**Software Design Description**

For the

**Fire and Inﬁltration Quashing System**

(FAIQ)

December 17, 2012

(14:58:47)

Project 4

Version 1.0

CSCI4830 - Introduction to Software Engineering

Fall 2012

Prepared for

***Dr. Mansour Zand***

Written by

***Shawn OBrien***

Diagrams by

***Shawn OBrien***

**Revision/Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Date | Name | Activity |
| 1 | 11/13/12 | Shawn OBrien | Skeleton starter pages. This. |
| 1 | 11/17/12 | Shawn OBrien | Prologue. Section 1. Todo list for doc. |
| 1 | 11/18/12 | Shawn OBrien | Introduction. Read IEEE document. |
| 1 | 11/18/12 | Shawn OBrien | References and Glossary. |
| 1 | 11/19/12 | Shawn OBrien | Section 4. |
| 1 | 11/20/12 | Shawn OBrien | Component / Communication Diagram. |
| 1 | 11/22/12 | Shawn OBrien | Instantiation Tree and Control Flow Diagrams. |
| 1 | 11/24/12 | Shawn OBrien | Section 6 – Interface Design Viewpoint. |
| 1 | 11/24/12 | Shawn OBrien | Section 6 – Continued. |
| 1 | 11/25/12 | Shawn OBrien | Section 6 – Completed. Re-read IEEE Document. |
| 1 | 11/26/12 | Shawn OBrien | Section 7 - Algorithm Viewpoint w/ User Logic. |
| 1 | 11/26/12 | Shawn OBrien | Table of Contents. Table of Figures. |
| 1 | 11/26/12 | Shawn OBrien | Prologue / Epilogue. |
| 1 | 11/27/12 | Shawn OBrien | Added Section 8 – Interaction Viewpoint. |
| 1 | 11/28/12 | Shawn OBrien | Cleaned up some of the source code. |
| 1 | 11/28/12 | Shawn OBrien | Inserted and highlighted some key code points. |
| 1 | 11/28/12 | Shawn OBrien | Studied UML. Sought high and low for UML app. |
| 1 | 11/29/12 | Shawn OBrien | Try UML apps. Complex UML Diagram w/MagicDraw. |
| 1 | 11/30/12 | Shawn OBrien | Converted, rotated and inserted diagrams. |
| 1 | 11/30/12 | Shawn OBrien | Spellcheck, Proofing and Printing. |
| 1 | 11/30/12 | Shawn OBrien | Email delivery. |

Prologue

This document is based on the following IEEE approved specification:

1)

**“IEEE Std 1016™-2009**

(Revision of

IEEE Std 1016-1998)

**IEEE Standard for Information Technology—Systems Design— Software Design Descriptions**

Sponsor

**Software & Systems Engineering Standards Committee**

of the

**IEEE Computer Society**

Approved 19 March 2009

**IEEE-SA Standards Board**

Authorized

**Abstract:** The required information content and organization for software design descriptions

(SDDs) are described. An SDD is a representation of a software design to be used for

communicating design information to its stakeholders. The requirements for the design languages

(notations and other representational schemes) to be used for conformant SDDs are specified.

This standard is applicable to automated databases and design description languages but can be

used for paper documents and other means of descriptions.

**Keywords:** design concern, design subject, design view, design viewpoint, diagram, software

design, software design description

The Institute of Electrical and Electronics Engineers, Inc.

3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2009 by the Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 20 July 2009. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics

Engineers, Incorporated.

PDF: ISBN 978-0-7381-5925-6 STD95917

Print: ISBN 978-0-7381-5926-3 STDPD95917”

I omit design overlays and specific design languages (with the exception of some UML and design pattern notation) from this design document. The former would invite tangential levels of detail that are likely unwarranted and could lead to confusion. I determined that there was insufficient time to master even one of these languages. Still, detail and consistency shall guide me throughout.

Table of Contents

[1 Introduction 6](#_Toc342068501)

[1.1 Purpose 6](#_Toc342068502)

[1.2 Scope 6](#_Toc342068503)

[1.3 Context 6](#_Toc342068504)

[1.3.1 Services 6](#_Toc342068505)

[1.4 Summary 6](#_Toc342068506)

[2 References 7](#_Toc342068507)

[3 Glossary 7](#_Toc342068508)

[3.1 “Design Attribute 8](#_Toc342068509)

[3.2 Design Concern 8](#_Toc342068510)

[3.3 Design Element 8](#_Toc342068511)

[3.4 Desing Entity 8](#_Toc342068512)

[3.5 Design Rationale 8](#_Toc342068513)

[3.6 Design Relationship 8](#_Toc342068514)

[3.7 Design Stakeholder 8](#_Toc342068515)

[3.8 Design Subject 8](#_Toc342068516)

[3.9 Designer 8](#_Toc342068517)

[3.10 Design View 8](#_Toc342068518)

[3.11 Design Viewpoint 9](#_Toc342068519)

[3.12 Diagram 9](#_Toc342068520)

[4 Stakeholders and Their Respective Design Concerns 9](#_Toc342068521)

[4.1 The Customer 9](#_Toc342068522)

[4.2 Back End Designer *viz.* Shawn OBrien 9](#_Toc342068523)

[4.3 Front End Desinger *viz.* Shawn OBrien 9](#_Toc342068524)

[4.4 Shared Design Concerns: Robust Operation and Clean Shutdown 10](#_Toc342068525)

[5 Architecture 10](#_Toc342068526)

[5.1 Main Components with Communication Vectors 11](#_Toc342068527)

[5.2 Revealing Instantiation Tree 12](#_Toc342068528)

[5.3 Functional Abstraction with Control Flow 13](#_Toc342068529)

