Individual Project - CM0343

Interim Report

James Briggs (1029537)

Abstract

This report details the background of the project including an in depth description of the problem at hand, existing solutions and why these aren't always feasible. I will also describe how my approach will function with justification of how it presents a better approach than existing solutions. A coherent, bulleted list of project requirements is presented along with justifications. The requirements are expanded upon throughout the report relating back to the original project intentions.

Table of Contents

Abstract 1
Table of Contents 2
Introduction 3
Background 5
Approach 8
Conclusions 12
References 13

Introduction

This project aims to ultimately solve the problem of specifying cross-platform, extensible Graphical User Interfaces (GUIs). There have been many attempts at solving this problem, some of which have been majorly successful such as Microsoft Visual Studio, however the problem of cross-compatibility still remains. The approach I propose involves using XML to specify GUIs as an abstraction above the GUI framework used. The key objective is to implement a method of specifying a GUI solely within an XML file which can be ported across multiple platforms, programming languages and GUI libraries. I aim to make the system as adaptable as possible to a wide range of applications whilst still maintaining a simple and uncluttered architecture. This involves creating the system in a way that doesn't limit the potential uses or confine it to a certain platform or way of working. As part of the project I aim to implement an event definition mechanism whereby events and subsequent actions can be wired to GUI components within an XML file. Once the specification has been set out I plan on creating a visual GUI editor similar to the one found in Microsoft Visual Studio [1] or QT Creator [2]. This should help ease GUI development as well as enforcing the business logic of the system. To enable easy use of the XML GUI definitions within applications I will create a library for each of the language/GUI framework combinations.

On completion this project could have a potentially vast array of beneficiaries. At first glance it appears that it is solely focused towards desktop application programmers however this is not the case. Whilst application programmers may be the most prevalent users I feel the system could also be of great use to researchers developing prototypes or performing experiments. Traditionally creating intuitive GUIs is a laborious process renowned for its difficulty however using tools such as the one I am proposing can greatly speed up the process. The abstraction from the code could even allow novice users or designers unfamiliar with traditional programming to specify interfaces and event maps. The system suits many design methodologies and facilitates Rapid Application Development (RAD) to a great extent (initial application prototypes could be created from designs with relative ease) [3].

In order to complete the project on time and maintain a simplistic yet useful architecture the scope has to be limited to an extent. Whilst the concepts demonstrated could be applied to almost any programming language and GUI framework I have chosen to limit development to two combinations: Python+wxWidgets and Java+Swing. I will also limit the supported GUI components to those available in both wxWidgets and Swing. This will ease the complications of supporting two GUI frameworks whilst maintaining platform consistency.

Due to the nature of the project I feel the Agile software development method is most appropriate. As this project is somewhat exploratory a defined development plan cannot feasibly be created. An iterative, incremental approach suits the project far better than the traditional waterfall model. Initially I will start with a small set of GUI components (3 or 4) in order to test the feasibility of the planned approach. This set will be expanded throughout the project until a mostly complete set is implemented.

In summary the key outcomes of the project will include two libraries for loading, rendering and implementing the event signalling elements of the GUI as well as a visual editor to ease design and development. Additional, less tangible, outcomes include the method of describing GUIs in XML and component schemas.

Background

Traditional command line applications are arguably easier to develop than GUI applications however they are not as user friendly. Development of GUI applications usually invokes a large time overhead as it is notoriously onerous. Transforming a written or drawn design into a hard coded user interface represents a fairly non-trivial task as it is essentially mapping a visual design into a coded specification. This burdensome task can be trivialised through the use of a visual GUI designer which can encode the interface without requiring the user to write code himself. Simplifying the GUI development process has applications not just for desktop application developers but also for researchers and designers who may want to create interfaces quickly and at a higher level than code.

There are an abundance of GUI toolkits available to developers in every language however they are not standardised with each one taking a different approach to the problem. Usually interfaces have to be hard coded into an application whereby each element has to be instantiated individually. This often results in many lines of code muddying the actual application logic simply to render the GUI and handle events. Additionally most GUI frameworks have syntactical differences of a varying degree. This means that a learning curve is involved for each framework used. When developing for a single platform this is not usually a problem however when developing across multiple platforms (such as Windows, Mac and Linux) this represents a significant issue. Many developers experience significant problems when trying to create applications for multiple platforms. For example the music streaming service Spotify was forced to entirely rewrite their GUI code when porting the application to Linux [4]. This leads to poor maintainability and potential inconsistencies.

