Calum J. Eadie

Video Processing Language for the Raspberry Pi

Computer Science Tripos, Part II

Girton College

May 16, 2013

Proforma

Name:	Calum J. Eadie
College:	Girton College
Project Title:	Video Processing Language for the Raspberry Pi
Examination:	Computer Science Tripos, Part II, May 2013
Word Count:	11,980 words
Project Originator:	Dr A. Blackwell
Project Supervisor:	Dr A. Blackwell

Original Aims of the Project

My project aimed to (1) provide an educational environment for children to explore video processing on the Raspberry Pi and (2) provide an example to other developers in accessing the video processing capabilities of the Raspberry Pi through (a) determining a set of capabilities to control and (b) an API for them, (c) developing a visual language and (d) a development environment, and finally (e) evaluating the project with a small structured user study.

(74 words)

Work Completed

I have created the first video processing educational application for the Raspberry Pi (1). I carried out performance analysis of key video processing technologies, demonstrated current limitations, developed a video mixing API and contributed to existing open source projects (2). I implemented all core work items (a-e) and extended the project through (i) supporting external content sources, (ii) adding general-purpose computation, (iii) adding optimisations such as video caching, (iv) extending the evaluation to a large child user study (50 person) and (v) a controlled experiment (5 person), which obtained promising evidence of contribution towards learning outcomes.

(96 words)

Special Difficulties

None.

Declaration of Originality

I Calum J. Eadie of Girton College,
being a candidate for Part II of the Computer Science Tripos,
hereby declare that this dissertation and the work described in it
are my own work, unaided except as may be specified below, and
that the dissertation does not contain material that has already
been used to any substantial extent for a comparable purpose.

Signed

Date

Table of Contents

List of Figures

Chapter 1 - Introduction

1.1 Overview

My project involved creating a visual programming language and a development environment, through which children could explore video processing on the Raspberry Pi. All the core work items were completed and several extensions were implemented. Through early formative user testing and later evaluative structured user studies, the main success criterion has been achieved. This required the successful creation of sample programs by users, which demonstrate behaviours from a test suite of end user scenarios. The project has been evaluated in terms of usability problems, attainment of learning outcomes and performance. In the course of the project several open source contributions have been made to the Raspberry Pi community.

1.2 Background

The Raspberry Pi is a cheap credit card sized computer, developed with the intention to provide a platform for programming experimentation and teaching of basic Computer Science in schools. Critically for this project, the hardware is particularly suited for video processing, with comparatively weak general-purpose performance. Whilst the CPU is a modest 700 Mhz, 256 Mb chip, the GPU is capable of BluRay quality playback and fast 3D processing (Upton, 2013).

There are many examples of existing educational applications for assisting with exploration of Computer Science and programming. Some have been ported to the Raspberry Pi, of which a particularly notable example is MIT's SCRATCH (Maloney et al., 2010), a programming language learning environment. SCRATCH exposes the capabilities of the underlying machine through abstractions based on the physical environment of the user, with agents such as a cartoon cat and actions such as moving *n* steps. The user interacts with programmable characters whose behaviour is specified by composing blocks in a visual language.

There are currently few applications exploiting the Raspberry Pi's video processing capabilities and no educational applications. Although the hardware has the potential to achieve strong video processing performance, the Raspberry

Pi software platform is early in development and so there are high barriers to entry in creating video processing applications.

1.3 Main objectives

My project aimed to create an educational environment for children to explore video processing on the Raspberry Pi. More specifically, the project had the following main objectives.

Objective 1. To provide an environment for children to explore video processing by creating a visual programming language and a development environment for video processing.

As well as playing to the strengths of the Raspberry Pi, video processing was selected as an area with the potential to motivate children to experiment with programming and Computer Science.

A *visual*, rather than *textual*, language design was selected based on the success of visual languages such as SCRATCH (Maloney et al., 2010) (Wilson & Moffat, 2010) and ALICE (Conway et al., 2000) for similar design needs.

Objective 2. To provide an example to other developers in accessing the video processing capabilities.

By providing such an example, my project aimed to reduce the currently high barriers to entry in creating video processing applications on the Raspberry Pi.

Chapter 2 - Preparation

In this chapter, I further analyse the project requirements and discuss how the project was planned to mitigate key risks. I then discuss investigation into video processing on the Raspberry Pi, including performance analysis of key technologies. I follow this by applying Human Computer Interaction theory to the language design.

2.1 Requirements analysis

2.1.1 Work items

To refine the main objectives (§1.3) I decided on a set of core and extension work items. As the Raspberry Pi platform is early in development there were many areas with significant unknowns and so the planning needed to be flexible. This motivated an adaptive software development methodology where general work items were refined throughout the preparatory stages of the project.

I identified the following core work items.

Core 1. Determining a set of capabilities that the end user could create programs to control.

Core 2. Developing an API to access the capabilities.

Core 3. Designing a visual language syntax.

Core 4. Developing an editor for the visual language.

Core 5. Developing an interpreter for the visual language.

Core 6. Performing a small structured user study with adults.

I also identified the following possible extensions.

Extension 1. Extending the small structured user study to a large user study with children.

Extension 2. Adding the ability to use external content sources.

Extension 3. Creating an environment where the user can explore data structure and algorithms through analogy to structure of the content source.

Extension 4. Adding dynamic composition of programs.

2.2 Planning

2.2.1 Capabilities

The video processing capabilities were a significant unknown; so to mitigate risk I proposed four potential capabilities that I would go on to investigate. These are explained below in decreasing order of preference.

Capability 1. **Live video transformation.** This would allow the user to programmatically apply transformations such as colour manipulation and spatial scaling to videos in real time.

Capability 2. **Video mixing.** This would allow the user to programmatically combine clips from many videos into one video.

Capability 3. **Compositing video data with 2D and 3D animation.** This would allow the user to programmatically control the overlay of animation onto a single video.

Capability 4. **3D animation without video.** This would allow the user to programmatically control 3D animation.

2.2.2 Risks

I identified the three areas with the most risk, which were therefore most likely to cause changes to project design.

Risk 1. **The video processing capabilities of the Raspberry Pi.**

This was relevant for choosing between the options for programmable capabilities, as it determined what level of performance the platform supported.

Risk 2. **Selecting and accessing external content sources.**

Capability options (1), (2) and (3) require video. The Raspberry Pi has limited storage for local video and interacting with the files would require a conceptual model of the file system that was undesirable to require in this project.

Risk 3. **Designing and implementing a novel visual language.**

This was relevant for choosing between the options for programmable capabilities, as it determined which capabilities could be represented by a language designed for users with little to no programming experience.

2.2.3 Next steps

In the following sections I discuss the work I undertook to mitigate the risks I had identified. For risk (1) I investigated the basic and advanced video processing capabilities. For risk (2) I investigated external content sources and for risk (3) performed preparatory language design.

2.3 Technology investigations

2.3.1 Investigating candidate video processing technologies

In §1 I discussed the high barriers to entry for developers creating video processing applications. Specifically, there were limited documentation and examples available when this project started. To understand which technologies were candidates for providing the required capabilities I surveyed the software architecture of the Raspberry Pi.

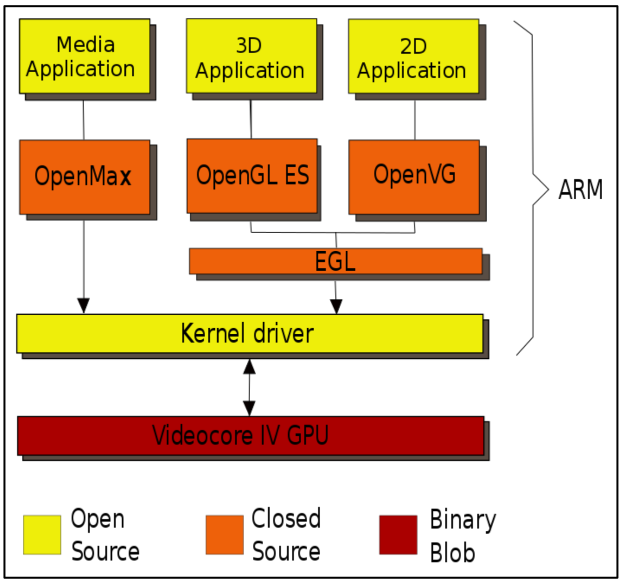


Figure 1: Raspberry Pi Software Architecture (Wirepath, 2012, reproduced with permission)

I found that OPENMAX provides processing of audio, video and still images whilst processing of 3D graphics is provided by OPENGL ES, a subset of OpenGL for embedded systems. OPENGL ES includes GLSL, a shading language, which was of particular interest at this stage as it had the potential to support live video transformation.

In the following sections I investigate basic and advanced video processing capabilities, showing that GLSL can perform video transformations. I go on to show that due to lack of support for OPENMAX in higher-level frameworks it was not feasible to use live video transformation in the project.

2.3.2 Investigating basic video processing capabilities

I began by investigating the performance of video playback without manipulation. I surveyed available video players and found OMXPLAYER was the only one that could theoretically play high definition video, as it was unique in being able to access the GPU through OPENMAX.

The Raspberry Pi uses a Broadcom BCM2835 processor, which supports hardware decoding of high definition H.264 format video (Broadcom, 2013). This investigation set out to verify whether OMXPLAYER could access this capability and whether other formats were supported.

2.3.2.1 Method

I created a test suite that allowed me to play videos through OMXPLAYER. For each video I measured the time taken between starting playback and reaching a marker at 20 seconds into the video. This gave a measure of video playback performance.

2.3.2.2 Discussion

The tests verified that OMXPLAYER achieved strong performance in playback of high definition (1280x720) videos, using the H.264 video format with either an MP4 or FLV container. Tests of playback using the VP8 format suggested OMXPLAYER was not capable of VP8 playback. Further research confirmed that support for VP8 on the GPU was experimental.

The conclusion from this experiment was that video mixing, that is, video playback without manipulation of the video, was feasible.

2.3.3 Investigating advanced video processing capabilities

After showing that playback of high definition video was possible using OMXPLAYER I went on to investigate whether live video transformation was also feasible.

Transforming video was a particularly unexplored area so I discussed options with core Raspberry Pi developers. We proposed using OPENMAX to decode H.264 encoded video on the GPU and then to use OPENGL to render the decoded video frames onto an EGL surface. To transform the video frames GLSL shaders would be applied to the EGL surface, for example a shader could be used to stretch the video frame or change the distribution of colours. This approach is illustrated in Figure 2.

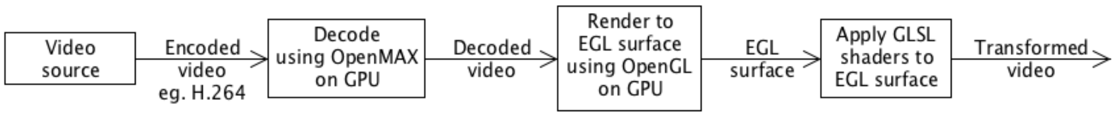

Video source
Encoded video eg. H.264
Decode using OpenMAX on GPU
Decoded video
Render to EGL surface using OpenGL on GPU
EGL surface
Apply GLSL shaders to EGL surface
Transformed video

Figure 2: Proposed data flow for video transformation using OpenMAX and OpenGL

I surveyed potential options for implementing this approach and found within the project time constraints I would not be able to develop directly on top of OPENMAX and OPENGL. Specifically, from the discussions with the core Raspberry Pi developers, there is limited documentation for the Raspberry Pi implementations of OPENMAX and OPENGL, which deviate from the standards for these technologies. Therefore, this approach involved too much risk given my graphics and OPENMAX/OPENGL experience. I found that JOGAMP, a JAVA binding, and QT5 with QML provided feasible interfaces to OPENGL. I selected JOGAMP based on superior documentation and availability of examples of transforming video.

2.3.3.1 Method

I selected MOVIECUBE, an example that was highly representative of the use cases for the project, to analyse. MOVIECUBE involved a 3D cube with a video projected onto each face. The projections were transformed by colour transformations and the cube could be rotated by user input so the transformations were necessarily live.

I followed a similar approach as §2.3.2.1, using a test suite to repeatedly play a video through the MOVIECUBE example and measuring performance with a time interval. I measured the time taken from spawning the MOVIECUBE process to reaching 20 seconds into the video.

2.3.3.2 Results

Container format	Video format	Audio format	Resolution / pixels^2	Time until 20s marker reached / seconds	Video quality
MP4	H.264	AAC	1280x720	155	Good
WEBM	VP8	Vorbis	1280x720	130	Good
FLV	H.264	AAC	854x480	74	Poor
WEBM	VP8	Vorbis	854x480	70	Poor

Table 1: Excerpt from results of MovieCube performance tests.

2.3.3.3 Discussion

The experiment showed that the MOVIECUBE example was only able to achieve real time performance of live video transformation for very low resolutions (144x80 pixels).

I studied the source code to understand if this was a limitation of the example or of JOGAMP as a whole. I found that the current release of JOGL did not support using OPENMAX for video decoding; therefore this was a fundamental limitation in the current version of JOGAMP. The source code showed that video was being decoded by FFMPEG on the CPU, rather than OPENMAX on the GPU. The difference between this and the proposed approach is illustrated in Figure 3.