[6 Interface Design Viewpoint 14](#_Toc342068530)

[6.1 Design Concerns 14](#_Toc342068531)

[6.2 Design Elements 14](#_Toc342068532)

[6.2.1 Types and Attributes 14](#_Toc342068533)

[6.2.2 Relationships 14](#_Toc342068534)

[6.2.3 Constraints 15](#_Toc342068535)

[6.2.4 Purpose 15](#_Toc342068536)

[6.3 Rationale 16](#_Toc342068537)

[6.4 Internal Interface Interaction 16](#_Toc342068538)

[6.4.1 Implementation Environment and Tools 17](#_Toc342068539)

[6.4.2 Encoded Interface Definitions 17](#_Toc342068540)

[7 Algorithm Viewpoint with User Operation Logic 19](#_Toc342068541)

[7.1 GUI Component Enumeration 20](#_Toc342068542)

[7.1.1 Restart 21](#_Toc342068543)

[7.1.2 Menu 21](#_Toc342068544)

[7.1.3 Progress Bars 21](#_Toc342068545)

[7.1.4 Text Edits 21](#_Toc342068546)

[7.2 Sensor Logic with Investigate Option 21](#_Toc342068547)

[7.3 Alarm! 22](#_Toc342068548)

[7.4 Evacuate! 22](#_Toc342068549)

[7.5 Damage Progression 22](#_Toc342068550)

[7.6 Tabbing Between Zones 22](#_Toc342068551)

[7.7 Information Console 23](#_Toc342068552)

[8 Interaction Viewpoint 23](#_Toc342068553)

[8.1 Qt Signals and Slots 24](#_Toc342068554)

[24](#_Toc342068555)

[8.2 Simulator 25](#_Toc342068556)

[8.3 UML Diagram 26](#_Toc342068557)

[9 State Dynamics Viewpoint 28](#_Toc342068558)

[Figure 1 11](#_Toc342068474)

[Figure 2 12](#_Toc342068475)

[Figure 3 13](#_Toc342068476)

[Figure 4 20](#_Toc342068477)

[Figure 51 24](#_Toc342068478)

[Figure 6 27](#_Toc342068479)

# Introduction

## Purpose

Show and describe, in detail, how the FAIQ system works.

## Scope

The scope of this document drills down into the depths of the design concerns associated with this project. The Scope only steps back far enough to yield multiple, detailed design viewpoints. For example, I present a relatively low-level algorithmic viewpoint, as well as a similarly detailed interaction viewpoint.

## Context

One may describe the context of this document as one user interacting with a graphical user interface that simulates a command console with functions available to counter both fire and security threats emulated within a large building.

The command console is semi-automatic, in that if the user is absent or unresponsive during a confirmed alarm, then the system simulates requesting emergency services external to the building. The system does not automatically handle fire or security *control* measure, nor does it expedite occupant evacuation.

Also available to the user is a non-interactive read-only information display console that scrolls relevant emergency information about the entire building, as well as the status of individual zones. The information console also displays other pertinent information, such as how fast a given fire or incursion spreads. The user will also see command echo information, casualty rates and the general status of all zones within the entire building.

### Services

Other than the obvious command and control simulation services, the command console also allows the user to exit or restart the simulation. As far as specific simulation services, please refer to one of the architecture diagrams in (Section 5) or the *Project Description* document noted in the References, [section 2](#_References).

## Summary

The design stakeholders of the FAIQ project, as well as their corresponding design concerns shall herein be identified and expounded upon.

Different design views compose the bulk of this paper, that is, I offer many different perspectives of the design of the FAIQ system. Seemingly, enough to exhaust the explication of all of my design concerns, and their respective attributes, within the context of design viewpoints. The three viewpoints that I managed to complete, perform a complete subsumption of all design entities and relationships, apply all design constraints, and expound upon the respective design rationales. (More viewpoints were planned. Unfortunately, the membership of Project 4 has decreased from four members to one member.)

Where a design viewpoint expresses activity, design attributes express that which is acted upon. All design attributes shall have a name, type and purpose.

My design viewpoints show how the system works. As my audience is technically minded, I avoid an arduous, fine-grained explanation of *how* my system works. This seems to be the intent of the IEEE specification. In fact, all of the IEEE specifications employed thus far breakdown and isolate the core functionality in a stepwise, refining manner. I expect that by the end of my current analysis, that there will be nothing left to explain. Let me see if this comes to pass. I must restrain myself, for I now begin to feel a compulsion to write all the code.

I declare consistency by the lack of conflict between my design elements and my design views.

I include a high-level, generic, functional diagram of my architecture in [section 5.1 Main Components with Communication Vectors](#_Main_Components_with). In contrast, I provide low-level, specific definitions of my object-oriented classes, replete with attributes and methods, in [section 8.3 UML Diagram](#_UML_Diagram)8.3.

# References

1. *Project Description.* Dr. Mansour Zand. September 2012. *Q:\!School\4830 Intro to SE\trunk\\_Reference\!Project Description.docx.*

2. *IEEE Standard for Information Technology – Systems Design – Software Design Descriptions (IEEE Std 1016™-2009 [Revision of IEEE Std 1016-1998]).* Software & Systems Engineering Standards Committee of the IEEE Computer Society. Approved 19 March 2009, IEEE-SA Standards Board

# Glossary

As paraphrasing definitions may result in loss of meaning, the following definitions come directly from IEEE 1016-2009.

## “Design Attribute

An element of a design view that names a characteristic or property of a design entity, design relationship, or design constraint.

## Design Concern

An area of interest with respect to a software design.

## Design Element

An item occurring in a design view that may be any of the following: design entity, design relationship, design attribute, or design constraint.

