

In RobWork frames come in 3 different types: fixed frames, moveable frames and joints. Fixed frames are frames that have a constant transform relative to the parent frame.
Movable frames are frames which transform can be freely changed.
Joints are frames that can be assigned values for position, velocity limits and acceleration limits. This type of frame is usually used for devices. Joints can be further divided into 4 subtypes: prismatic joint, revolute joint, dependent joint and virtual joint.

Prismatic joints are joints which motion is linear along a constant direction. Thinking of a pneumatic piston can be an intuitive way of thinking about prismatic joints.

Revolute joints are joints which motion is based on a rotation around a single axis. Thinking of hinges can be an intuitive way of understanding revolute frames.

Dependent joins refer to joints which transform depends on on or multiple other joints. Dependant joins can also be divided into 2 subtypes, dependent prismatic joints and dependent revolute joints, adding the motion specification of the prismatic joint and revolute joint previously mentioned.

Virtual joints BLAH BLAH BLAH.

Frames in a WorkCell are required to have a parent and are given a unique name so that no frames can be confused for another. Only one frame in the WorkCell has no parent. This frame is called WORLD and is created when the WorkCell is constructed. The WORLD frame can be seen as the global 3D cartesian coordinate system for the WorkCell.

3.3 Objects

Contrary to frames which represents the relationships in the environment, objects represents physical things in the scene. Also contrary to frames is that the name of an object does not have to be unique.

In order for objects to get a relationship to the environment it is placed in, it has to be associated to a frame. This frame is called the base frame of the object. An object can be associated to multiple frames but only have one base frame.

An object consists of two important elements, a geometry and a model.
A geometry is used to represent the actual geometry of the object. The geometry can be scaled and transformed to allow for modifications. In order to perform the transform the geometry need a reference frame, hence the geometry is attached to a frame. RobWork is capable of creating simple geometries like spheres, boxes and cylinders, however it is also possible to import complex geometries. Geometries are also being used for the collision detection provided in RobWork.
A model is a graphical representation of the object. Models consists of geometries, materials, colors and texture information as well. It is also possible to apply a transform to and get the transform of a model.
Usually when an object is created, a geometry is created for collision detection and a model is created to visually represent the object in a viewer(RobWorkStudio etc.).

There are two types of objects in RobWork, rigid objects and deformable objects. Rigid objects are objects which geometry does not change. Rigid objects can also posses information about inertia and mass. A deformable object however has the ability to alter its geometry via control nodes.

3.4 Devices

A device can be considered (in the case of a joint device) to be a collection of joints and objects denoting the setup of a device e.g. the FANUC LRM200 robotic arm. The device also contains the configurations for the joints contained in the device. these configurations are contained in a single configuration vector, making it easy to control the device. it is also possible to get and set the bounds, velocity limits and acceleration limits for the joints.

Devices can be of 3 different types: joint device, mobile device and SE3 device. Mobile devices are devices that is differentially controlled e.g. a robot rover. SE3 devices are devices that BLAH BLAH BLAH. Joint devices are that consists of moving joints much like the previously mentioned FANUC LRM200 robotic arm.

Joint devices can be of 5 types: Composite device, composite joint device, parallel device, serial device and tree device.
Serial devices are the simplest form of device since it consist of joints set in serial to each other. Many simple robotic arms like the before mentioned FANUC LRM200 are serial devices.
Tree devices are devices which joints follow a tree structure, meaning that a joint can have multiple children but only one parent joint. This also means that a tree device must have more than one end effectors. This type of device is typically seen in dexterous hands.
Parallel devices are devices that at some point in the structure of the is created a circle e.g. a joint goes to two joints that then both go to the same joint. The two middle joints are said to be in parallel to each other and are called parallel legs in RobWork. Parallel legs can consist of multiple joints as well as just one.
Composite devices and composite joint devices are devices that are constructed from a series of other devices. The devices in the composite device may not share joints. Just like tree devices, composite devices can have multiple end effectors. However unlike the tree

devices, composite devices does not require the path to the end effectors to have a common base.