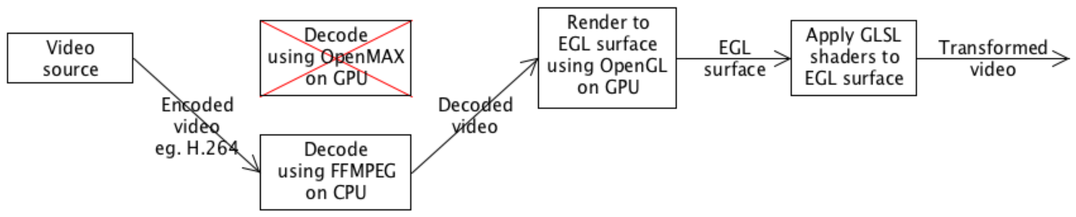

Video source
Encoded video eg. H.264
Decode using OpenMAX on GPU
Decode using FFMPEG on CPU
Decoded video
Render to EGL surface using OpenGL on GPU
EGL surface
Apply GLSL shaders to EGL surface
Transformed video

Figure 3: Actual data flow for video transformation using JogAmp.

Studying comments in the JOGL source code indicated that, whilst there was a partial implementation of OPENMAX support in JOGAMP, it would have been too risky to attempt to extend it for this project.

```
/**
 * OpenMAX IL implementation. This implementation is currently not
 * tested due to lack of an available device or working
 * <i>software</i> implementation. It is kept alive through all
 * changes in the hope of a later availability though.
 */
public class OMXGLMediaPlayer extends EGLMediaPlayerImpl {
```

Extract 1: Evidence of partial OpenMAX support in JogAmp.

The conclusion from this experiment was that, as JogAmp does not currently support using OpenMAX to provide hardware video decode, live video transformation was not feasible for this project.

2.3.4 Investigating external content sources

I set out to find an external content source that would be able to provide video content for the user to programmatically manipulate using capability (1), (2) or (3) [§2.2.1]. The source needed to provide video in a format that can be played on the Raspberry Pi. It also needed to be familiar to users and feasible to access.

2.3.4.1 Method

I chose YouTube as the most preferred candidate for an external content source as it has the largest market share of all online video providers (comScore, 2010) and so would be a familiar domain for users.

To test whether YouTube provided video in a compatible format and was feasible to access I studied a text user interface YouTube client, yt.

2.3.4.2 Discussion

I demonstrated selecting YouTube video content through the yt interface and playing the content using OMXPlayer. This showed YouTube provided video in a format compatible with OMXPlayer and therefore was an appropriate external content source for video.

Studying the yt code showed it was feasible to access YouTube content through their APIs. This also showed that the YouTube API provided access to information about the videos that could feasibly be used to explore basic data structures and algorithms.

I published the extensions to `YT` as open source contributions, which have been well received by the Raspberry Pi community.

The conclusion from this experiment was that YouTube was a suitable external content source.

2.3.5 Conclusions

In the technological investigations, I looked at four possibilities for Raspberry Pi capabilities that the language could be designed to control. I came to the following conclusions.

- The preferred capability, live video transformation, was not feasible however the next most preferred, video mixing, was feasible.

- YouTube was a feasible external content source. In demonstrating this I made open source contributions, which were well received by the Raspberry Pi community.

2.4 Language Design

A key objective of this project was to create an environment that would allow users with little to no programming experience to create programs. In the section I will explain how I applied theories, processes and techniques from HCI and Interaction Design to this problem.

2.4.1 Computational concepts

From investigating the video capabilities, I had identified that the video processing would have to be restricted to playing segments from videos and varying speed and sound. With the opportunity for programmatically controlling video processing reduced, I chose to make the environment more compelling by expanding into general-purpose computing.

Wing (2006) highlights *computational thinking* as a fundamental skill, important in disciplines outside of Computer Science. This perspective has influenced current thinking on Computer Science education, including reforms of primary and secondary education in the UK (Department of Education, 2013).

I considered which core computational concepts should be included based on the original paper by Wing and a complementary formulation from the International

Society for Technology in Education (ISTE, 2011). I then considered how these general concepts could be explored through manipulation of video segments.

Concept 1. Sequencing, doing one step after another

This was involved through the ordering of video segments.

Concept 2. Selection, doing one thing or another

This was involved through selecting one video segment or another.

Concept 3. Repetition

This was involved by repeatedly playing a sequence of video segments.

2.4.2 Conceptual models

Johnson and Henderson (2002, p. 26) describe a *conceptual model* as "a high-level description of how a system is organised and operates". For example, when a professional software developer is programming they may use several conceptual models. When programming in a functional language they may use a conceptual model based on a program execution model of functions as the basic control structure. At a different level of abstraction, they may also use a conceptual model in terms of the fetch execute decode cycle and knowledge of the computer architecture.

Rogers et al. (2011, p. 35) recommend developing a conceptual model before making detailed design decisions about the end product, in this project how the user would manipulate a description of a program in an editor. This is particularly important in this project, as the computational concepts involved will be novel and challenging for the users.

2.4.3 Motivation for a conceptual metaphor

As users would not have experience of computational concepts, I used a common technique called *conceptual metaphor* to help them understand programming in terms of familiar concepts. A successful example of conceptual metaphor is the spreadsheet. Spreadsheet applications are particularly interesting; through using the spreadsheet conceptual metaphor a user is supported in achieving sophisticated computational tasks, with limited programing experience required (Ko et al., 2011).

Before VisiCalc made electronic spreadsheets well known in the late 1970s and early 1980s, paper based spreadsheets were in wide use. They supported manual calculation through a regular grid arrangement of values and modification of those values through pencil markings and erasing.

Electronic spreadsheets use the conceptual metaphor of a paper spreadsheet through an analogous structure of scalar values in a grid arrangement with mathematical relationships between the values. This conceptual metaphor supports more complex activities, such as automatic calculation and macro recording, which are outside the conceptual model of a paper spreadsheet.

2.4.4 Selecting a conceptual metaphor

To select a conceptual metaphor, a mapping that would help users understand the computational concepts in terms of familiar concepts, I reviewed existing environments and notations from the domain of video manipulation. This approach is recommended by Rogers et al. (2011, p. 400) for developing a conceptual metaphor.

This included reviewing:

- **Scratch**: A visual programming language learning environment from the MIT Media Lab.
- **Screenplays and storyboards**: Complementary notations for describing videos.
- **iMovie and Final Cut Pro**: Desktop video editing environments.
- **Mozilla Popcorn Mixer**: An environment for combining segments of YouTube videos with other forms of media.

The review suggested that creating a movie script would be a good conceptual metaphor for programming. I will expand on this in the next section.

2.4.5 Selecting an interface metaphor

An *interface metaphor* is a metaphor that is instantiated in the user interface of a system (Rogers et al., 2011, p. 44). In this project the editor is the user interface for manipulating the language and the language the interface for controlling the Raspberry Pi. In a similar way to how the visual representation of a spreadsheet as a grid of values in an electronic spreadsheet is a metaphor for a familiar paper spreadsheet, I developed an interface metaphor for the language that helped the user understand how a program could be manipulated by analogy to familiar interaction with screenplays.

Erickson (1990) suggests a set of questions to ask of potential interface metaphors. I will discuss some of these questions below to explain why it was a good candidate.

Question 1. How much structure does the metaphor provide?

Screenplays have hierarchical structure at the level of acts, scenes and lines. This maps to the hierarchical structure of programs. They are read out by actors sequentially, which maps to the execution model of single threaded programs.

Question 2. Will your audience understand the metaphor?

Whilst it is difficult to reason about the specific literary experiences a child will have, many Drama and Language/Literature syllabuses include exposure to the internal structure of plays and films. Therefore it is feasible that a child may be able to relate the hierarchical structure of a program and its sequential execution model back to the structure of a play they have studied and the order in which actors performed speech and actions.

Question 3. How extensible is the metaphor?

Screenplays typically do not include selection and repetition however variations such as “choose your own adventure” books, where the reader chooses between many possible narratives, show that the format is extensible.

2.5 Prototyping visual design of the language

2.5.1 Initial prototype

I followed established practice by creating *low fidelity prototypes* of the visual representation of the language using sketches (Buxton, 2010). This approach is useful as it encourages criticism and is easy to modify.

To prepare for this I researched *visual representation*, the principles by which markings on a surface are made and interpreted. The key outcomes where that visual representation can be very specific to particular domains, and that improving on conventions requires significant skill (Blackwell, 2011). Hence, in this project I concentrated on applying existing conventions from the domains of video manipulation, screenplays and YouTube.

2.5.2 Refining the prototype

To refine the visual language design I applied the *Cognitive Dimensions of Notations* framework, introduced by Green (1989), with a group of experienced designers. This is a *formative analytic evaluative technique*. The framework supports discussion of *trade offs* involved in the design of a notation, in a similar way to how a processor designer must trade off between size, cost, and performance. A processor designer cannot shrink a processor indefinitely without affecting cost or performance. Similarly, the language design could not be optimal in all dimensions.

To illustrate how the framework can be applied, I will discuss iteration over a selection of prototypes for expressing control flow.

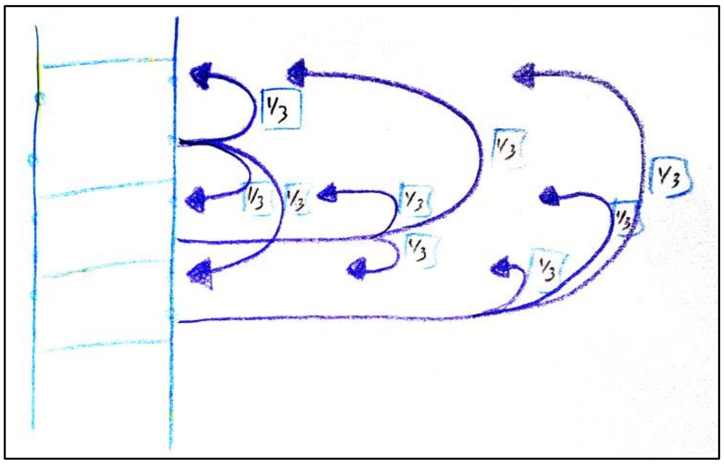


Figure 4: Early prototype, expressing control flow using node and link representation.

In an early prototype (Figure 4), I used a *node and link* representation for control flow. In this particular example the control flow is probabilistic. Jumps are made from one section of the program to another based on a specified probability distribution over possible control flows. In terms of Cognitive Dimensions of Notation there is low *hidden dependency*, as the control flow is explicitly represented. In order to view the control flow easily, described as having high *visibility*, the notation uses a lot of space. However, using a lot of space, high *diffuseness*, leaves less space for representing other aspects of the program.

In a later prototype (Figure 5), I traded off *diffuseness* and *viscosity*, by reducing the amount of space required through sacrificing how easily changes can made. Specifically, by using a containment structure rather than the node and link representation of control flow from Figure 4, more manipulation is required to change the control flow. This was an overall improvement, as on the whole gaining the space was worth slightly more resistance to change.

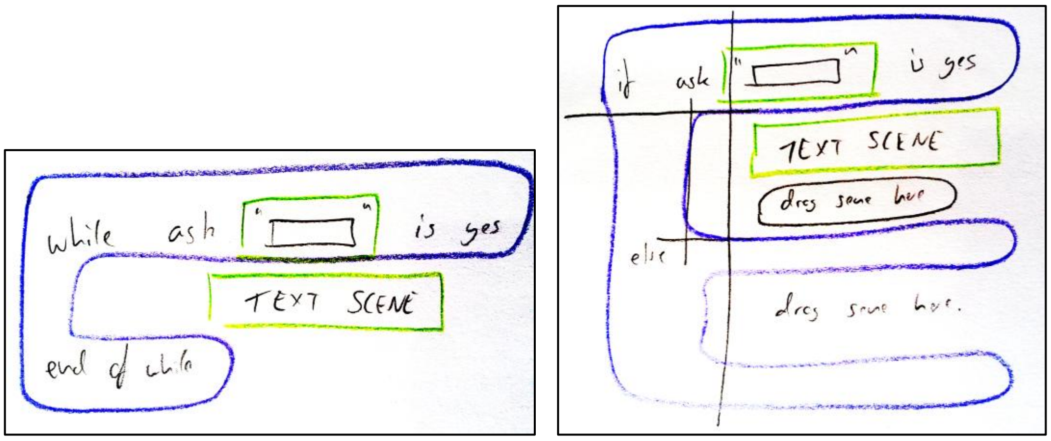


Figure 5: Later prototype, expressing control flow through regions and shapes.

2.6 Summary

In this chapter, I further analysed the project requirements and investigated areas of key uncertainty to help the implementation go smoothly. I investigated video processing on the Raspberry Pi to decide between potential capabilities for the language to control. Through performance analysis of key technologies I showed that the more ambitious live video transformation capability was not feasible, as JogAmp does not currently support hardware video decoding on the Raspberry Pi. I then selected video mixing as the next most preferable capability. I followed this by discussing key theory from Human Computer Interaction and applied that theory to the language design. In particular, I selected the conceptual metaphor of a creating a movie script to help users understand programming in terms of familiar concepts from the domain of video manipulation.

Chapter 3 - Implementation

In this chapter, I describe the implementation of the language as a *domain-specific* language through a *piggybacking* approach, translating to PYTHON code and adding domain-specific capabilities with APIs. I discuss the use of *parse tree* and *widget tree* data structures, and algorithms to translate between them and Python code. I then describe how the editor was implemented, including extending Qts drag-and-drop capability to support *type enforcement* and *type hinting*. I follow this by discussing the implementation of domain-specific capabilities; including accessing the video processing capabilities, open source contributions and extending the project through performance optimisations.

3.1 Walk through

I will begin this chapter by briefly describing a use case based on the activities that participants performed in the user studies (§4.2, §4.3).

Alice is a 13-year-old child. She regularly watches videos on YouTube and has acted in her school's production of Twelfth Night. She is currently studying the script of A Midsummer Night's Dream in her English classes. Like most children her age, although she is taught ICT at school, she has not been exposed to programming or Computer Science.

Alice is asked to reorder a video by playing the second half before the first half. She starts with an empty script.