## Design Entity

An element of a design view that is structurally, functionally, or otherwise distinct from other elements, or plays a different role relative to other design entities.

## Design Rationale

Information capturing the reasoning of the designer that led to the system as designed, including design options, trade-offs considered, decisions made, and the justifications of those

decisions.

## Design Relationship

Element of a design view that names a connection or correspondence between design entities.

## Design Stakeholder

An individual, organization, or group having an interest in, or design concerns relative to, the design of some software item.

## Design Subject

A software item or system for which an SDD well be prepared.

## Designer

The stakeholder responsible for devising and documenting the software design.

## Design View

A representation comprised of one or more design elements to address a set of design concerns from a specified design viewpoint.

## Design Viewpoint

The specification of the elements and conventions available for constructing and using a design view.

## Diagram

A logically coherent fragment of a design view, using selected graphical icons and conventions for visual representation from an associated design language, to be used for representing selected design elements of interest for a system under design from a single viewpoint.” (1)

# Stakeholders and Their Respective Design Concerns

## The Customer

My customer requires an interface that is fast, easy to use, robust and capable of handling concurrent inputs, outputs and combinations thereof.

## Back End Designer *viz.* Shawn OBrien

I design the simulator that pushes simulated events to the GUI. The simulator requires a robust GUI that handles concurrent events. Further, the GUI itself acts as the central data structure, so it must update its state very quickly in order to provide accurate state data to the simulator. The simulator uses this state date to both seed and throttle its simulation algorithm.

The simulator must contain algorithms to generate “random” events that make sense given the current GUI state. This includes at least two threads to generate concurrent “random” events for each of five Tabs i.e. one Tab per zone.

Further, I bear responsibility for ten threads, corresponding to a pair of sensors in each of five zones. Threads can be tricky. I must provide a proper entry point for each thread, in addition to a well-defined termination condition. I will implement a mechanism so the threads do not dominate the CPU: a delicate maneuver where the threads lie dormant most of the time, but burst into activity at correct intervals to simulate events.

## Front End Designer *viz.* Shawn OBrien

I design not only the layout of the GUI, but also the logic to determine which components are available to the user after every event. I also ascertain whether the event emanates from the simulator or the user. I rely upon sensible data from the simulator in order to provide the user with a meaningful experience.

The layout is non-trivial. Compared to Microsoft Office, well, then it is very trivial. In any case, my layout must appear intuitive to the user. Proper placement of components allows the user to act swiftly. I limit user input to the mouse. No time exists for me to define or for the user to memorize hotkeys. I hope to leverage the middle mouse button to switch between zones. I use a *tabbed* widget for each zone, where only one zone is visible at a time, as in any tabbed web browser.

The data stream on the consolse, however, prints data for all zones, plus the overall state of the system. I must be clear and concise with my scrolling information, finding a balance between repetition, new events, and overall summaries.

One may be lead to believe that since all zones share the console, that mechanisms for mutual exclusion must be put into place. This will not be necessary because the Qt Platform constrains all GUI activity and events to one thread – the main program thread. This main program thread runs in a continuous event loop to handle GUI requests.

## Shared Design Concerns: Robust Operation and Clean Shutdown

Since I run two levels of instantiation, as well as two levels of threads, I must avoid crashes or unintelligible inputs. I plan to design clear and well-understood interfaces. Thus, I will successfully combine the GUI with the simulator.

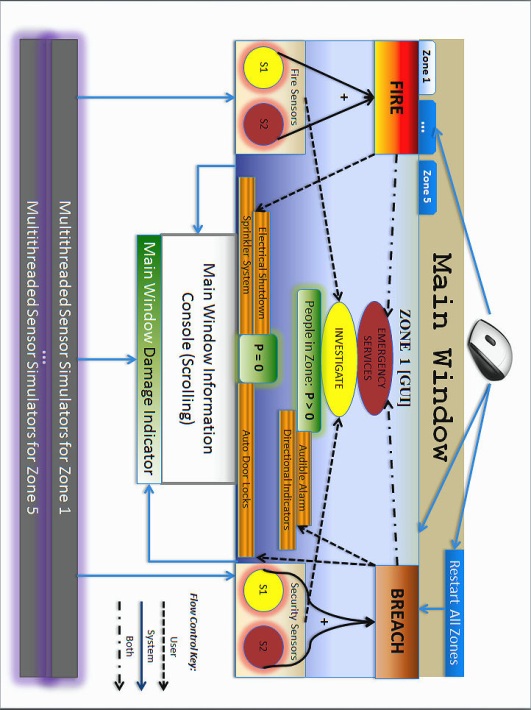
No less important, I must implement an orderly, sequential shutdown. I risk dreaded occurrences, like segmentation faults or a hung application. The design concerns for the back end consist of not only cancelling all ten threads, but also waiting for the tripped flags that confirm that the threads exited. The snag here is the possibility of leaving rogue processes alive in the operating system.

# Architecture

The following diagrams confirm that I am using a striated, stacked architecture that I further describe as an event-driven, object-oriented architectural style. I minimized n-way communication and strove for loose coupling.