There have been several attempts to create XML dialects for defining user interfaces however these are usually limited to a specific or proprietary rendering system, such as Microsoft Visual Studio combined with Windows Forms (WinForms) or Mozilla's XUL (XML User Interface Language) used by the Gecko layout engine in products such as Mozilla Firefox or the Songbird music player [5]. The Gecko layout engine is used within products such as Mozilla Firefox to both render the content of HTML pages and the applications user interface. The versatility of the engine comes with the overhead of increased complexity. For example the Gecko engine includes support for styling user interfaces as well as a complex Document Object Model. For users wanting to create an interface quickly and simply (such as researchers) or less experienced users this is overkill and will likely hinder progress. The Gecko layout engine (when used within Mozilla Firefox) essentially allows the user to specify the layout of an interface in an XML based dialect with accompanying CSS for defining the appearance of elements. The system I am proposing takes a more lightweight approach which results in simpler use and a smaller learning curve or time overhead.

The cross platform Qt toolkit is not limited to a specific platform however it is not considered a lightweight framework and requires a licence for commercial use (an Open Source version is available however the programmer is obliged to Open Source the entire project). The significant difference between existing systems and my project is that it can be used in conjunction with a potentially large number of different GUI libraries in addition to being built with extensibility in mind.

The project I am proposing draws many comparisons to the existing method of prescribing web pages using HTML and CSS whereby the HTML element describes the components of a page and the CSS part describes their layout. The overall architecture of the system is also somewhat similar to the Model View Controller (MVC) pattern whereby the view portion is represented by the GUI specification file, the controller part by the API library and the model part by the actual logic implemented within both the XML event mapping mechanism and the Python/Java files. The logical method of arranging components is naturally hierarchically as shown in Figure 1. This makes it simple to represent components within an XML file as well as helping to arrange them visually within an interface. In order to provide a programmatic level of access to GUI components whilst still maintaining the encapsulation of the library I am planning to implement an API following a similar method to the Document Object Model (DOM) whereby interface components can be accessed and manipulated through code. Similarly to the component schema this will be limited at first however will gradually be built upon to create an acceptable level of completeness.