3.5 States and State Structures

In RobWork the structure of the frames are represented in trough the State Structure. The State Structure also holds the state of the individual frames. The State object is a collection of the states of all the frames contained in the State Structure. This is done as a kinematic tree.

3.6 Namespacing in RobWork

3.7 Typical use of RobWorkStudio and WorkCell-files

4 Qt

Qt, pronounced "cute", is an open source cross-platform framework, mostly used for GUI(graphical user interface) programming. Qt has an easy to (re)use API(application programming interface), which in return gives high developer productivity. QT is C++ class library, hence new developers using Qt should have some understanding of C++. This chapter introduces terminologies used in Qt, and tries to give some general insight to how Qt operates and works regarding GUI development.

4.1 Qt Class Hierarchy and Object Model

Qt broadly uses inheritance to create subclasses of instances in a natural way. QObject is the most basic class in Qt, see FIGURE. A lot of classes inherit from QObject, like QWidget, which is the base of all user interface objects. C++ offers efficient runtime for a object oriented scheme, but lacks in regard to flexibility due to the static nature of the C++ Object Model. Qt has implemented the QObject as the hearth of the Qt Object Model, which preserve the efficient runtime while also offering more flexibility for the GUI domain. The Qt Object Model is implemented with standard C++ techniques. Some of the features that the Qt Object Model adds are e.g.

- Inter-Object Communication called Signal and Slots in the Qt Object Model. This topic is expanded upon in section 4.4.
- Object Trees which structures ownership of objects in a natural fashion. This topic is expanded upon in section 4.2.

4.1.1 The Meta-Object System

Due to the Qt Object Model the Meta-Object System was in turn created, which on the bottom line provides the Signal and Slots for inter-object communication and other features from the the QT Object Model. The Meta-Object System is based on three things: **a)** the QObject class **b)** the Q_OBJECT macro and **c)** the Meta-Object compiler(moc). Each QObject or subclass of QObject has an instance of QMetaObject created to hold the meta-data information, e.g. the name of the class or the class's meta-methods(signal, slots and other member functions). The Q_OBJECT macro helps and defines the meta data for the moc at compile time. Please refer to figure 1 to see influence of the Meta-Object System in compile time.

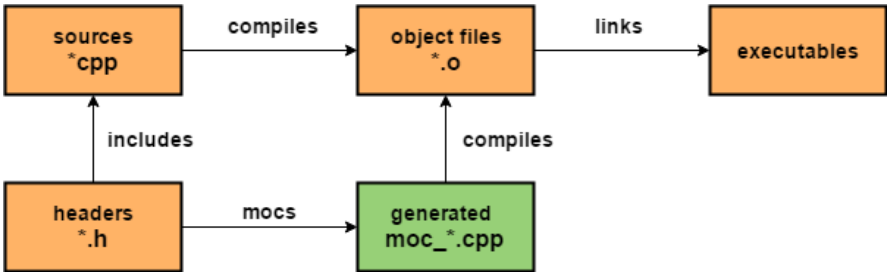


Figure 1: This figure shows how the the Meta-Object System is integrated at compile time. The yellow boxes indicate the normal C++ compiling procedure, whereas the green box is the added moc, which is compiled into the object files

4.2 QWidgets

QWidget is the base of all user interface objects(buttons, menus etc.). QWidget handles all events from the system the application is running on, i.e. In Qt, events are QEvent objects which is created upon outside activity (like a click on a mouse). Subclasses of QEvent involve more parameters to characterize a certain event, e.g. mousePressEvent(QMouseEvent* event). The event object is then sent to a specific QWidget object (maybe a button) and the QWidget handles the event with the according event handler.

As mentioned in 4.1 ownership of objects is structured in a tree, this means that a QWidget can have QWidget's within it self, see figure 2. A QWidget with no parent is called a top-level widget, which means the QWidget is an independent window. An instance like QWidget subclass QDialog(a pop up dialog window) is a top-level widget. QDialog can be instantiated with a parent, but the QDialog is still a top-level widget in this case, though the position of the dialog window is now centred relative to the parent.