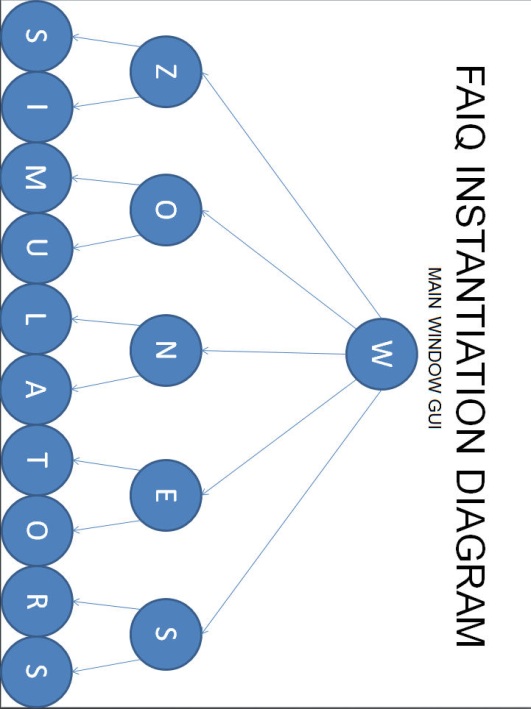
For instance, the entire Tab widget wrapper could be replaced with any other kind of widget that had an identical parameterized constructor and sensor() signal slot. If this were a real life contingency, then I would incorporate a pure virtual C++ interface from which the Tab class must derive (as well as any alternate widget wrappers.)

## Main Components with Communication Vectors



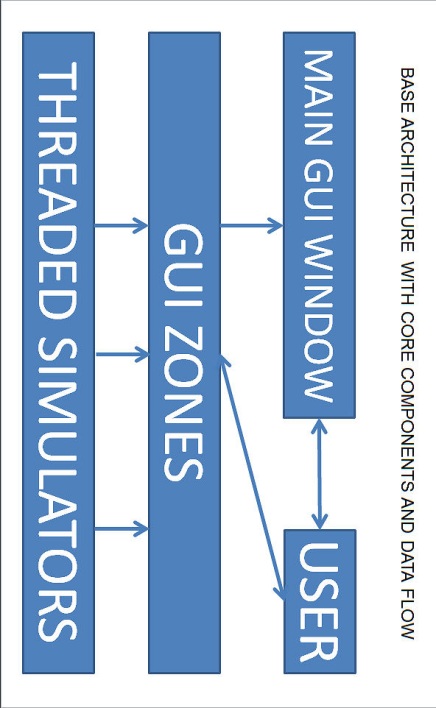
Figure

## Revealing Instantiation Tree



Figure

## Functional Abstraction with Control Flow



Figure

# Interface Design Viewpoint

## Design Concerns

I concern myself here strictly with the internal interfaces of the FAIQ system. The external interface received treatment from previous documents. Further, with each viewpoint covered, less material exists to cover in the next viewpoint – my program is not terribly large or complex. See section [1.3 Stakeholders and Respective Design Concerns](#_Stakeholders_and_Respective) for services provided by the external user interface. For a deeper understanding of the user interface and its operation, see section [7 Algorithm Viewpoint with User Operation Logic](#_Algorithm_Viewpoint_with).

## Design Elements

Herein lays an exhaustive explication of the primary design elements. Keep them in mind and follow cross-references when provided as you delve into later viewpoints.

### Types and Attributes

Three principal design entities exist, named after the actual class names: 1) MainWindow – the GUI main window that provides the entry point into the program, and an event loop that handles all GUI events; 2) Tabs – the main window child widgets, namely, exactly five tabbed widgets that are also governed by the main thread event loop; 3) Sims – the simulators, instantiated and threaded by the Tab widgets, provide random inputs to the Tabs in order to stimulate the user.

One may categorize MainWindow and Tabs as standard GUI components, both governed by a single thread; whereas the Sim class stands apart, providing non-gui worker threads that emit signals with a random frequency.

### Relationships

The GUI entities know about each other. The MainWindow instantiates the Tabs. MainWindow holds back pointers to the Tabs in order to signal a simulation restart or shutdown, and to execute proper deletion. Further, MainWindow reads the individual Tabs’ damage progression in order to display an aggregate damage progress bar.

As the MainWindow houses the information console for all of the Tabs, each Tab gets a pointer back to the MainWindow for logging status information. I thought of sending log messages via signals that would have obviated keeping a pointer back to the main window. But I ran into a widget surprise.

One of MainWindow’s child widgets is a Tab Widget, which is a container for other widgets. That is, when I add a widget to the Tab Widget, the added widget gets a little tab bar at the top left. But this little tab bar *viz*. QtabBar resides within the Tab Widget. Therefore, to get at the tab bar, I needed access to the tab widget, which required access to MainWindow.

I might be able to “refactor” things so Tab does not need that pointer back to MainWindow. I may not have time.

The MainWindow does not know about the Sim objects. The Tabs do, but only hold onto pointers for the singular purpose of canceling the Sim threads.

The Sim objects know nothing. They simply emit random signals and do not care about their fate.

### Constraints

The singular design constraint imposed by the system is that all GUI events must occur in the main thread. This forced me to make the simulators minimalist. That is, I made the Sim class ignorant of the GUI, as Sim objects contain one worker thread each. Each Tab instantiates a pair of Sim objects. The existence of these threads, in pairs, satisfies the original system requirement of simulated concurrency.

I have not had to implement any mutual exclusion mechanisms despite the fact that I bombard the main event loop with concurrent signals. This is the case for a couple reasons. First, the API I selected, the Qt Platform, contains a multitude of thread-safe components, including the main event loop that starts when main() instantiates object MainWindow.

Second, even though the Tabs support reentrancy because of their one-to-two correspondence with the Sims, this characteristic is not necessary. In the background, the main event loop intercepts the Sim signals and queues them. The singular main GUI thread then processes them one at a time, which harks back to the original Qt constraint that all GUI events must occur in the main thread.

### Purpose

MainWindow exists to provide the infrastructure for the program. Visually it provides a bounded window that describes the application on the display device. Functionally, in itself, it provides the parent object for additional widgets, and, most importantly, it provides the efficient main event loop.