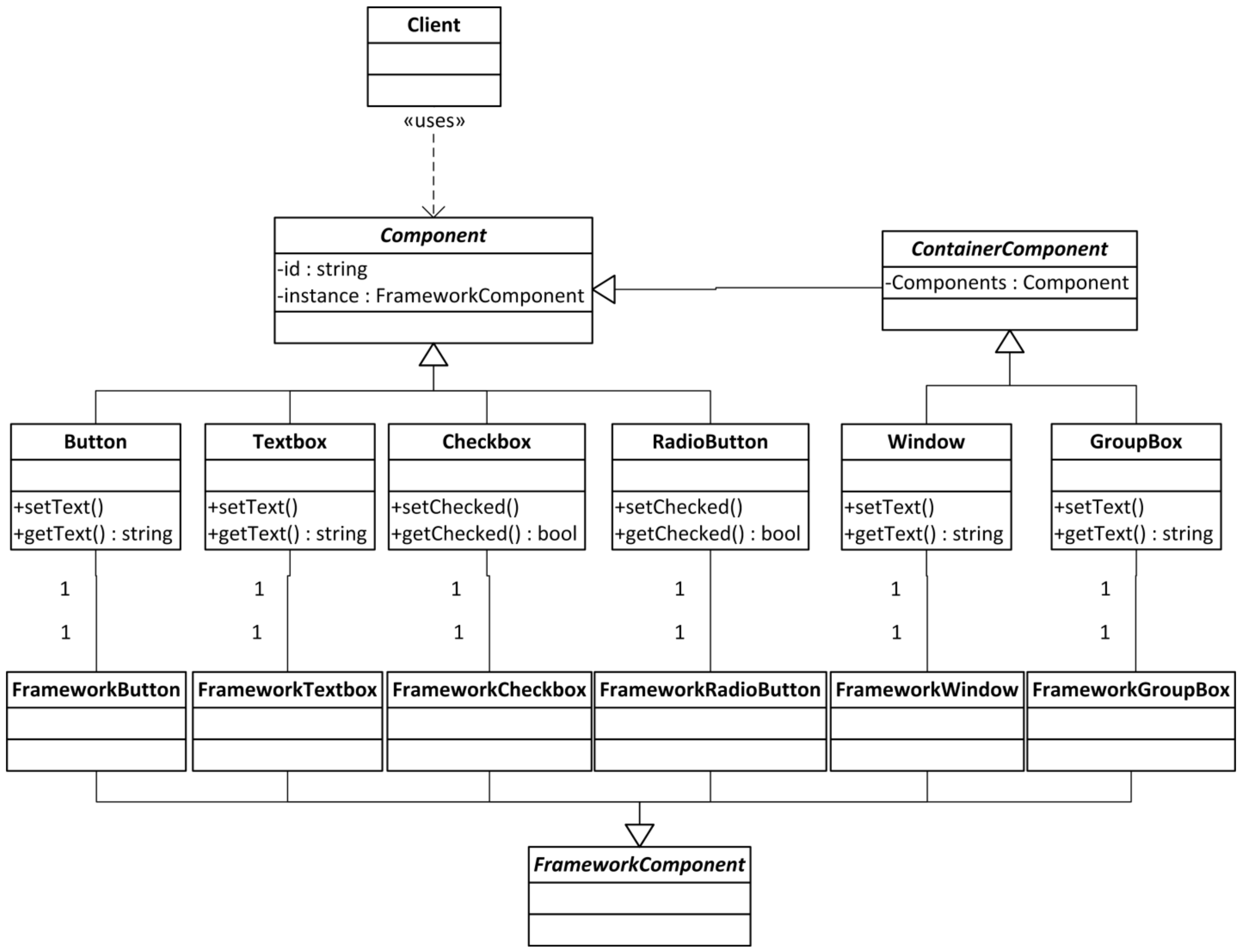

Figure 1: *UML Class Diagram showing the component hierarchy.*

As GUI definitions must conform to a specific style it is necessary to implement XML schema files defining all of the available GUI elements and their attributes. XML schemas have the advantage over Document Type Definitions (DTDs) as they allow specification of data types for element attributes. This is important for this project as a single mistake within an XML file could result in an entirely unrenderable interface or unforeseen results. The use of schema files also enhances the prospects of extensibility and allows for somewhat trivial versioning as long as elements within the schema correspond to elements available within the library.

As mentioned previously there are several similar systems already in place. Although they have some drawbacks there are many elements in which they excel. For example, Microsoft Visual Studio includes a very powerful visual forms designer which allows developers to create a user interface by dragging and dropping components onto a window. These components can be interactively rearranged or edited using the property inspector. Microsoft Visual Studio itself makes use of XML files for storing the layout of components however this is limited to the design stage. C#, Visual Basic or other .NET Framework based code is transcompiled from these files. This would not be an appropriate strategy for this project as the XML interface definition files are designed to be cross-platform and not limited to a particular language. Additionally the complication of translating an XML file to compilable Java/Python complicates the otherwise straightforward build process.

Event handling within GUI applications is a complex task. For this project I have borrowed many of the ideas behind signal programming and the observer pattern. Essentially events applicable to each component can have input/output functions associated with them. The mappings between events and functions are created within an XML file whereas the functions themselves are prescribed within the code behind an application.

Approach

There are a number of functional and non-functional requirements associated with this project. As mentioned in the introduction it is difficult to set a rigid structure and specification for the final system as it is somewhat exploratory. The following details some of the loose requirements of the project:

- Functional
  - Render an interactive GUI across multiple platforms, languages and libraries from a single XML interface definition file. In order to cater for a large audience, particularly when dealing with advanced users such as researchers, cross compatibility is essential. Many research projects involve cross disciplinary work where departments may have varying platforms.
  - Implement a visual interface designer capable of generating XML interface definitions documents. Defining interfaces solely using a text editor is a difficult task and cannot be visualised without repeatedly loading the interface into the desired application, making changes and re-loading the interface.
  - Implement two API libraries programmatically exposing the functionality of the system. Targeting two programming languages (Python and Java) helps to reinforce the cross-platform nature of the project.
  - Emulate the functionality of the Document Object Model (DOM) through the API. The DOM is a proven way of effectively specifying a hierarchy of elements within a user interface.
  - Create an event signalling/handling mechanism capable of encoding within an XML document.
  - Facilitate a high level of GUI customisation. Users needs vary greatly. A successful system should allow users to create a range of different applications not limited to a specific domain.
  - Produce a small number of example applications demonstrating the diversity of the final system.
- Non-functional
  - Create and fully document XML schemas describing the available components and their attributes/events. For the purposes of initially learning how to use the system, future reference and maintainability documentation is necessary.
  - Create a flexible system built for expandability. Future changes in the frameworks should be allowed for in the system.
  - Maintain a simplistic, lightweight and consistent approach throughout the system. This will make it simpler to quickly and effectively implement interfaces.
  - Facilitate both novice users (who may prefer visual design tools) and advanced users (who may require higher levels of customisation via lower level interfaces).
  - Allow both small/trivial and larger scale/complex applications to be created.