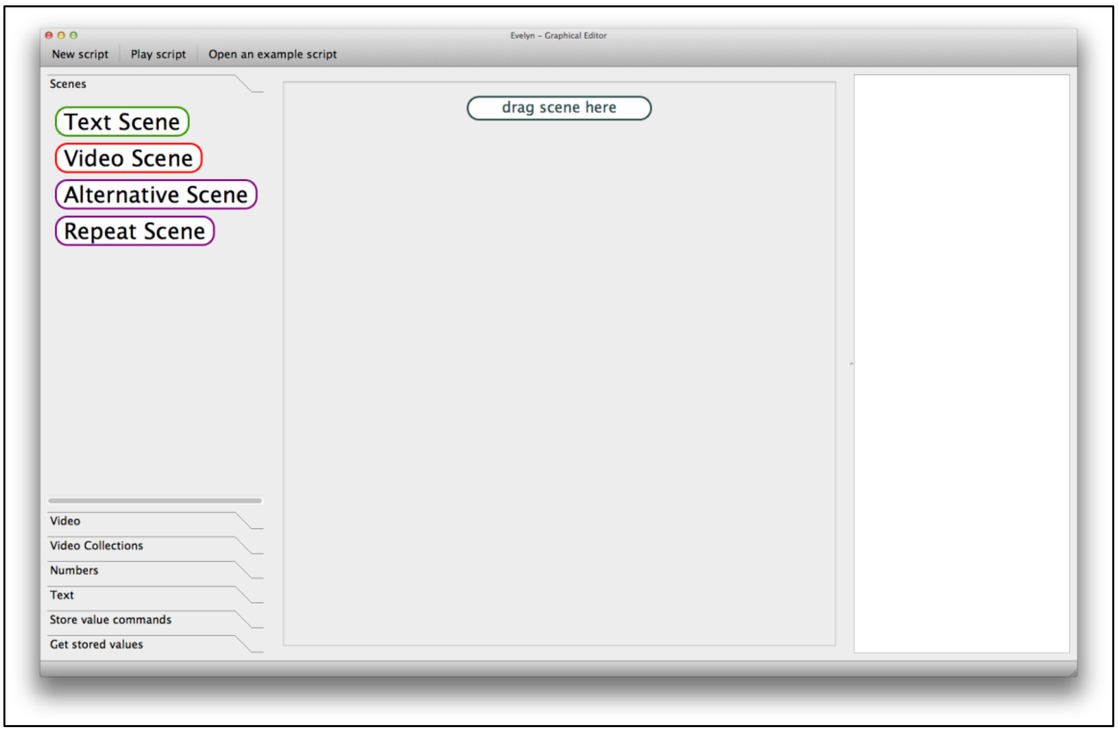
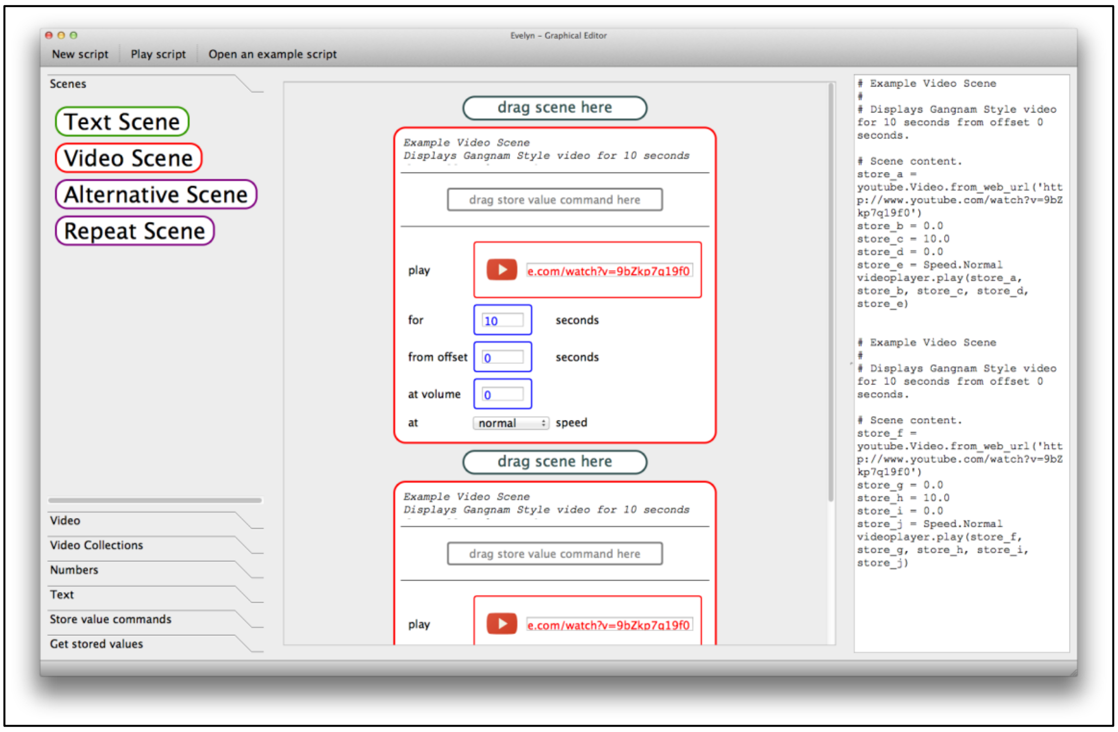


Figure 6: Screenshot of editor showing an empty script.

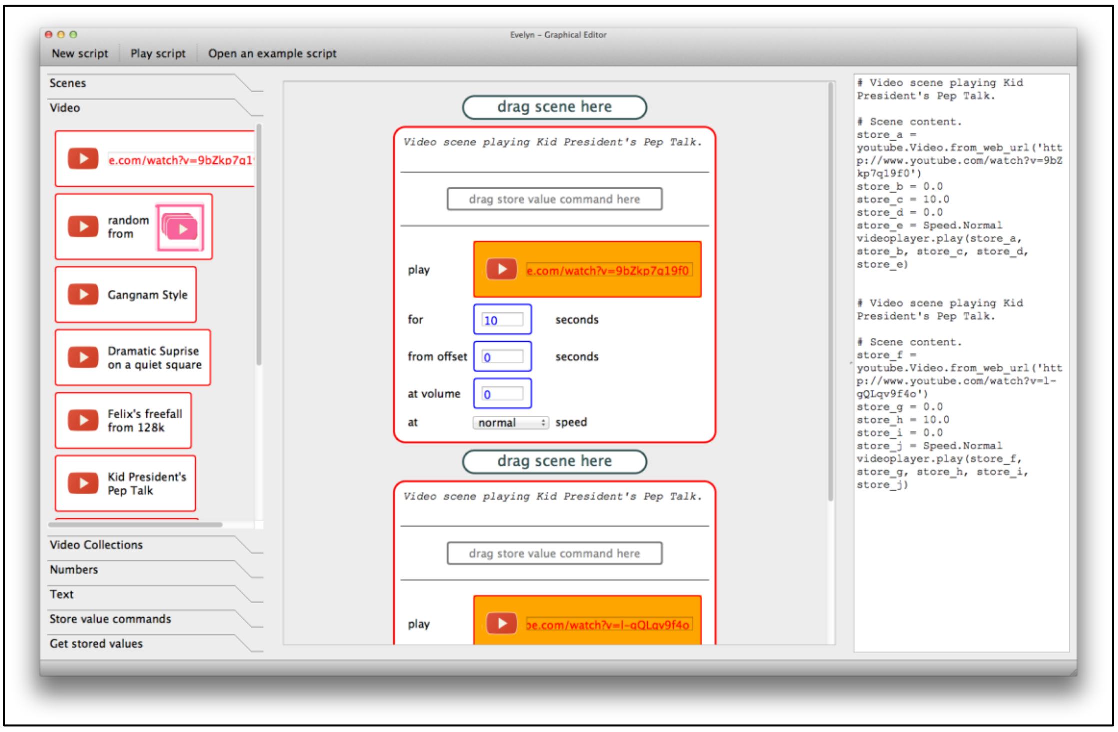
Alice has been told that creating a computer program using the editor is similar to creating a film script. She remembers from her acting and English classes that films and plays are divided into scenes that follow one after another. She wonders how she will use the editor to create the video that she has been invited to create.



```
# Example Video Scene
#
# Displays Gangnam Style video
for 10 seconds from offset 0
seconds.

# Scene content.
store_a =
youtube.Video.from_web_url('htt
p://www.youtube.com/watch?v=9bZ
kp7q19f0')
store_b = 0.0
store_c = 10.0
store_d = 0.0
store_e = Speed.Normal
videoplayer.play(store_a,
store_b, store_c, store_d,
store_e)


# Example Video Scene
#
# Displays Gangnam Style video
for 10 seconds from offset 0
seconds.

# Scene content.
store_f =
youtube.Video.from_web_url('htt
p://www.youtube.com/watch?v=9bZ
kp7q19f0')
store_g = 0.0
store_h = 10.0
store_i = 0.0
store_j = Speed.Normal
videoplayer.play(store_f,
store_g, store_h, store_i,
store_j)
```

Figure 7: Screenshot of editor showing sequential video scenes.

Alice is not sure what her script will do, so she clicks play script and views two ten second clips from a popular music video, played one after the other. She realises the video corresponding to a YouTube URL is being played, and drags a different video value from the palette into the video slot of the video scenes. As she drags the "Kid President's Pep Talk" video value across, the video slots in the video scenes highlight to indicate matching types. She also adjusts the number values in the video scenes to play the second half of a video before its first half.



Video scene playing Kid President's Pep Talk.

play	e.com/watch?v=9bZkp7q19f0	
for	10	seconds
from offset	0	seconds
at volume	0	
at	normal	speed

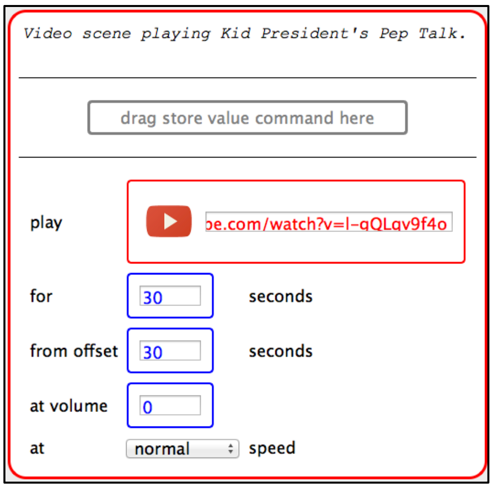
Video scene playing Kid President's Pep Talk.

play	be.com/watch?v=l-gQLqv9f4o

```
# Video scene playing Kid
President's Pep Talk.

# Scene content.
store_a =
youtube.Video.from_web_url('htt
p://www.youtube.com/watch?v=9bZ
kp7q19f0')
store_b = 0.0
store_c = 10.0
store_d = 0.0
store_e = Speed.Normal
videoplayer.play(store_a,
store_b, store_c, store_d,
store_e)


# Video scene playing Kid
President's Pep Talk.

# Scene content.
store_f =
youtube.Video.from_web_url('htt
p://www.youtube.com/watch?v=l-
gQLqv9f4o')
store_g = 0.0
store_h = 10.0
store_i = 0.0
store_j = Speed.Normal
videoplayer.play(store_f,
store_g, store_h, store_i,
store_j)
```

Figure 8: Screenshot of Alice changing dragging video value onto a video value slot in a video scene.

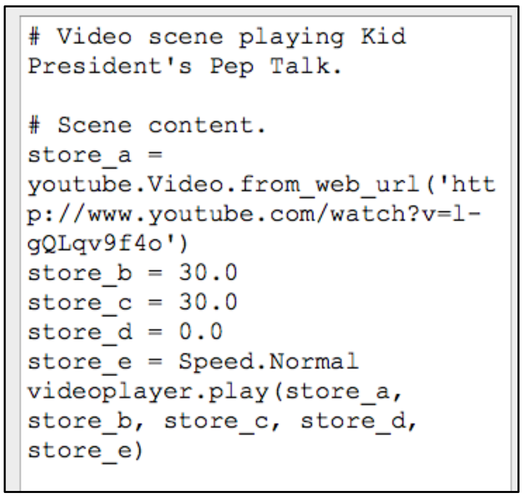
Alice notices there is a correspondence between the script content she has been manipulating and some text next to it, which appears strange to her. Alice is told she has been manipulating a visual representation of her script and that the strange text is a textual representation in a programming language called PYTHON.



Video scene playing Kid President's Pep Talk.

drag store value command here

play	be.com/watch?v=l-gQLqv9f4o	
for	30	seconds
from offset	30	seconds
at volume	0	
at	normal	speed



```
# Video scene playing Kid
President's Pep Talk.

# Scene content.
store_a =
youtube.Video.from_web_url('htt
p://www.youtube.com/watch?v=l-
gQLqv9f4o')
store_b = 30.0
store_c = 30.0
store_d = 0.0
store_e = Speed.Normal
videoplayer.play(store_a,
store_b, store_c, store_d,
store_e)
```

Figure 9: Screenshots highlighting correspondence between visual and textual representations of a computer program in the editor.

3.2 Overall architecture

In this section I will give an overview of the architecture of the system. At a high level the system is made up of:

Component 1. Editor

The editor is a graphical application and provides the environment through which users can manipulate scripts in the language.

Component 2. Language

The language is the means through which users control the capabilities of the Raspberry Pi. It uses the conceptual metaphor of a movie script to help users with little to no prior programming experience understand computational concepts through analogy to familiar concepts (§2.4). The language is visual and realised through the interface of the editor.