Tabs conveniently simulate the five building zones. Deploying a single gigantic window object with duplicitous controls for each zone, defies proper coding practices, and would create a confusing, monolithic interface. Like all modern browsers, the tabs provide one-click access to a single zone. From an implementation perspective, such a design only requires the definition of a single class that may be easily instantiated five times.

The Sims exist not simply to provide a realistic simulation for the user, but in their multiplicity, they satisfy the system requirement of a program that handles and emulates concurrent events.

## Rationale

I justify the resultant design of this internal interface view as follows, where code and platform reuse played a surprisingly large role.

I had available some existing code that I previously wrote for an open-source project. I also had an older version of the free Qt Platform, including editor, compiler, form designer, debugger and myriad excellent libraries.

I should note that this code reuse came at a much higher price than I expected. I trimmed, and cut and sculpted the existing code for a very long time, probably removing 70% of the original code. Starting from scratch, or selectively cannibalizing the pre-existing code would have been faster.

Having the development platform available at about 250MB allowed me to distribute it to the entire team, on either a USB stick or using Internet cloud storage. The Qt Platform is free software from Nokia. A commercial version does exist as well. If I had to use the *current* version of Qt, I would have had to actually install the compiler manually, and then compile all of the Qt libraries. I spent a few hours on this, but continually ran up against a mismatch in the compiler version. So I dug up Qt version 4.60, an all-in-one package, from an old backup tape.

I decided upon Tabs because the previous application used Tabs. I think I would have used Tabs anyway, yet it was nice to have the architecture in place.

Similarly, though the worker threads from the pre-existing code performed much more extensive work (requiring semaphores,) once I trimmed all the excess code away, I had a nice little threaded class to spawn from the Tabs.

The loose coupling among the three cardinal entities, I believe, justifies my choice to leverage the pre-existing code, and to, mostly, keep the original architecture in place.

Ultimately, the existence of multiple threads, as well as both GUI and non-GUI components, provide the rationale for my internal interface design.

## Internal Interface Interaction

The time has arrived to take the lid off and fully expose the nature of my implementation environment and tools, as this is necessary for the forthcoming encoded interfaces.

### Implementation Environment and Tools

Nokia develops and actively maintains a powerful, multi-platform, C++ framework, complemented by a rich set of C++ libraries, with any eye on quick deployment of cross-platform windowed applications. They call it Qt. Nokia packages Qt with a complete IDE, including a slick editor, the MingW version 4.4 compilers, an integrated form designer that provides drag-and-drop GUI-building facilities and code generation, not to mention a Python version of the classic UNIX gdb debugger.

I chose version 4.60 of the Qt Platform, Copyright © 2010 Nokia Corporation (see section [6.3 Rationale](#_Rationale) for additional details,) because it was available and I had some existing code. Qt is 32-bit and runs on Windows XP and Windows 7. I therefore chose C++ as my development language, using the Qt Platform, running on Windows 7. My target is primarily Windows 7, or whatever runs in the PKI lab, but my app will run on Linux and Xwindows (though I have yet to test on my virtual machines.)

### Encoded Interface Definitions

IEEE 1016 provides for a distinct interaction viewpoint, so I encroach upon that with actual C++ code in the form of class definitions here, and startup/shutdown interactions in the next section, directly below.

class MainWindow : public QMainWindow { … }

provides the interface to Qt via inheritance.

The constructor interfaces with the GUI form component that I built with Qt Designer (the form designer) by constructing a pointer to the form:

MainWindow::MainWindow () : ui(new Ui::MainWindow) { … }

The handy *ui* pointer grants access to all of the MainWindow GUI controls.

Here, limit your focus to the keyword “new” in the following code, where MainWindow creates the Tabs:

void MainWindow::createTabs () {

for (int i = 0; i < NUMZONES; ++i) {

…

tw->insertTab(i, tabs[i] = new Tab(), QIcon(), label);

}

}

MainWindow similarly deletes the Tabs.

Here, observe the simplicity of the program entry point:

int main (int argc, char \*argv[]) {

QApplication app(argc, argv);

MainWindow w;

w.show();

return app.exec();

}

Similar to MainWindow inheriting from QMainWindow, Tab inherits from Qwidget:

class Tab : public QWidget { … }

Also in similar fashion, Tab instantiates the corresponding widget form, and holds back a pointer to it:

Tab::Tab () : ui(new Ui::Tab) { … }

You will find the following far more stimulating. Tab creates the pre-threaded Tab objects using the QthreadPool library:

void Tab::simInit () {

for (int i = 0; i < NUMSIMS; ++i) {

sims[i] = new Sim();

connect(sims.at(i), SIGNAL(sensor()), this …

pool.start(sims.at(i));

}

}

When the user exits the program, Tab banishes the threads by tripping a cancel flag via direct function invocation. QthreadPool automatically deletes the Sim objects. Notice that Sim knows nothing about MainWindow or Tab. Modular replacement could be completed very easily.