This project involves creating more than one interface. The most obvious interface will be the visual GUI designer which will allow users to create interfaces in a drag-and-drop style. The second interface is the API implemented within libraries for each programming language and platform combination. This will expose components programmatically allowing their attributes to be accessed and manipulated within applications. This helps to enhance the flexibility of the system whilst adding an interface layer for more advanced users.

A system such as this demands high levels of customisation and dynamicity. Self describing markup languages like XML lend themselves very well to these sorts of applications. Whilst pure XML does not provide a strict way of monitoring and enforcing document restrictions the combination of the visual interface designer and XML schemas does. As an example certain elements (such as the top level window) have an instance limit that must be imposed in order to create a functional interface. XML has the major benefit over compiled code of not being tied to a specific platform. Interface definitions can be used on any supported platform without the need for transformation or rewriting. The use of XML schemas facilitates future expansion. Extension should be a trivial exercise as long as each of the elements described within a schema are reflected by an implementation within the API library.

The processes involved in creating, storing, rendering and interacting with an interface require data to flow between many different points in a number of formats. Within the interface editor the component list must first be populated based on the components prescribed by the XML schema. With this validation data such as attribute types or number of allowable instances should be included. Instances of components are stored as class instances which, upon saving of the document, are encoded into XML form. When loading a document they are decoded back into Python class form. There are two potential approaches to this: XML files could be manually created by the editor based on the attributes of classes or a package such as pyxser could be used to automatically serialize the data. The disadvantage of automatic serialization is the lack of control leading to potentially compatibility issues (a Java serializer may produce different output to the Python serializer). Developing the system to create XML definitions to a defined, concrete schema is more labour intensive however produces more consistent, predictable output. Within the actual client application data can be represented in a more compact, efficient way. Some of the attributes required by the editor may not be required within the client application. Most of the data can be stored within a component instance specific to the framework and retrieved in a standardised way through the use of adapter classes holding the component instance.

Similarly the event signalling/handling system also has complicated data flow requirements. An XML based mapping of functions to component events is defined within the interface editor. This must be implemented automatically within the client API libraries. This may prove difficult as functions will need to be called dynamically upon event execution based on a mapping provided at run time (usually event handling functions are defined at compile time).

In order to implement the project effectively a defined structure and set of data types are required. As shown in Figure 1 a layer of abstraction is required between the client application and the underlying GUI elements within the chosen frameworks. This helps to maintain consistency in addition to a level of encapsulation. When manipulating components this allows validation constraints to be imposed or additional code to be executed. Figure 1 also demonstrates a clear distinction between container components and inline components. Essentially container elements can include any number of components within them (to a potentially infinite depth) whereas inline components cannot include any. As demonstrated by Figure 1 container components can be both a leaf and branch node of the component tree whereas inline components can only be a leaf. Naturally a single container component is required as the root of the tree. Representing components in a hierarchical manner helps to maintain organisation of the interface in addition to lending itself well to encoding within an XML file. Additionally the event-to-function mappings will need to be encoded in XML form. These mappings will include information such as function names input/output variables and data types.

Many principles from both the proxy and adapter patterns will be used. The adapter pattern allows a layer of abstraction to be added over existing classes. As mentioned previously this is required to maintain cross platform consistency and encapsulation. This is also similar to the proxy pattern as the abstracted class effectively acts as a proxy between the underlying GUI component from the framework and the API library/XML interface definition.

I will take an iterative and incremental approach to development whereby the system is implemented cyclically. Each cycle will consist of implementing a portion of the system, analysing the success and refining it if necessary.

I expect to encounter many problems during the design and implementation of the remaining part of the system. During the implementation of the first prototype I found that wxPython (the GUI framework used) came with a steep learning curve. It also appears that the framework includes several glitches or "gotchas" for example a theoretically well laid out GUI did not render as expected as factors such as margins and operating specific differences had an effect. Additionally the documentation is not the as concise or accurate as it could be with many aspects of the framework only covered briefly or not at all. One of the most notable problems I expect to encounter further along development is inconsistencies between GUI frameworks. It may prove challenging to overcome these issues in a graceful approach.

The current prototype (see Figure 2) developed shows how an XML interface definition file can be processed and ultimately rendered within a sample application. This proves that the project concept is sound and can be reliably developed to the original specification with only minor, low level differences such as the use of XML schemas over Document Type Definitions (DTDs).