Component 3. Capability APIs

The capability APIs provide access to the capabilities of the Raspberry Pi. Live video transformation was shown to be infeasible (§2.3.3), so the video API is focused on video mixing. The APIs also provide access to YouTube content, which has structure that allows computational thinking to be explored. The APIs include optimisations to improve video playback performance.

This project makes use of existing systems, notably QT, the PYTHON interpreter, OMXPLAYER and YOUTUBE-DL. Furthermore the project extends some existing systems, including PYOMXPLAYER. The system architecture, including use and extension of existing systems, is illustrated in Figure 10.

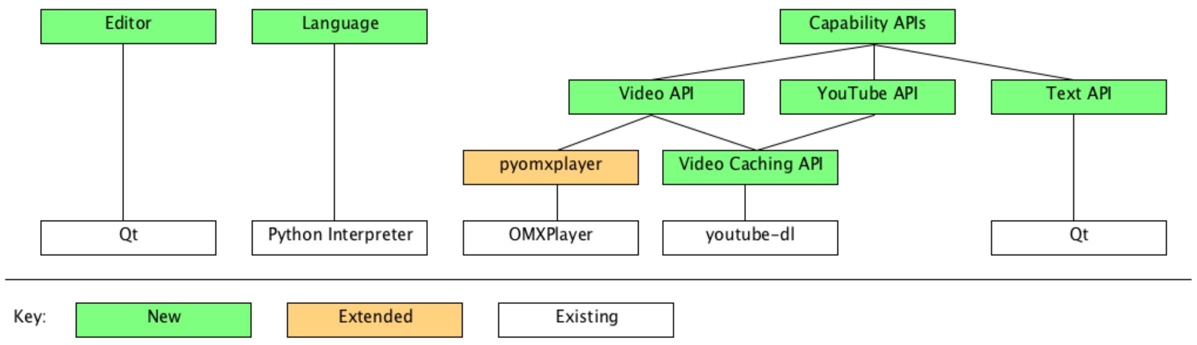


Figure 10: System Architecture.

3.3 Development Environment

3.3.1 Language

I developed all components in PYTHON, which I selected for several reasons. Firstly, I was familiar with the language, having used it in a professional context. Secondly, the second main objective of this project is to provide an example to other developers in accessing the video processing capabilities. PYTHON is popular within the Raspberry Pi community and in particular is well supported as an educational language on the Raspberry Pi (Upton, 2013). Therefore, it was a good choice for creating examples for experienced and developing developers. Finally, libraries providing basic access to the capabilities required existed for PYTHON and were extensible.

3.3.2 Unit testing and API documentation generation

I used unit testing throughout the project. This allowed me to use *test-driven development*. I followed the *reStructuredText* convention for writing *docstrings* in PYTHON, which allowed me to use automated *API documentation generation*. This was particularly useful in the later stages of the project for visualising relationships between classes.

3.3.3 Version control and backups

I used hosted GIT repositories for storing all project files, including code and reports. In addition to using hosted version control, I used the file hosting service DROPBOX to provide greater assurance.

3.4 Language

I have developed a *domain-specific language* (DSL). The language is tailored towards the specific application domain of video mixing and to the requirements of users with little to no prior programming experience. I will describe the language implementation in terms of the framework and DSL development patterns described by Mernik et al. (2005).

3.4.1 Analysis

I analysed the use of terminology in the target domains of film scripts and YouTube to determine appropriate terms for use in the language. For example, in the editor rather than using terms such as "run" or "execute" for the action of executing a script I chose the term “play”. Though the former terms would be very appropriate in an editor for professional programmers, their interpretation in the domain of film scripts and YouTube would be quite inappropriate. I also analysed the visual representation used in these domains and incorporated elements, such as a symbol used to represent a video in the YouTube domain, into the notation.

3.4.2 Design

Mernik et al. (2005) characterise approaches to DSL design along two orthogonal dimensions. Firstly, the extent to which a design relates to an existing language and secondly, the extent to which a design is formally specified.

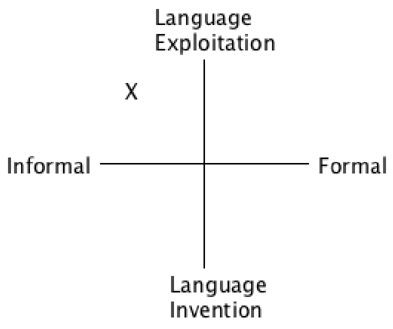


Figure 11: Language design approaches.

In this project I have *piggybacked* domain-specific features on a subset of the PYTHON language, an approach described by (Spinellis, 2001) and (Mernik et al., 2005). PYTHON provides the core language features of control flow and variables, whilst I have developed APIs which provide the domain-specific capabilities of video playback and YouTube interaction. I chose this approach for two reasons. Firstly, ease of implementation. It was not necessary to reimplement such well understood features. As stated by Wile (2003), “you are almost never designing a programming language”. Secondly, the correspondence to PYTHON allowed users to see their scripts expressed in the domain-specific visual notation of the project and in PYTHON code at the same time.

By providing both domain-specific and general-purpose programming language representations, I have taken an alternative approach to the current market leaders. In contrast, in products such as SCRATCH a user only interacts with a domain-specific representation and has no way to see their program in a general-purpose language. The significance of this approach is discussed further in §5.2.

As I will describe in the next section, I used *source to source* translation and so the language mirrored PYTHON's semantics. The semantics were described informally through correspondence to PYTHON's semantics, so the semantics of a construct in the language are the semantics of the PYTHON code it is translated into.

3.4.3 Implementation

Spinellis (2001) and Mernik et al. (2005) describe several implementation patterns; I considered between the *compiler* and *interpreter* patterns and chose the compiler pattern.

- **Compiler pattern**: Constructs from a DSL are translated to base language constructs and library calls before execution. At run time, the resultant base language code is then executed by the interpreter or compiler for the base language. By translating from a DSL to a general-purpose language this is a form of source-to-source translation.

- **Interpreter pattern**: At run time, constructs from a DSL are interpreted by a program written in the base language.

I chose the compiler pattern as the resultant PYTHON code could be shown in the interface and provide users with both representations of their scripts.

In a project with greater real time performance needs the interpreter pattern may be preferable as it can be used to provide domain-specific features. For example, whilst the language is using video mixing as a motivational use case for education, a DSL could also be used for professional video mixing. Such an application may require real time performance and may have to use speculative execution and branch prediction in order to buffer videos. These optimisations have to be applied at run time and would necessitate applying the interpreter pattern over the compiler pattern.

3.4.4 Types

I used a four-type system: *videos*, *video collections*, *numbers* and *text*.

I chose to combine integers and floating-point numbers into one type and apply implicit type conversions. Whilst understanding of number representation is a desirable learning outcome, in this project I am focusing on more general computational thinking principles. This decision relates to a core debate in the design and evaluation of educational programming languages, characterised as *programming-to-learn* versus *learning-to-program* (Mendelsohn et al., 1990) (Rode et al., 2003).

I chose to use a strong type system, that is, one with no unchecked run-time type errors, based on the approach adopted by Alice (Conway, 1997).

3.4.5 Parse tree data structure

I used a *parse tree* data structure as an intermediate representation between the editor, where the language is represented through user interface classes, and the interpreter, which operates on PYTHON code. This provided a useful separation of concerns between the user interface and language representation.

I used a class for each language construct. For example, the fragment below calculates half the duration of a video. It involves an operation to determine the duration of a video stored in a variable, an atomic value and an arithmetic operation to combine these two. In the parse tree, the fragment is represented by an object for each language construct.

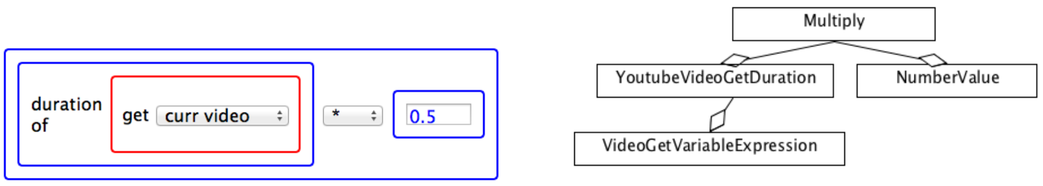


Figure 12: A language fragment as it is represented using widgets in the editor and the corresponding parse tree representation using a UML class diagram.

3.4.6 Translating the parse tree representation to Python code

I used a recursive strategy to translate from the parse tree representation into PYTHON code. Each class in the parse tree is responsible for translating the language fragment it represents into PYTHON code. In general, a class generates code for its own level and combines this with code generated by its children in the parse tree.

For example, the fragment in Figure 12 would translate to the PYTHON code "`curr_video.duration() * 0.5`". The translation algorithm for the MULTIPLY class roughly corresponds to substituting the code generated by its left and right operands into "`<left operand> * <right operand>`".

In this section I will highlight a few cases in greater detail.

3.4.6.1 Translating the root of the parse tree

The algorithm for translating the root of the parse tree is a special case as it handles initialisation of *user-defined* variables. To make the language more accessible to the target users I chose to allow expressions using the value of a variable to come before statements assigning a value to it. To translate into correct PYTHON code it was necessary to initialise user-defined variables to sensible default values for the particular type of variable.

I used the following algorithm to translate the root of the parse tree.

```
1   get names for all user-defined variables by recursively searching
    for variable references

2   initialise each user-defined variable to a value appropriate to
    its type

3   for each child scene:

3.1  translate the child scene using the translation algorithm for
     that particular type of scene

3.2  append the Python code for the child scene to the script to be
     outputted
```

Extract 2: Algorithm to translate root of parse tree into Python code.

To implement the algorithm each parse tree class implements a method to translate the subtree it parents into PYTHON code. Furthermore, each class implements a method to determine the names of all user-defined variables referenced in its subtree, by type.

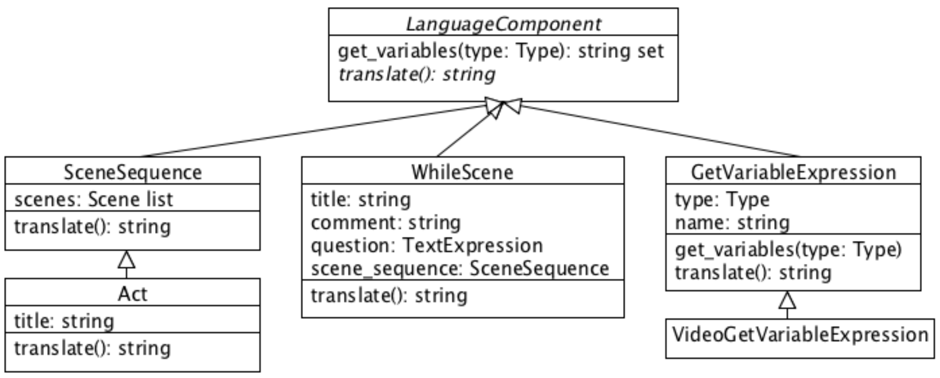


Figure 13: UML class diagram for a small subset of the parse tree classes.

The variable finding algorithm is recursive with two cases.

- **Recursive case:** Implemented by LANGUAGECOMPONENT, the base class for all parse tree classes. The variables are determined by taking the union of the names of all user defined variables, of the appropriate type, in the subtrees rooted at the children of the class.

- **Base case:** Implemented by classes representing variable use. If the type matches, the set of variables for that class is the one element set consisting of the name of the variable represented by the class. Otherwise, the empty set is returned.

3.4.6.2 Translating a repeat scene

I used the concept of a scene from a movie script as a conceptual metaphor for structure in a computer program. A video is played with a *video scene*, text is displayed with a *text scene* and control flow is controlled using *repeat* and *alternative* scenes for repetition and selection.

I used the following algorithm to translate repeat scenes, which provide conditional repetition of a sequence of scenes.

```
1   translate comments

2   create a Python while loop

2.1  loop condition: use API to ask user a boolean question

2.2  loop body: recursively translate the scenes in the body
```

Extract 3: Algorithm to translate a repeat scene into Python code.

The conditional repetition is implemented directly as a PYTHON while loop. The scene sequence being repeated is translated recursively and indented to follow PYTHON syntax.

3.4.6.3 Language execution

The language is executed by first generating a parse tree from the user interface and translating that into PYTHON code. The PYTHON code is then executed by the PYTHON interpreter, with the support of capability APIs to access the capabilities of the Raspberry Pi. This is illustrated in Figure 14.

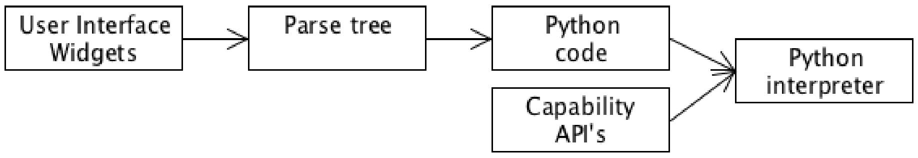


Figure 14: Execution pipeline from user interface widgets through to Python code.