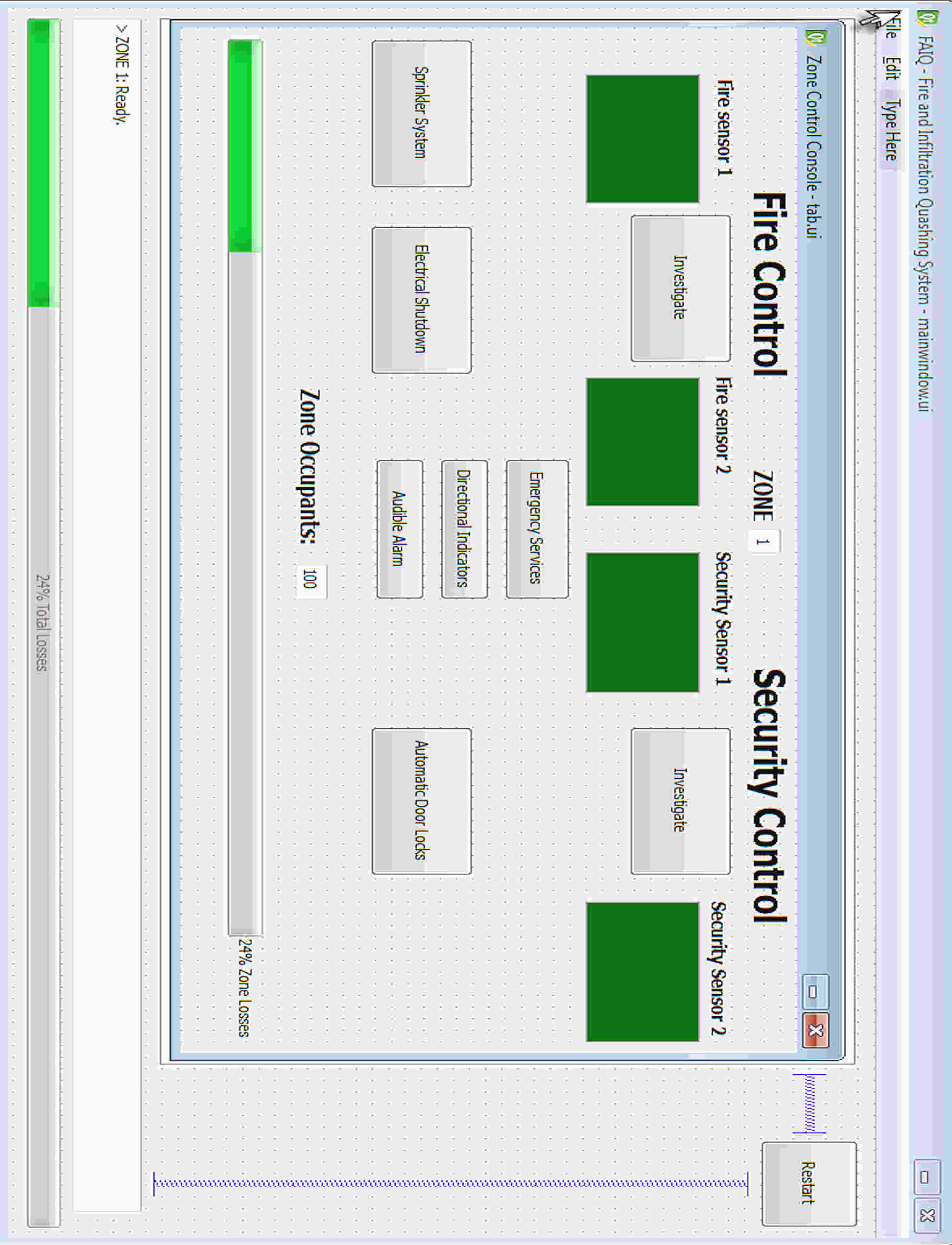
Anything further transgresses into other, distinct viewpoints. It is time to return to English in any case.

# Algorithm Viewpoint with User Operation Logic

The simulation algorithm maps well to the logic required by the operator. After all, they are pitted against one another. Hence, I shall focus with primality on the FAIQ algorithm, yet will introduce complementary operator approaches along the way. This compensates for focusing exclusively on the internal interfaces in the previous section.

The user’s goal is to minimize loss of life and damage to the building. Thus, the most important action is requesting Emergency Services. After that, immediately commencing evacuation, properly, increases the user’s chances of success. Finally, activating control measures while trying to save the entire building engages the user until first responders arrive or the zone is destroyed. The former immediately stops loss of life and damage; whereas the latter leads to the demise of any remaining occupants. Damage accrues until emergency personnel arrive for all zones. Conversely, the loss of all zones results in 100% damage and loss of all remaining occupants.

## GUI Component Enumeration



Figure

Note that in Figure 4 directly above that the information console has been extensively minimized in order to showcase the other controls. This is actually a screen shot from the Qt form designer with a single Tab form overlaid on the MainWindow form. The user manual will include actual application screen shots where there are five tabbed tabs. (See reference (2) *Project Description* in section [2 References](#_References) for the original, detailed purpose of the buttons and sensor components.)

### Restart

Push to Restart simulation. This resets all internal logic *and* external controls.

### Menu

File | Exit to shut down and quit (achievable by hitting the red “X” in the upper right corner.) Edit | Restart mimics the Restart button (directly above.)

### Progress Bars

The MainWindow progress bar monitors total damage across all zones. Each tab displays a progress bar for local zone damage totals.

### Text Edits

The identifying “ZONE N” text edit remains static and serves the obvious purpose. Far more dynamic, the “Zone Occupants” text edit emulates zone populations remaining initially and after evacuation commences.

## Sensor Logic with Investigate Option

The FAIQ algorithm, housed within the Tab class, relies on the arrival of non-parameterized, random signals from the Sim class objects. The Qt thread-safe random number generator allows periodic signal emissions from the Sim object by putting the worker threads to sleep for a random, non-zero interval.

When a tab receives a signal, it rejects roughly half of them, which is merely the product of performing enough analysis to arrive at a realistic rate. A simple parity check using modulus two determines whether the signal applies to either fire or security emulation. Each fire alarm has a pair of sensors, as does each security alarm. A single activated sensor may be a false alarm and warrants investigation. When both sensors activate, this indicates a definite fire or security breach.