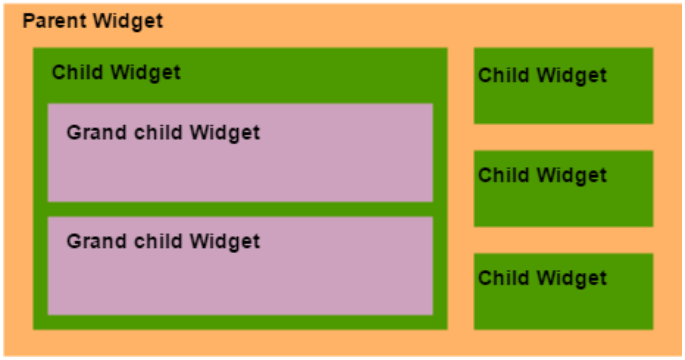


Figure 2: Qt structures ownership of objects in a parent child relationship. The diagram shows a parent widget with various child widgets in a layout, more on layouts in section 4.3.

When a QWidget is used as a container to hold and group children, the QWidget is called a composite widget. A parent widget is clipped to the size that it children requires, though this can be changed in the widget's size policy.

4.2.1 QMainWindow

QMainWindow is a subclass of QWidget, and is very essential to a Qt GUI application, since the QMainWindow is a framework for the application user interface. As seen on figure 3 a QMainWindow can have a menu bar widget, toolbar bar widgets, docked widgets and a status bar widget, though a QMainWindow must have a central widget, even if that widget is only a empty placeholder.

QMainWindow is usually a good class to use as the framework for an GUI application, though it is optional whether to use it or not. In the case of RWS, QMainWindow is used, and figure 3 nicely reflect the structure of RWS' GUI, where the central widget is a custom subclassed QWidget using Qt GUI modules providing classes for OpenGL integration for graphic rendering. Various plug-ins to RWS are available to be docked in the docking area, or to be top-level windows (more on plug-ins in section 4.6) and tool bars and a menu bar are present as well for use.

QMenuBar -> pulldown menu items QToolBar -> moveable panels QDock -> docked or top-level window Central Widget -> RWStudioView3D -> openGL

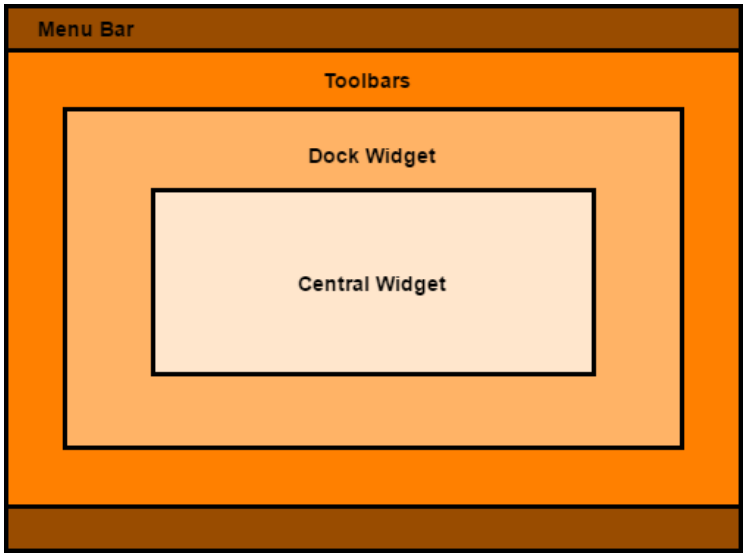


Figure 3

4.3 QLayout

Arrange child widgets within a widget Updates when contents change QHBoxLayout example. CODE AND PICTURE OF LAYOUT

4.4 Signal and Slots

Communication -> objects Alternative to callback -> explain callback Signal and slots -> connect() RET TO FIGURE Signal -> Slot -> normal function (only special thing is it can be connected to a signal) -> found by moc

4.4.1 Difference between events and Signal/Slots

4.5 Subclassing

4.6 Plug-in

5 Describe functionality the user needs, and sort in order of importance

- Loading objects into the WC, this does not necessarily mean with a drag and drop-like motion, but rather as a proof of concept without regards to the GUI.
- Simple browser for objects with a load button.
- Improvement of the GUI, this means adding a drag and drop-like feature to the GUI, where the user would drag an object from the object browser window into the WC.