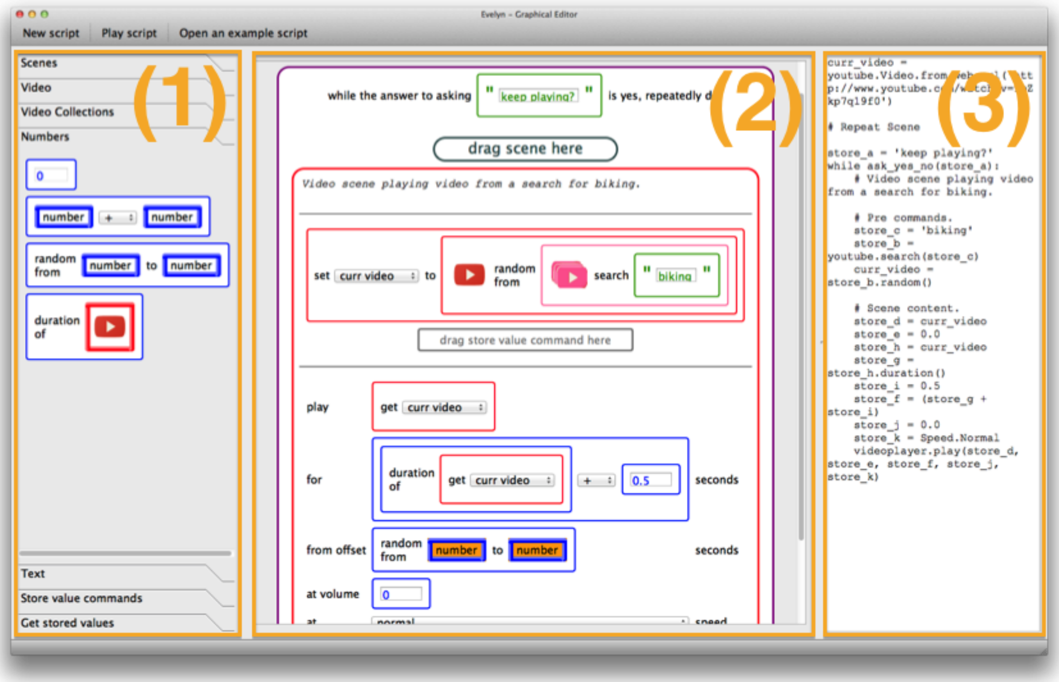
To execute the code I used PYTHON's support for dynamic execution of PYTHON code, provided by the EXEC statement. I used an execution environment isolated from the editor so that a user's program would not affect the reliability of the editor. In a non-isolated environment a user's program could affect the editor in ways such as changing a shared data structure. Separation was achieved using separate instances of system libraries and by using a separate global scope.

3.5 Editor

I implemented the editor using PYTHON and the PYSIDE PYTHON bindings for QT. The key components of the editor are:

1. A palette of available language components.
2. An area for manipulating the current script.
3. A view of the PYTHON code corresponding to the current script.

Users manipulate scripts by dragging language components from the palette onto slots in the manipulation area. This form of interaction allows for types to be enforced and combinations of blocks to be suggested.



```
curr_video =
youtube.Video.from_web[illegible]('htt
p://www.youtube.c[illegible]/w[illegible]tc[illegible]v=[illegible]Z
kp7q19f0')

# Repeat Scene

store_a = 'keep playing?'
while ask_yes_no(store_a):
    # Video scene playing video
from a search for biking.

    # Pre commands.
    store_c = 'biking'
    store_b =
youtube.search(store_c)
    curr_video =
store_b.random()

    # Scene content.
    store_d = curr_video
    store_e = 0.0
    store_h = curr_video
    store_g =
store_h.duration()
    store_i = 0.5
    store_f = (store_g +
store_i)
    store_j = 0.0
    store_k = Speed.Normal
    videoplayer.play(store_d,
store_e, store_f, store_j,
store_k)
```

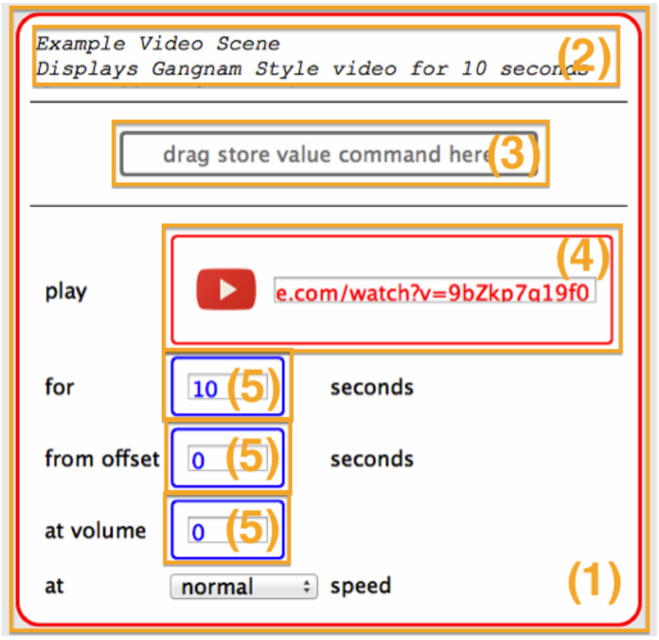
Figure 15: Screenshot of the editor showing (1) palette, (2) script manipulation area and (3) Python code view.

3.5.1 Implementing language components in Qt

The QT graphical framework organises elements of the user interface into widgets. I used a widget for each language component. This simplified translation between the widget and parse tree representations, as the structure was similar.

To implement language components I used a combination of approaches. For simple constructs, such as the value of a number, it was sufficient to place a standard QT widget into a simple layout and apply QSS styling. However, most components were more complex and involved either extending a standard QT widget or creating a bespoke widget.

For example, a video scene is represented by a bespoke widget (Figure 16). The widget (1) consists of a widget for representing a comment (2), followed by a widget for representing a list of store value commands (3) and then widgets for representing the parameters of the scene (4, 5).


Example Video Scene
Displays Gangnam Style video for 10 seconds

play e.com/watch?v=9bZkp7q19f0
for 10 seconds
from offset 0 seconds
at volume 0
at normal speed

Figure 16: Screenshot of a video scene widget, an example of a bespoke widget.

The comment widget (2) adjusts its vertical size based on the size of the content it stores. This was implementing by using QT's *signal and slot* mechanism to trigger recalculation of the minimum height based on the number of lines and the line height.

The value widgets (4, 5) also adjust their size. They make effective use of horizontal space by adjusting between minimum and maximum widths based on the size of string they store. This was implemented using QT's *size hint* mechanism, where a widget can implement a method that calculates its preferred size. In this case the width is the number of characters weighted by character width and restricted between minimum and maximum widths.

3.5.2 Translating between widget and parse tree representations

To implement drag-and-drop of language components; saving and loading of scripts; and script execution it was necessary to translate between the widget and parse tree representations of a script.

Both representations are tree-like data structures, with deliberately similar shapes to simplify translation.

To aid maintainability, I structured the modules such that the widget module uses services from the parse tree module without the parse tree module having any dependencies on the widget module. Due to this asymmetry, the widget classes implement the WIDGET−TO−PARSE−TREE algorithm whilst a factory implements the PARSE−TREE−TO−WIDGET algorithm.

To implement the WIDGET−TO−PARSE−TREE algorithm, each class in the widget representation is responsible for generating the root of the parse tree associated with it and combining the parse trees generated by its children. This is illustrated in Figure 17.

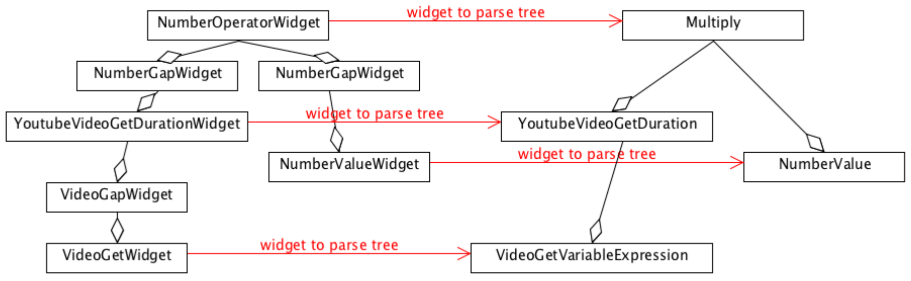


Figure 17: Example of creating parse tree from UI widget classes using example from Figure 12.

To implement the PARSE−TREE−TO−WIDGET algorithm, I applied the *factory method* pattern, a creational design pattern (Gamma et al., 2005). This avoided tightly coupling the parse tree with the editor user interface. In this pattern, a factory is used to create objects without the client specifying the exact class of object to create. When part of the editor needs to translate a parse tree into widgets it calls the widget factory with a parse tree and the translation details are managed by the factory. The widget factory performs the reverse mapping to that illustrated in Figure 17.

3.5.3 Drag-and-drop

In the editor, scripts are manipulated by dragging language components in and out of slots. This form of interaction provides usability benefits summarised under the principles of *Direct Manipulation* by Schneiderman (1983).

I implemented drag-and-drop by extending QT's drag-and-drop capabilities to support communicating parse trees between widgets. I chose to use parse trees rather than widgets as the form of communication, as they capture all important details, including type information, without unnecessary user interface details. QT uses the MIME standard for communicating information for *within-application* and *between-application* drag-and-drop. I used a custom MIME type for parse tree information.

I used the following algorithm for language components that could be copied.

```
1   when a mouse move event is received:

1.1  use the widget-to-parse-tree algorithm (§3.5.2) to generate the
     parse tree representation of the widget which received the mouse
     move event

1.2  notify other widgets that a move has started so they can hint at
     possible move destinations

1.3  serialize the parse tree representation and wrap in a MimeData
     object

1.4  wait on completion of the drag

1.5  notify other widgets that move has completed so they can stop
     hinting at possible move destinations
```

Extract 4: Drag phase in drag-and-drop algorithm.

I used the following algorithm for language components that could receive other language components.

```
1   when a drag enter event is received:
1.1  if widget is not read only:

1.1.1 if MimeData is consistent with a parse tree being dragged, and
      not text, images, or other forms of data:

1.1.1.1if type is consistent:

1.1.1.1.1   allow drag to continue
```

Extract 5: Drop phase of drag-and-drop. Type checking as language component dropped onto slot.

```
1   when a drop event is received:

1.1  deserialize the parse tree representation and use the parse-
     tree-to-widget algorithm to create widget corresponding to the
     language fragment being dragged

1.2  replace or fill in gap with widget
```

Extract 6: Drop phase of drag-and-drop. Deserialization and translation as language component dropped onto slot.

3.5.4 Type enforcement

The drag-and-drop mechanism enforces correct types by preventing a user from dropping a language component onto a slot of an incompatible type. This *type enforcement* mechanism is a form of *forcing function*, an aspect of a design that prevents the user from taking an action without consciously considering information relevant to the action (Norman, 1988). This was important, as users, with little to no prior experience of types in programming languages, have to create type correct programs.

3.5.5 Type hinting

Norman (1988) also emphasises the importance of *affordances*, aspects of a design that suggest how an object should be used. In user testing, I found users experienced difficulty when they attempted an incorrect type combination. Though they recognised there was an issue with their previous action, they were unsure what next action they should take to achieve the goal of their previous action. I improved the design by using *type hinting* to provide affordance. When a user starts to drag a language component, possible destinations of compatible types are highlighted. I implemented this through a periodic walk over the graph of language widgets, toggling the highlight of components of compatible type.

3.6 Capability APIs

I added domain-specific capabilities to PYTHON using the following APIs.

- **Video API:** Responsible for video playback.
- **Text API:** Responsible for full screen display of text and dialogues.
- **YouTube API:** Responsible for access to YouTube data, such as URLs for streaming video and other video metadata.
- **Video Caching API:** Responsible for optimising video playback by caching videos.

The relationships between APIs and their dependencies on extended and existing systems are illustrated below.

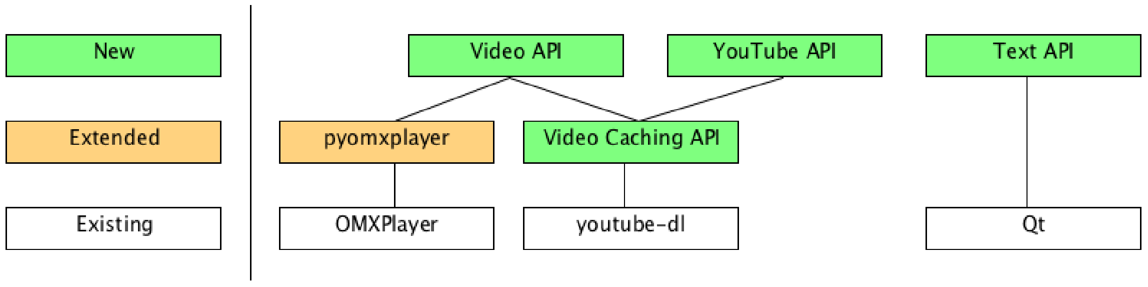


Figure 18: Relationships between APIs and dependencies on extended and existing systems.

3.6.1 Video API

To achieve programmatically controllable video playback I extended PYOMXPLAYER, an existing PYTHON wrapper for OMXPLAYER.

PYOMXPLAYER uses the PEXPECT library to control an OMXPLAYER process. PEXPECT uses a PSEUDO-TTY to communicate with the child application. This allows it to support programs that are designed to be used interactively, which often bypass standard output streams, by sending data directly to their controlling TTY.

I used test driven development to identify and fix bugs in PYOMPLAYER, and then to extend it by adding offset, speed and volume control. For example, I identified the output format used in the latest version of OMXPLAYER had changed and fixed problems this caused.

The performance of the PYOMXPLAYER approach is limited by the reliability of process communication over PEXPECT. I found that an OMXPLAYER process would not respond to commands until several seconds after creation. By studying the source code I confirmed that OMXPLAYER has to initialise a video processing pipeline and set up video streaming before it can process user input. I determined the minimum duration that can be waited before sending commands and exposed this constraint to the wrapping VIDEO API.

In a longer project, modifying OMXPLAYER itself could reduce this delay, for example, by adding command line arguments for offset, speed and volume.

3.6.2 Video Caching API

As an extension, I added video caching to optimise the performance of video playback.

Ideally, when a script is executed video segments should immediately follow each other without delay. I found that due to the time needed to start an OMXPLAYER process and, more significantly, network behaviour there was a 5-10 second delay.