The prototype is built in Python using wxPython, a framework based on the popular C++ library wxWidgets. It loads and renders an XML GUI definition file containing a number of components (currently limited to buttons, text boxes, radio buttons, labels and group boxes) using the standard ElementTree XML package within Python. I will continue to expand on this prototype so that a larger set of components are available and rendering bugs (such as the poor positioning shown in Figure 2) are fixed. The code is currently located in a single file as opposed to separately in an API library. This is one of the next major steps in regards to the implementation. Once the Python library is complete I will implement a similar library in Java with the goal of rendering consistent user interfaces across libraries. The final stage of implementation will be the creation of the visual GUI designer. This will effectively be a visual "drag-and-drop" style application whereby a hierarchy of elements can be arranged.

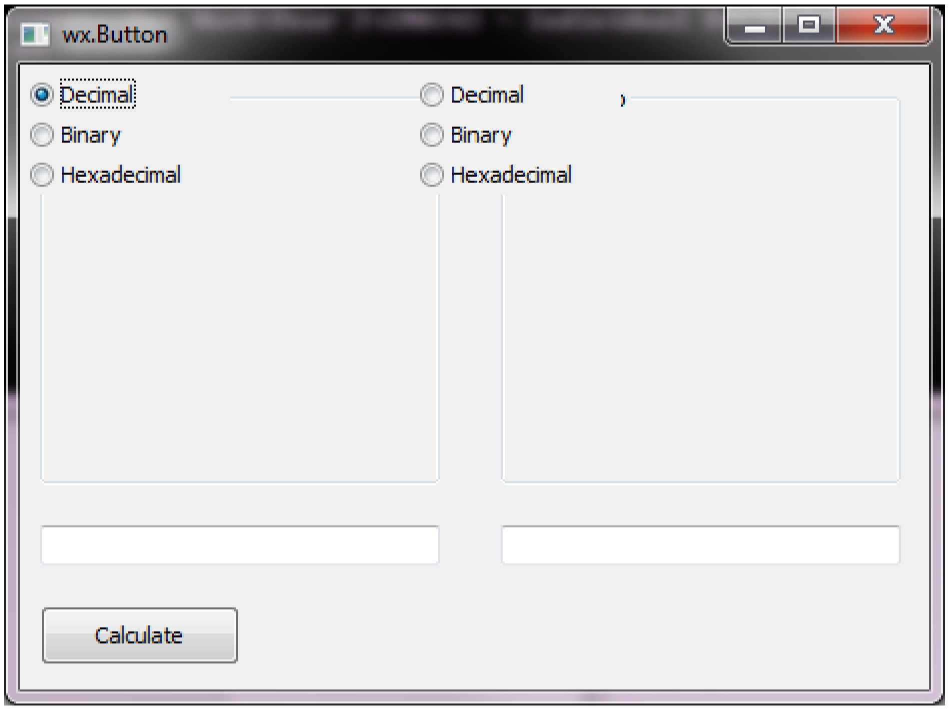

Figure 2: *Example prototype application rendered from an XML file using Python+wxWidgets.*

Conclusions

In summary the goals of the project are to create a set schema for defining a user interface using XML, create two API libraries which allow this interface to be loaded and rendered in an application and implement a visual GUI designer enabling users (both novice and experienced) to quickly and simply create an interface. Throughout simplicity, reusability and versatility will be the key themes. A simple schema, set of associated tools and working practice will be the ideal project outcome. The architecture of the system should promote interface reusability with use across multiple different platforms being a simple, trivial process.

I aim to create a high quality system which may solve some of the issues found within other existing products. The initial prototype demonstrates that the concept of the project is sound and that further implementation can continue in the manner outlined. Over the coming weeks I will complete the initial Python implementation as an API library and base the Java implementation upon this. Once complete I will develop the visual interface editor. I have a solid basis for development as well as a reliable development method (iterative and incremental) which should see the project through to completion to a high standard.

References

[1] “Windows Forms Designer,” Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/e06hs424(v=vs.80).aspx. [Accessed 14th December 2012].

[2] “Qt Creator IDE and tools,” Digia, [Online]. Available: http://qt.digia.com/Product/Developer-Tools/. [Accessed 14th December 2012].

[3] “What is Rapid Application Development?,” CASEMaker, 2000. [Online]. Available: http://www.casemaker.com/download/products/totem/rad_wp.pdf. [Accessed 14th December 2012].

[4] “A sneak peek into Spotify's secret labs,” Spotify Ltd, [Online]. Available: http://www.spotify.com/uk/download/previews/. [Accessed 14th December 2012].

[5] “Gecko,” Mozilla Foundation, [Online]. Available: https://developer.mozilla.org/en/docs/Gecko. [Accessed 14th December 2012].