Upon successful signal reception, a Tab attempts to activate the first fire or security sensor. If successful, the user has one chance to hit the Investigate button, with a low probability of deactivating the first sensor i.e. the hypothetical “false alarm” scenario. The user may miss this chance if the Tab activates the second sensor prior to investigation. Either way, the Investigate button goes away when both sensors activate.

## Alarm!

A pair of activated sensors indicates a definite alarm, whether it be fire sensors or security sensors. The remaining control buttons then become available. If the user fails to press the Emergency Services button within ten seconds, the system “presses” it automatically.

It takes a random amount of time for Emergency personnel to arrive. Once they “arrive,” no further simulated damage occurs.

Until they do, the operator must match wits with the simulator. Fires spread. Security breaches cause more and more damage. The user may choose from many control measures. Despite their availability, premature usage may cause loss of life and increase damage. The user must monitor how many occupants remain in the zone, as well as the progression of the damage, plus guessing when Emergency Services will arrive.

For instance, activating the Sprinkler System prior to Electrical Shutdown electrocutes a random number of occupants. Similarly, tripping Automatic Door Locks during a fire will trap and burn an additional random number of occupants.

At the same time, the simulator does not count deceased occupants. Therefore, thinning the crowd a little to save the entire zone from destruction often results in fewer losses.

## Evacuate!

Other than premature "death,” the occupants will not begin to evacuate until the directional indicators are activated, prior to the Audible Alarm. The reverse order will cause some occupants to be “trampled” to death.

## Damage Progression

Simulated damage occurs over time. Control measures slow down rates of damage. The arrival of Emergency first responders stops damage from spreading. It is possible to lose entire zones, even using the best operator logic. This adds a sense of urgency to the user experience and reflects my inability to calibrate the mixture of random probabilities very well.

## Tabbing Between Zones

The goal is to save all zones with zero loss of life. Some opportunities may be missed because only one Tab displays at a time. The simulation continues whether a tab is visible or not.

One design concern I have is whether I can flash, or otherwise draw attention to, hidden tabs that require user attention. I will have to wait until final implementation for this.

I limit user interaction to the mouse. Learning hotkeys during a short demonstration takes too long. Further, removing the keyboard lessens the likelihood that my professor will be able to crash my application by messing around with the keyboard.

Both the mouse constraint and the “hidden tab” constraint restrained my algorithm. This forced me to inflate the random time intervals in order to compensate for the user switching tabs or maneuvering the mouse. I think I achieved a natural, realistic feel.

Though perhaps more applicable to another viewpoint, I should mention that I made large buttons on purpose. Easy to click. I also appreciate the autonomous worker threads, as they keep the main GUI thread highly responsive.

## Information Console

The information console echoes the algorithmic logic, as well as the logic employed by the operator. Console messages paint a picture of how the algorithm operates and interacts with the user. I aim for conciseness, providing hints to the user, and humorous admonishments for “killing” occupants or “losing” entire zones.

# Interaction Viewpoint

The design concern for this viewpoint, messaging, distinguishes itself as the chief concern for a couple of reasons. First, the other concern recommended by IEEE 1610 – object communication – does not play a very large role in the FAIQ system, except primitive forms of communication, namely, instantiation and deletion. Second, the FAIQ system does not amass any data that must be passed hither and tither. The FAIQ system changes state quickly by using the robust Qt signals and slots mechanism. Once state changes, previous state data become stale and goes away.

## Qt Signals and Slots

## 

Figure 1

“The signals and slots mechanism is type safe: The signature of a signal must match the signature of the receiving slot. (In fact, a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.) Since the signatures are compatible, the compiler can help me detect type mismatches. Signals and slots are loosely coupled: A class which emits a signal neither knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensures that if you connect a signal to a slot, the slot well be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. They are completely type safe.”[[1]](#footnote-1)

## Simulator

void Sim::run() {

while (!m\_cancel) {

QThread::sleep((::qrand() % 20) + 3);

emit sensor(); // signal

…

}

}

Here, the simulator thread for a Sim object sleeps for 3 to 23 seconds, and then emits a generic signal – whence it goes, Sim does not know. Ultimately, a Tab object “receives” the signal in a slot, coincidentally having the same name:

void Tab::sensor () { // slot

static int cnt = 0;

if ((::qrand() % 100) < 50) // reject half

return;

trip(cnt % 2 ? ui->fireTextEdit1 : ui->secTextEdit1,

cnt % 2 ? ui->fireTextEdit2 : ui->secTextEdit2,

cnt % 2);

++cnt;

}

As you can see the signal is picked up by the slot, whereby about half of the signals find themselves rejected for timing purposes. The remainder trip a fire or security sensor based on the parity of a local, increasing static int.

Skipping the complicated trip() function, let me show the slot that handles the signal emitted by the Investigate Button:

void Tab::on\_secInvestigateButton\_clicked() { //slot

ui->secInvestigateButton->setEnabled(false);

con("Investigating security sensor 1...");

if ((::qrand() % 100) < 75) {

ui->secInvestigateButton->setVisible(false);

con("Suspicious visitors in lobby.");

return;

}

ui->secTextEdit1->setEnabled(false);