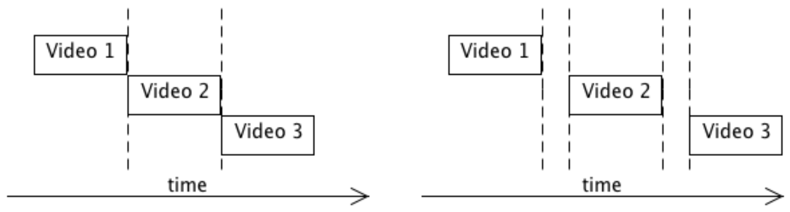


Figure 19: Ideal and actual video switching behaviour.

The general problem is that video playback requires a network throughput high enough to match the bit rate of the video, approximately 1.5 Mbit/s for the Baseline H.264 profile used in this project.

YouTube uses progressive HTTP download over TCP to serve videos (Ghobadi, et al., 2012). Though the YouTube content delivery server implements custom TCP flow control by sending the first 30 to 40 seconds of video as fast as possible, it still has to implement congestion control to prevent congesting the path to the receiver. In particular, in the *slow start* phase of TCP congestion control the throughput grows exponentially from a small initial value (M. Allman, V. Paxson, 2009). Therefore, the nature of congestion control enforces a limit on how quickly network throughput can match the video codec bit rate.

Within the context of use cases for this project, videos exhibit several forms of locality, including:

- **Temporal and spatial locality of frames within a video**: If a frame is played it is likely that the same frame, or a nearby one, will be needed in the near future.
- **Temporal locality of videos:** If a video is played then it is likely the same video will be played again in the near future.
- **Locality of related videos:** If a video is played, it is likely a video related to it by YouTube's suggestion mechanism would be needed in the near future.

Therefore it was appropriate to cache whole YouTube videos, which I implemented by providing an API for accessing video files. I used the following strategy.

```
if a video is already cached:
  return path to video
else:
  begin download in a separate thread
  wait until video ready
  return path to video
```

Extract 7: Core algorithm used in Video Caching API

To avoid greatly increasing the delay when there is a cache miss, the video is downloaded asynchronously and the path provided as soon as enough of the video has been downloaded for OMXPLAYER to be able to fill its first buffer. This delay was important, as OMXPLAYER is not designed to read from local files whilst they are being appended to and so reads a number of bytes from a file without first checking the file size.

To further optimise performance, I restricted the number of possible related videos in the YOUTUBE API. This increased the locality of related videos and increased the cache-hit rate.

I also implemented cache warming. This helped the evaluation go smoothly as I was able to automate warming the cache with videos from the user study script and likely related videos.

3.7 Summary

In this chapter, I described the implementation of the language as a *domain-specific* language using a *piggybacking* approach. I discussed the use of *parse tree* and *widget tree* data structures, and algorithms to translate between them and Python code. I then described how the editor was implemented, including extending Qts drag-and-drop capability to support *type enforcement* and *type hinting*. I followed this by discussing the implementation of domain-specific capabilities; including accessing the video processing capabilities, open source contributions and extending the project through performance optimisations.

Chapter 4 - Evaluation

In this chapter, I describe the user studies I performed, including extending the evaluation with a large child user study (50 person) and a controlled experiment investigating learning outcomes (5 person). I also discuss application of a testing strategy for fault detection and basic performance analysis of the editor.

4.1 Overview

As far as I am aware, there are no existing systems where a direct comparison to this project would be appropriate. The approach this project takes to video composition sits somewhere between a desktop video editor and a text editor being used to manipulate a general-purpose programming language. These approaches are so different in their form of interaction and their aims to this project that it would not be a useful comparison.

With this in mind, I have evaluated the project using the following approaches.

Approach 1. Application of a testing strategy to minimise faults.

I used automated unit, integration and regression testing throughout the project to minimise faults. This included testing: translation between the widget and parse tree representations and PYTHON code; operations on the parse tree such as finding user defined variables; handling of user input such as the various types of allowed YouTube video identification strings; and behaviour of the video player and caching APIs.

Approach 2. Structured user studies.

I ran a large user study with children across two public science events, which demonstrated achievement of the main success criterion (§B.4). These studies also explored usability problems in the language and editor.

I improved on the child user study with a smaller controlled experiment, where greater coverage of the language features and more structure was possible. This allowed usability problems to be explored more systematically and the attainment of learning outcomes to be explored.

Approach 3. Performance analysis of the editor.

Although performance was not an explicit requirement, I extended the evaluation by measuring the current performance of key processes within the editor.

4.2 Child user study

In the child user study, I investigated the following:

- Achievement of the main success criterion.
- Usability problems in the programming language and editor.

4.2.1 Design

The study involved participants aged between 10 and 18 years old with little to no prior programming experience and took place at two public science events, during the Cambridge Science Festival.

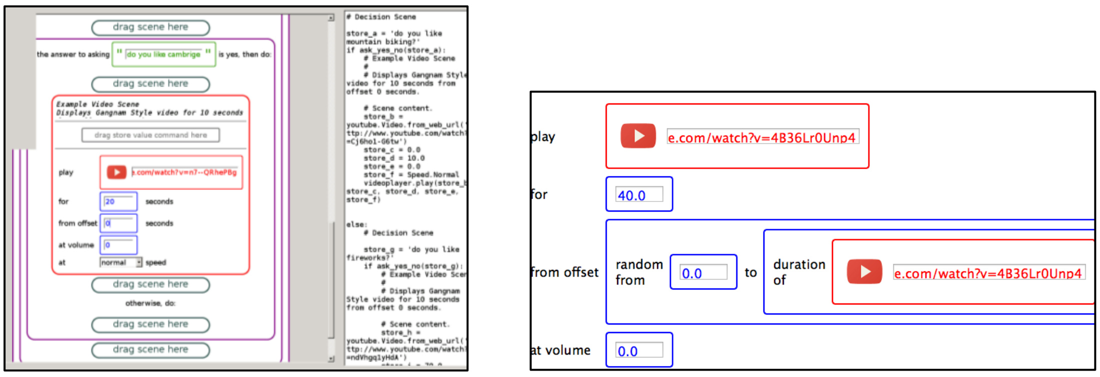
I demonstrated the system to participants in small groups, with a stand at the events, and invited them to experiment with the system themselves. As they interacted with the system I recorded their speech and behaviour. Each session involved a range of levels of guidance, from purely demonstrating examples to inviting participants to interact with the system autonomously.

People often came in groups and moved freely between stands, so it is not possible to accurately state the number of participants, as in a controlled experiment. The estimated number of participants is 50.

4.2.2 Results and discussion

4.2.2.1 Main success criterion

The scripts created by participants demonstrate achievement of the *main success criterion*; successful creation of sample programs in the visual language, which demonstrate video processing behaviours from a test suite of end user scenarios (§B.4). In this section I will discuss a couple of examples.



```
# Decision Scene

store_a = 'do you like
mountain biking?'
if ask_yes_no(store_a):
    # Example Video Scene
    #
    # Displays Gangnam Styl
video for 10 seconds from
offset 0 seconds.

    # Scene content.
    store_b =
youtube.Video.from_web_url(
ttp://www.youtube.com/watch
=Cj6hol-G6tw')
    store_c = 0.0
    store_d = 10.0
    store_e = 0.0
    store_f = Speed.Normal
    videoplayer.play(store_
store_c, store_d, store_e,
store_f)


else:
    # Decision Scene

    store_g = 'do you like
fireworks?'
    if ask_yes_no(store_g):
        # Example Video Sce
        #
        # Displays Gangnam
Style video for 10 seconds
from offset 0 seconds.

        # Scene content.
        store_h =
youtube.Video.from_web_url(
ttp://www.youtube.com/watch
=ndVhgq1yHdA')
```

Figure 20: Samples from scripts created by participants in the child user study.

The first screenshot is from a script created by two participants with very different video interests. They used selection to create a script that would play one of two possible video segments depending on user input.

The second screenshot is from a script created by a participant interested in information about videos. They used mathematical operations on video metadata to play a sequence of video segments from random positions into the videos.

4.2.2.2 Usability problems

To analyse the recordings of a participant's speech and observations on their behaviour I used an *open coding* approach. I started by identifying specific problems that individual participants had experienced, and then grouped similar problems together to identify general usability problems. I then used the frequency of occurrence of a problem as a measure of its relative severity.

The five most common problems are described below.

	Usability problem	**Frequency**
C1	Participants expressed confusion over the meaning of the icon used to represent a video value. In particular participants expected clicking on it would run their a script.	10
C2	Participants expected to be able to edit values by clicking and the typing into empty slots for those values.	7
C3	Participants used values where a scene involving content of the same type was needed and vice versa. In particular participants attempted to create a new video scene by dragging a video value into the scene edit component.	7
C4	Participants expressed frustration with a recurring problem in the system causing unscheduled termination approximately every 30 minutes of use.	4
C5	Participants attempted to drag language components by elements inside the area of the component that did not initiate a drag.	4

Table 2: Five most common usability problems in the child user studies.

I will review the problems identified here when I discuss the results of the controlled experiment.

4.2.3 Conclusions

The child user study demonstrated achievement of the main success criterion and identified usability problems. The study was restricted by the constraints of the events where it took place and so I went on to improve on the study by running a smaller controlled experiment.

4.3 Controlled Experiment

In the controlled experiment, I investigated the following:

- Usability problems in the programming language and editor.
- Attainment of learning outcomes.

4.3.1 Design

For practical reasons, it was necessary to carry out this study with older students, rather than attempting to recruit young children for experimental sessions. The experiment involved five participants, with little to no prior programming experience.

To investigate the attainment of learning outcomes a pretest-posttest design was used, where a participant's understanding of computational concepts was measured before and after interacting with the system. As participants interacted with the system, the *think-aloud protocol* was used to investigate usability problems. In the think-aloud protocol (Lewis, 1982), participants carry out tasks whilst talking about what they are looking at, thinking, doing and feeling. The think-aloud protocol was selected as it is cheap to use, hence feasible for this project, and allows for a large amount of qualitative data to be collected from a few participants.

The experimental design is summarised in Figure 21.

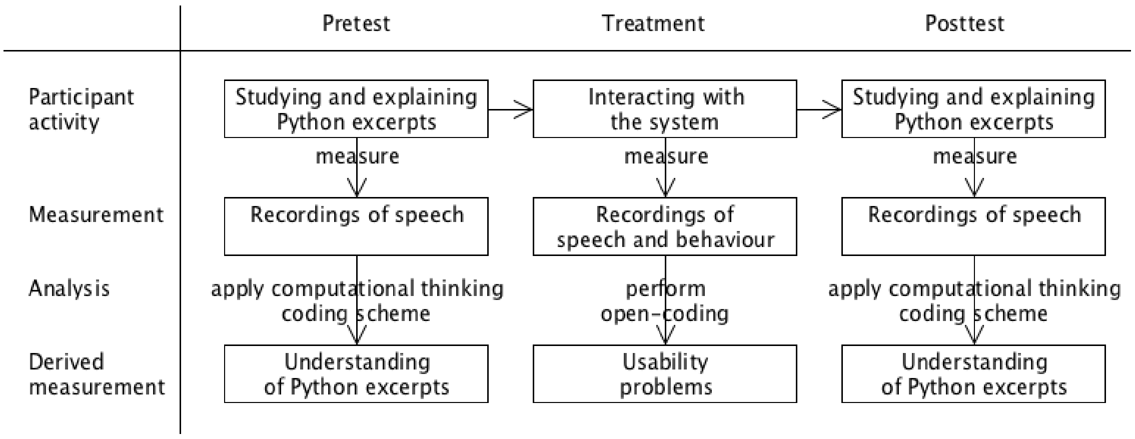

Pretest
Treatment
Posttest
Participant activity
Studying and explaining Python excerpts
Interacting with the system
Studying and explaining Python excerpts
measure
measure
measure
Measurement
Recordings of speech
Recordings of speech and behaviour
Recordings of speech
Analysis
apply computational thinking coding scheme
perform open-coding
apply computational thinking coding scheme
Derived measurement
Understanding of Python excerpts
Usability problems
Understanding of Python excerpts

Figure 21: Summary of controlled experiment design.

I ran the experiments in person and to reduce variation in confounding variables followed a script. The full user study plan, including scripts, is included in §A.6.

In the following sections I will explain the experimental design in greater detail.

4.3.1.1 Usability problems

Usability problems in the language and editor were investigated through a think-aloud design.

Participants performed three different types of activity distributed across 11 examples of increasing complexity. The activities were: researcher led demonstrations, primarily used to introduce concepts; simple modifications of incomplete examples; and complex modifications, involving changing structure within examples.

In a larger project, it would be preferable to give the participants more time to interact with the system to cover more forms of interaction. In particular, there was limited scope for exploratory design. However, the section involves a lot a novel concepts and it would have been unrealistic to expect the participants to master the concepts in the time available.

4.3.1.2 Learning outcomes

Attainment of learning outcomes was investigated using a pretest-posttest design.

Participants were shown excerpts from PYTHON programs, before and after interacting with the system. They were asked to attempt to explain the excerpts and a recording was made of their responses. To measure their understanding of the scripts, a coding scheme was used to map their vocal responses onto computational thinking concepts. The number of concepts they articulated was then used to measure understanding.

The coding scheme was based on formulations of computational thinking by Wing (2006), the Internal Society for Technology in Education (2011) and the Computing at School Working Group (2012). The scheme is explained in greater detail in §A.6.

A large list was used to increase the chance of catching notable expressions and it was anticipated that participants would identify a minority of concepts. Though the concepts were phrased in a technical way, the responses were assessed on whether there was a reasonable correspondence between utterances and concepts. In a project with greater available resources, it would be preferable for the coding to be performed by an external blind rater to reduce bias from direct contact with the project and participants.

The coding scheme used a simple weighting, each concept carries the same weight. In a more sophisticated study, different weights could be used to take into account the complexity of different concepts.

4.3.2 Results and discussion

4.3.2.1 Usability problems

I applied the same open-coding approach from §4.2.2.2 to the recordings of speech and behaviour of participants as they interacted with the system. I first identified specific problems and then grouped similar problems together. The five most common problems are described below.

	Usability problem	**Frequency**
A1	Participants expressed confusion between the video and video collection types.	9
A2 (C3)	Participants used values where a scene involving content of the same type was needed and vice versa. In particular participants attempted to create a new video scene by dragging a video value into the scene edit component.	6
A3 (C5)	Participants attempted to drag language components by elements inside the area of the component that did not initiate a drag.	5
A4 (C1)	Participants expressed confusion over the meaning of the icon used to represent a video value. In particular participants expected clicking on it would run their a script.	4
A5	Participants expressed confusion with the concept of selecting a random video from a collection of videos related to a video.	4

The results suggest which usability problems are most severe. As expected, there is overlap with the problems identified in the child user study (A2/C3, A3/C5 and A4/C1) however there also problems that rank as much more significant in this study (A1 and A5). The controlled experiment covered the language more thoroughly and gave participants more opportunity to interact with the system, which helps explain why A1 and A5 are measured as more severe.

I will discuss the most severe usability problems in greater detail by considering causes and improvements.

4.3.2.2 Problem A1

In the language, the colour red is associated with the *video type* and pink with *video collection type*. An *icon* from the YouTube domain is used to represent the correspondence to YouTube videos and *multiplicity* of this icon is used to represent a collection of videos. Problem A1 suggests these differences do not sufficiently differentiate the video and video collection types.

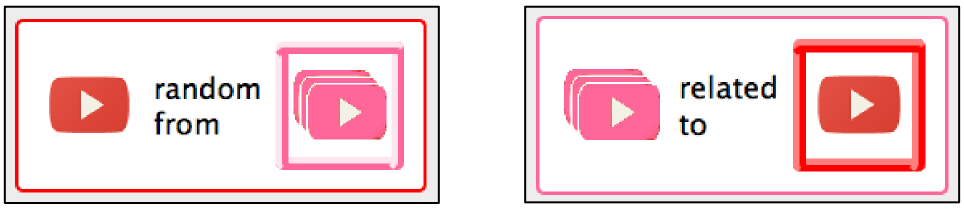


Figure 22: Screenshots demonstrating the use of colour and icon design to represent the video and video collection types in the language.

The representation might be improved by using alternative representations of multiplicity in the video collection icon, for example by placing the icons next to each other rather than stacking them. Improvement of the design could be carried out systematically by proposing several alternatives and receiving feedback from representative users of their suitability.

4.3.2.3 Problem A2

In the language, *video and text values* are parameters to *video and text scenes*, along with duration and other scene specific parameters. Colour is used to associate values with scenes that use them as content. Different shapes are used to differentiate between values and scenes. Problem A2 suggests values and scenes are represented too similarly.

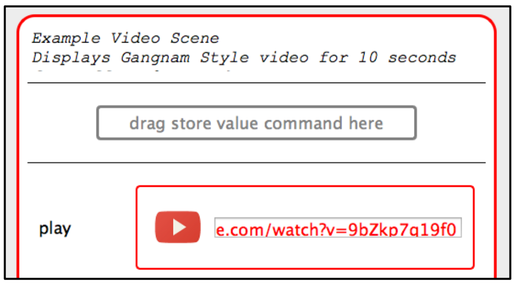

Example Video Scene
Displays Gangnam Style video for 10 seconds
drag store value command here
play
e.com/watch?v=9bZkp7q19f0

Figure 23: Screenshot demonstrating the use of colour to associate a video value with a video scene, for which it provides the content.

The most common problem was that participants tried to add a video by dragging a video value, rather than video scene, into the script manipulation area (highlighted in Figure 15). Less common was that participants would try to fill in a gap requiring a value with the scene that uses that value.

The language might be improved by making the visual representation of values and scenes more different, for example by using metaphors from the domain of film scripts such as a paper like textured background. As the problem has a general form, more informative feedback could be used, such as explaining values and scenes to the user when they make such an error. Finally, an approach used by SCRATCH could be adopted, where the editor performs a similar action instead, to maintain progress and motivation, even if it is not quite what the user intended.

4.3.2.4 Problem A3

Language components are implemented by combining many QT widgets together, including standard QT widgets for text entry and for choosing between a set of options (§3.5.1). Due to prioritisation of features, for some widgets drag-and-drop is not implemented for these standard input widgets, just for the area outside. Problem A3 shows that this is serious usability issue, the core problem being that from the users point of view, a language component is a whole unit and there would be no reason why only part would form an active region.

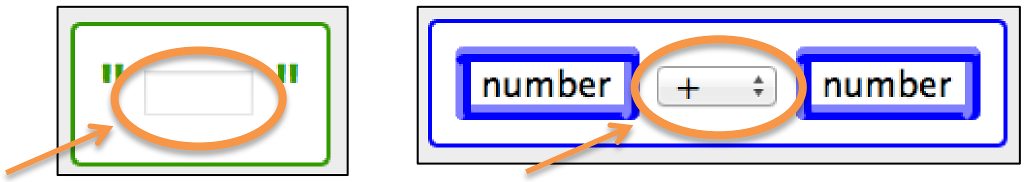


Figure 24: Highlighted "cold spots" where standard Qt widgets are used in draggable language components.

By adding behaviour to input widgets derived from the standard widgets it would be possible to support the drag-and-drop scheme used in the editor.

4.3.2.5 Learning outcomes

The pretest and posttest measurements are summarised below.

Participant	Pretest understanding / number of identified concepts	Posttest understanding / number of identified concepts
A	4	8
B	4	18
C	4	7
D	3	10
E	2	2

Table 3: Summary of pretest and posttest understanding measurements.

To test the statistical significance of these results I used the *Wilcoxon Matched Pairs* test with the alternative hypothesis that understanding of the scripts, measured by the number of identified concepts, would be greater after users interacted with the system. I selected a non-parametric test as there is insufficient data to support an underlying normal distribution and the Wilcoxon Matched Pairs test in particular as the variation is within-subject.

The test gave a suggestion that interaction with the system may contribute towards attainment of the learning outcomes in the experiment (p-value 0.07) and in general that using the system would contribute towards learning computational thinking concepts and processes.

4.3.3 Conclusions

I have identified usability problems in the language and editor, with their relative importance, and suggested improvements for the most severe problems.

There is reasonable (but not significant at the 5% level) evidence that interaction with the system may contribute towards learning computational thinking concepts and processes. I have not been able to consider threats to internal and external validity in detail, in particular the sample size and methodology. A more rigorous evaluation of this system would compare the learning outcomes to those in a controlled condition, with a larger sample size. For example, by using an existing educational product such as SCRATCH. Despite these caveats, the result is encouraging and will be discussed further in §5.

4.4 Performance analysis

4.4.1 Introduction

I extended the evaluation by carrying out basic performance analysis of the editor. To approximate the overall performance I used the time taken to load an example script. Specifically, the time between clicking on an interface element to initiate the load and the script being rendered and modifiable. I selected this measure as it involves all the core processes within the editor, specifically handling user input, translating between the parse tree and widget representations, translating from parse tree to PYTHON code and rendering the widget representation.

4.4.2 Design

I used the 11 example scripts from the controlled experiment (§4.3.1.1). These were a good representation of the scripts a user would create as they span the range of typical sizes and involve all language components.

For each script, I took three measurements to control for processor scheduling and memory management. I then took the mean average of each set of measurements and calculated the corrected sample standard deviation to estimate the error in the mean.

4.4.3 Results and discussion

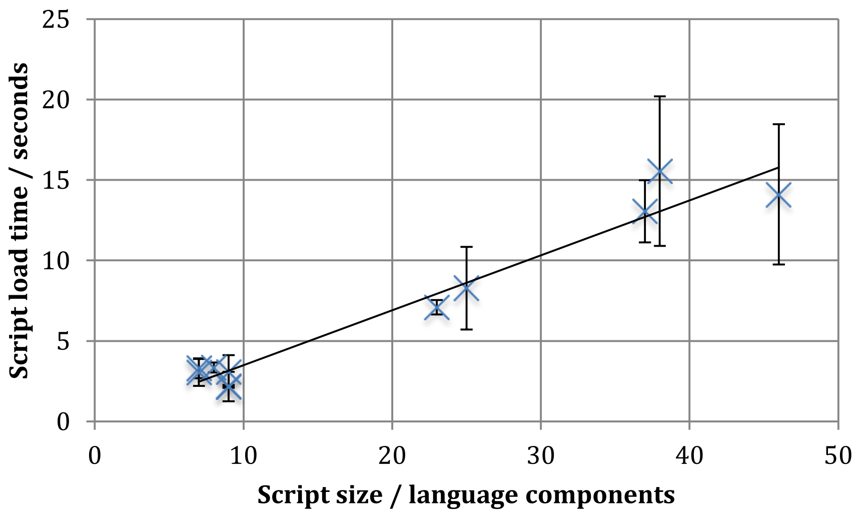


Figure 25: Script load performance of the editor.

The results suggest a linear relationship between the time taken to load a script and the size of the script. This is consistent with the complexity of the PARSE-

TREE-TO-WIDGET algorithm, which carries out a similar amount of work for each component in the parse tree and so is linear in the number of components.

The results show that the load times are large relative to user needs. Three limits for response times can be distinguished (Miller, 1968; Card et al., 1991).

- Up to 0.1 seconds, the user feels the system is reacting instantaneously.
- Up to 1 second, the users flow of thought will not be interrupted but they will notice the delay.
- Up to 10 seconds, the users flow of thought is interrupted however their attention is maintained.

The results suggest that for small to medium scripts the user's attention will be maintained however for larger scripts their flow of thought may be interrupted and attention lost. Myers (1985) recommends, where an immediate response cannot be given, continuous feedback should be provided to the user. Therefore, the system could be improved by providing feedback that a script is being loaded.

Furthermore, the system could be improved by reducing the load time through more efficient algorithms. In the current implementation, every time a new component is added a script change event is propagated up to the widget responsible for the script manipulation area and the script is translated into PYTHON code. In normal usage this behaviour is appropriate however when a script is loaded many components are added at once and so many unnecessary translations are being performed. Hence, the performance could be improved by disabling generation of script change events whilst a script is being formed.

In conclusion, basic performance analysis of the editor has shown its behaviour needs to be optimised to meet the response needs of users.

4.5 Summary

In this chapter, I described the user studies I performed, including extending the evaluation with a large child user study (50 person) and a controlled experiment investigating learning outcomes (5 person). I also discussed application of a testing strategy for fault detection and basic performance analysis of the editor.

Chapter 5 - Conclusion

5.1 Comparison with original goals

In the proposal, I specified the main success criterion of successful creation of sample programs in the visual language, which demonstrate video processing behaviours from a test suite of end user scenarios. Through 15 hours of user studies, involving over 50 participants, I have satisfied this criterion.

I refined the proposal by specifying two main objectives (§1.3). Objective 1 has been demonstrated through the user studies and Objective 2, providing an example to other developers in accessing the video processing capabilities, has been achieved through development of the Video API (§3.6.1) and open source contributions to the existing YT and PYOMXPLAYER projects (§2.3.2 and §3.6.1).

I completed all core work items (§2.1.1) and extended the project by supporting an external content source, YouTube; by providing exploration of basic data structures through information structures in the YouTube domain; and by running a large child user study (50 person).

I have evaluated the project through users studies and performance analysis. I identified the relative importance of usability problems in the system and through statistical significance testing of the controlled experiment results obtained a promising suggestion (p-level 0.07) that interaction with the system may contribute towards attainment of learning outcomes (§4.3.2).

5.2 Future work

This project takes an alternative approach to the current market leaders by providing a domain-specific visual language representation of a user's program next to the equivalent PYTHON code. In contrast, in products such as SCRATCH, users only interact with a domain-specific representation and have no way to see their program in a general-purpose language. This alternative multi-representation approach offers promising educational advantages, which could be explored through comparing the attainment of learning outcomes with and without the PYTHON code representation.

Investigating usability problems in the language and editor (§4.3.2.1) has identified where the most significant problems are and therefore where the usability of the system might be most beneficially improved. Furthermore, basic performance analysis of the editor has shown its behaviour needs to be optimised to meet the response needs of users.

The language and editor have been designed to support more advanced video manipulation capabilities as they become available. Since the early technological investigations (§2.3.3) there have been promising developments, which may be able to provide hardware decode of video alongside hardware accelerated video manipulation.

5.3 The Future

The technological expectations of children are radically changing. Such is the pace of technological change that technologies that were ground breaking for one generation become ubiquitous and taken for granted by the next. To create compelling educational products, children need to be given the ability to create programs of a comparable standard to the consumer devices they use.

In many ways the Raspberry Pi is the spiritual successor of the BBC Micro, a platform that inspired a generation to take up Computer Science. Indeed the Raspberry Pi foundation originally sought to brand their device as its direct successor. However, whilst the BBC Micro offered compelling general-purpose performance, for example by comparison to videogame consoles of the time, the Raspberry Pi's general-purpose performance is modest and so it is important to focus on its main hardware strength, video processing.

In this project I have created the first video processing educational application for the Raspberry Pi and broken new ground for other people to create more powerful, yet inexpensive and compelling, educational applications for the Raspberry Pi.