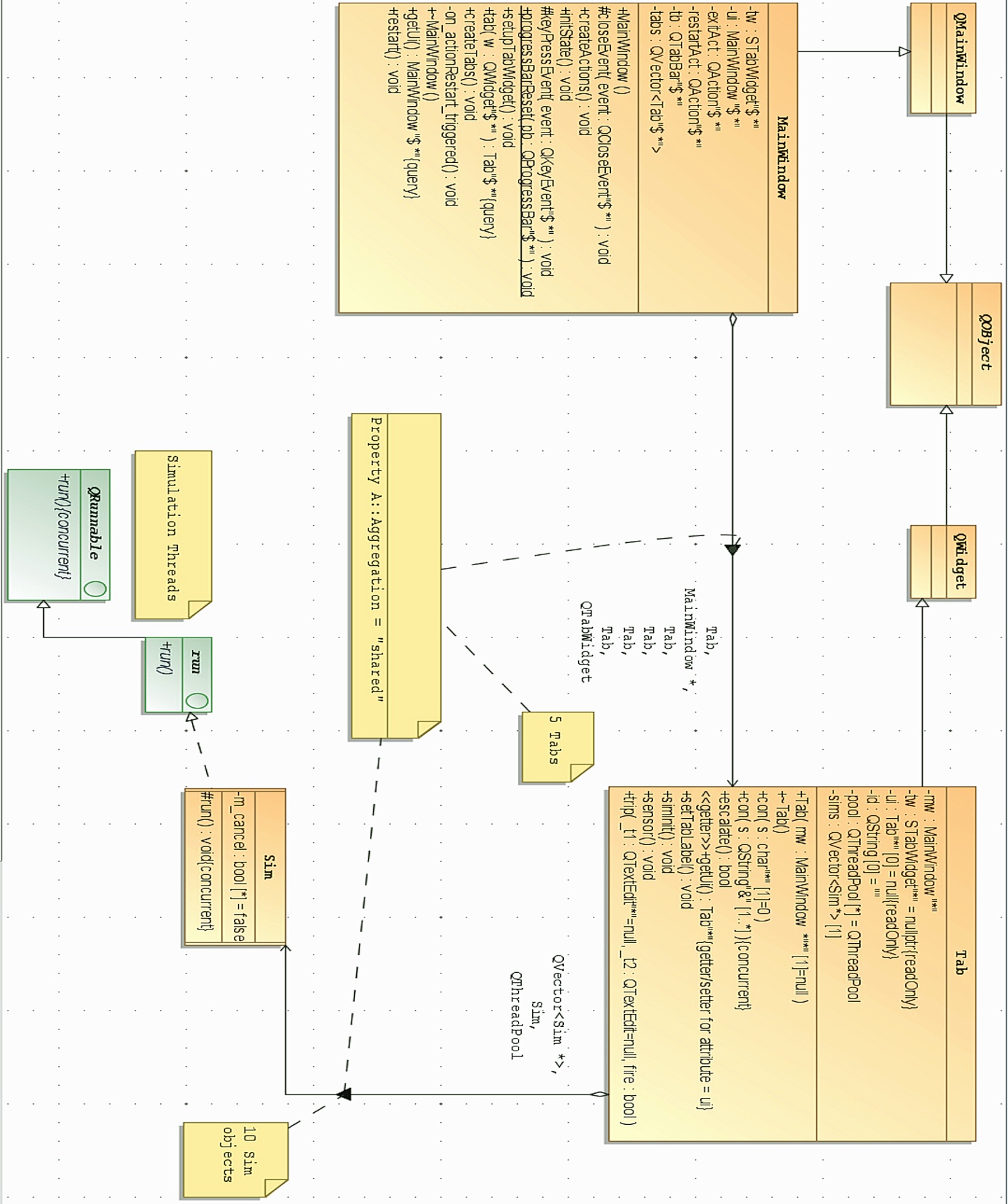
con("Security Investigation team report: FALSE ALARM.");

}

Noticeably, the investigate button usually fails to silence the first sensor, though occasionally the first sensor is in fact reversed. (Helper function Tab::con() uses the pointer back to MainWindow to directly dump messages to the MainWindow TextEdit.)

## UML Diagram

To round out the interaction viewpoint, I submit a detailed UML 2.0 class diagram. I hope the UML perspective pulls the last three views together into a cohesive whole:



Figure

# State Dynamics Viewpoint

Regrettably, this highly relevant view must be excluded because I do not have the capacity, or time, to complete it, alone, without jeopardizing my health, integrity and additional 4830 responsibilities.

I certainly hope that my program handles the necessary state dynamics well.

Epilogue

You have seen a bit of code in this document. Perhaps too much. Given the small scale of the program, I found it helpful to write rapid prototype code while working through the design documentation of various viewpoints. I only insert code that I intend to keep in the program, and then, only code that is both simple and illustrative.

An additional reason for the possibly premature arrival of code is that I had to aggressively breakdown the pre-existing code into relevant skeletal code so that my teammates could get their teeth into the coding process immediately after wrapping up the design.

At the risk of digressing, I must mention that my teammates completely and totally decided to excuse themselves from the entire design process, have not looked at any code, and are now incapable of assisting with the implementation or user manual because the last time they had any substantive input was during the writing of the System Requirement Specification.

At this point, it seems unlikely that they well be able to contribute anything useful for the remainder of the project, as they have zero interest in the project. They ignored no less than three versions of prototype code, along with the proposed Qt platform, and two working prototype applications.

In order to wrap up this injection, let me conclude by stating that I have not been contacted by any teammates in about a month. We banter after class, and they make false promises. During this last month I have spent countless hours writing ignored emails and setting up an exceedingly trivial repository for version control. I have written pages of tutorial emails as well. I continually ask: what do you guys think? Do you have a better idea? But I get no response.

I issued direct, well-defined imperatives on who should do what. No response. If I do nothing, they do nothing. I refrain from nagging or threatening and refuse to do so.

I laid every task, opportunity and tool at their feet.

[Back to Top](#_top)

1. Qt Assistant Version 4.6.3 Copyright © 2010 Nokia Corporation and/or its subsidiary(-ies). [↑](#footnote-ref-1)