Bibliography

Blackwell, A. (2011). Visual Representation. *The Encyclopedia of Human-Computer Interaction, 2nd Ed.* Retrieved from http://www.interaction-design.org/encyclopedia/visual_representation.html

Broadcom. (2013). BCM2835 Media Processor. Retrieved April 12, 2013, from http://www.broadcom.com/products/BCM2835

Buxton, B. (2010). *Sketching User Experiences: Getting the Design Right and the Right Design: Getting the Design Right and the Right Design* (p. 448). Morgan Kaufmann.

Card, S. K., Robertson, G. G., & Mackinlay, J. D. (1991). The information visualizer, an information workspace. *Proceedings of the SIGCHI conference on Human factors in computing systems Reaching through technology - CHI '91* (pp. 181–186). New York, New York, USA: ACM Press. doi:10.1145/108844.108874

CAS. (2012). *Computer Science: A curriculum for schools*. Retrieved from http://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf

comScore. (2010). comScore Releases May 2010 U.S. Online Video Rankings - comScore, Inc. Retrieved May 10, 2013, from http://www.comscore.com/Insights/Press_Releases/2010/6/comScore_Releases_May_2010_U.S._Online_Video_Rankings

Conway, M. (1997). *Alice: Easy-to-Learn 3D Scripting for Novices*. University of Virginia.

Conway, M., Audia, S., & Burnette, T. (2000). Alice: lessons learned from building a 3D system for novices. *... in computing systems*, 1–8. Retrieved from http://dl.acm.org/citation.cfm?id=332481

DfE. (2013). Consultation on (i) the Order for replacing ICT with computing and (ii) the regulations for disapplying aspects of the existing National Curriculum. Retrieved May 10, 2013, from https://www.education.gov.uk/consultations/index.cfm?action=consultationDetails&consultationId=1902&external=no&menu=1

Erickson, T. D. (1990). Working with interface metaphors. *The art of human-computer interface design* (pp. 65–73). Addison-Wesley.

Ghobadi, M., Cheng, Y., Jain, A., & Mathis, M. (2012). Trickle: Rate limiting YouTube video streaming. *Proceedings of the USENIX ...*.

Green, T. (1989). Cognitive dimensions of notations. *People and Computers V*.

ISTE. (2011). CT Vocabulary and Progression Chart. Retrieved from http://www.iste.org/docs/ct-documents/ct-teacher-resources_2ed-pdf.pdf

Johnson, J., & Henderson, A. (2002). Conceptual models: begin by designing what to design. *interactions*.

Ko, A. J., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., Wiedenbeck, S., Abraham, R., et al. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys*, *43*(3), 1–44. doi:10.1145/1922649.1922658

Lewis, C. H. (1982). Using the "Thinking Aloud" Method In Cognitive Interface Design.

M. Allman, V. Paxson, and E. B. (2009). RFC 5681 - TCP Congestion Control.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. doi:10.1145/1868358.1868363

Mendelsohn, P., Green, T. R. G., & Brna, P. (1990). Programming Languages in Education: The Search for an Easy Start. *Psychology of Programming*. Academic Press Limited.

Mernik, M., Heering, J. A. N., & Sloane, A. M. (2005). When and How to Develop Domain-Specific Languages AND. *ACM Computing Surveys*, *37*(4), 316–344. doi:10.1145/1118890.1118892

Miller, R. B. (1968). Response time in man-computer conversational transactions. *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, 267. doi:10.1145/1476589.1476628

Myers, B. A. (1985). The importance of percent-done progress indicators for computer-human interfaces. *ACM SIGCHI Bulletin*, (April), 11–17. Retrieved from http://dl.acm.org/citation.cfm?id=317459

Norman, D. (1988). *The Psychology of Everyday Things*.

Rode, J., Stringer, M., & Toye, E. (2003). Curriculum-focused design. *Proceedings ACM Interaction Design and Children*, 119–126. Retrieved from http://dl.acm.org/citation.cfm?id=953553

Rogers, Y., Sharp, H., & Preece, J. (2011). *Interaction Design*. John Wiley & Sons Ltd.

Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *ACM SIGSOC Bulletin*, *08*.

Spinellis, D. (2001). Notable design patterns for domain-specific languages. *The Journal of Systems and Software*, *56*, 91–99.

Upton, L. (Raspberry P. F. (2013). FAQs | Raspberry Pi. Retrieved April 12, 2013, from http://www.raspberrypi.org/faqs

Wile, D. (2003). Lessons learned from real DSL experiments. *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the* (p. 10 pp.). IEEE. doi:10.1109/HICSS.2003.1174893

Wilson, A., & Moffat, D. (2010). Evaluating Scratch to introduce younger schoolchildren to programming. *… of the Psychology of Programming …*. Retrieved from http://scratched.media.mit.edu/sites/default/files/wilson-moffat-ppig2010-final.pdf

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33. doi:10.1145/1118178.1118215

Appendix A - User Study Plan

A.1. Objectives

1. Investigate usability problems in the programming language and editor.
2. Investigate the attainment of learning outcomes gained through use of the programming language and editor.

A.2. High level methodology

Usability problems in the programming language and editor will be evaluated using a think aloud study.

Attainment of learning outcomes will be evaluated through a before/after study.

A.3. Participants

Persons with little to no prior experience of programming.

Controlled experiment – Adults, Child user study – Child aged between 10 – 18.

Experience will be measured by number of hours of programming experience.

Little to no prior experience will be defined by less than one hour of programming experience.

A.4. Precautions

Risk	**Precaution**
Inappropriate YouTube content	When accessing the YouTube API using options to exclude "racy" content. Monitoring users as they use the system and stopping the experiment if inappropriate content is shown.
Placing participants under stress	Making it explicit in the consent form

when surveying participants on their understanding of programing	that participants are not being assessed. Asking questions in a non-confrontational way.

A.5. Key to formatting used

Script

RESEARCH ACTION

PARTICIPANT ACTION

A.6. Methodology

Structure

The study will involve one participant at a time and have the following structure:

1. Introduction. The research will explain the objectives of the study and the structure.
2. Before section of before/after study.
3. Think aloud study.
4. After section of before/after study.
5. Debrief.

Introduction

The introduction will follow a script to reduce variation in participant's preparation before experiments and to make sure important ethical points are emphasised.

Thank you for interest in the study.

In my dissertation we have been developing a educational software package for the Raspberry Pi. The Raspberry Pi is a cheap credit card sized computer developed with the intention to provide a platform for programming experimentation and teaching of basic computer science in schools. The purpose of this project has been to create an environment for people without prior experience of programming to explore programming by manipulating YouTube content.

Before I go any further I'll mention a couple of important points form the consent form. An audio recording will be made for the exclusive use by this study and will be deleted after completion. You can withdraw from the experience and request deletion of all data related to your involvement at any time.

The objectives of the study are to:

1. *Investigate usability problems*
2. *Investigate attainment of learning outcomes.*

We're not evaluating your ability but the software we have developed.

The study will start with the before section of the before/after study, then the think aloud study and then the after section of the before/after study.

Before/after study

Design

The participant will be shown excerpts from PYTHON programs before and after the study. They will be asked to explain the excerpts and a recording will be made of their responses. A coding scheme will be used to map their vocal responses onto a representation of their understanding of the excerpts.

The study will use a simple example and a more complicated example. The examples will be correct PYTHON code. The code will include the idiosyncrasies present in the code generated by the graphical editor, in particular the use of temporary variables. These decisions have been made over using simplified PYTHON code as it will test whether interacting with the system helps a participant see through unnecessary details which is representative of real software development.

It is important that care is taken to avoid the participants being placed under stress. Participants will be asked to explain the excerpts with a strong emphasis that they are not expected to be familiar with the concepts and that the test is not of their ability but whether or not using the system helps them understand the excerpts. Furthermore I have received advice from other researches that using hard copy rather than presenting the code on the screen will help reduce stress on the participants and encourage the participants to be more open to speculating about the excerpts.

Procedure

Short example

```
# Example video scene using rope swing video. Plays video from an offset
# of 10 seconds into the video for 15 seconds.

# Scene content.

store_a                                                              =
youtube.Video.from_web_url('http://www.youtube.com/watch?v=4B36Lr0Unp4')
store_b = 10.0
store_c = 15.0
store_d = 2.0
store_e = Speed.Normal

videoplayer.play(store_a, store_b, store_c, store_d, store_e)
```

Long example

```
# Example sequence of video scenes using rope swing video and a video
# related to it. Plays rope swing video from an offset of 10 seconds
# into the video and for 15 seconds. Plays the related video from 1
minute
# into the video.

# Store value commands.

curr_video                                                     =
youtube.Video.from_web_url('http://www.youtube.com/watch?v=4B36Lr0Unp
4')

# Scene content.

store_a = curr_video
store_b = 10.0
store_c = 15.0
store_d = 2.0
store_e = Speed.Fast

videoplayer.play(store_a, store_b, store_c, store_d, store_e)

# Store value commands.

store_f = curr_video
store_g = store_f.related()
curr_video = store_g.random()

# Scene content.

store_h = curr_video
```

```
store_i = 60.0
store_j = 5.0
store_k = -2.0
store_l = Speed.Slow

videoplayer.play(store_h, store_i, store_j, store_k, store_l)
```

Coding

The responses will be coded against a list of programming concepts. The more programming concepts a participant accurately identifies the greater their understanding of the scripts. The programming concepts are phrased in a technical way however the participants responses will not be assessed on technical phrasing but whether there is a reasonable correspondence between their utterance and the concept.

In a larger project with more available resources it would be preferable to have an external blind rater who was not involved in running the experiment and so would not be biased by direct contact with the participant.

The concepts are organised into categories. A very simple coding scheme will be used where each entry carries equal weight. The participants understanding of the excerpts will be quantised by counting the number of concept entries where there is a reasonable correspondence between their utterances and the technical description.

In a more sophisticated study it may be preferable to assign different weights to different entries to take into account the importance and complexity of different concepts.

The concepts are based on the Computing at School curriculum. The list must be specified ahead of the experiment and so is large to increase the chance of catching notable expression of concepts. It is anticipated that candidates will identify a small minority of the concepts.

Key concepts:

1. Languages, machines and computation
   1.1. Identifies that the scripts describe how the computer should behave
2. Data and representation
   2.1. Identifies that there are different types of data
   2.2. Identifies that numbers are used for duration, offset and volume.

   2.3. Identifies that `Speed.Normal` corresponds to a speed of playback.
   2.4. Identifies that the video played relates to a line involving `youtube.Video.from_web_url('http://www.youtube.com/watch?v=4B36 Lr0Unp4')`
3. Communication and coordination
   3.1. Identifies that there must be communication between the computer running the script and YouTube
   3.2. Identifies that the machines would use the Internet to communicate
4. Abstraction and design
5. Computers and computing are part of a wider context
   5.1. Identifies that software can be used to make computers do tasks that they are better at that humans
   5.2. Identifies that the scripts could creative applications
   5.3. Identifies that computational thinking and programming have wider relevance than professional software development and academics study into Computer Science.

Key processes:

1. Abstraction, modelling and generalising
   1.1. Identifies that each scene a scene has two sections: a storage section and a main content section.
2. Designing and writing programs
   2.1. Sequencing – doing one step after another
      2.1.1. Identifies a line will be interpreted before lines following it down the page.
      2.1.2. Identifies that one line is interpreter at a time. Lines are not executed in parallel.
   2.2. Language concepts that support abstraction
      2.2.1. Identifies that the behaviour of `videoplayer.play` is to cause a video be played.
      2.2.2. Identifies that the behaviour of `videoplayer.play` is not static.
      2.2.3. Identifies that the behaviour of `videoplayer.play` depends on values specifying duration, offset, video and speed.
      2.2.4. Identifies that there is a relationship between the variable names in a line with a function call and previous assignment statements.
      2.2.5. Identifies that the nature of that information is to pass information around in the script.
      2.2.6. Identifies that the order of parameters is significant.
   2.3. Interaction with the program's environment

        2.3.1. Identifies that `http://www.youtube.com/watch?v=4B36Lr0Unp4` identifies a the webpage for a video.
        2.3.2. Identifies that `http://www.youtube.com/watch?v=4B36Lr0Unp4` is being used to identify the video in an abstract sense.
        2.3.3. Identifies that `store_g = store_f.related()` involves the concept of one YouTube video being related to another YouTube video.
        2.3.4. Identifies that `curr_video = store_g.random()` involves a change in multiplicity (from several videos to one video)

3. Debugging, testing and reasoning about programs
    3.1. Identifies that lines starting with `#` describe the behaviour of the scripts.
    3.2. Identifies that lines starting with `#` are secondary notation, that is they are not part of the notation expressing behaviour but help a reader understand what's going on.
    3.3. Identifies that the assignment syntax `a = b` causes a value to be stored in a location.
    3.4. Identifies that in the assignment syntax `a = b`, `a` is used to refer to a location.
    3.5. Identifies that the dereference syntax `a` allows a value stored in a location identified by name `a` to be used.
    3.6. Identifies that ` store_f = curr_video` involves the video identified by `http://www.youtube.com/watch?v=4B36Lr0Unp4` earlier in the script.

Think aloud study

Design

This study will introduce language features to the participant by increasingly complex examples.

In each example the researcher will start with describing a user need and then involve a language feature. For example by describing how a user may want to emphasise part of a video and then by involving a Repeat Scene and a Video Scene to repeat part of a video. This approach will be taken to engage the participant.

The examples have been chosen on the following principles:

- Prefer smaller increases in complexity to keep the participant motivated and interested.

- Prefer strongly motivational examples. In particular videos have been preferred that are popular or have an engaging quality. Compilations of notable videos and video view counts have been used.
- Prefer to introduce familiar concepts first, such as search before related.
- Prefer examples where the motivation to use language features is strong rather than contrived.

The aim of this study is to understand usability problems so we want to give the participants as much opportunity as possible to interact with the system and vocalise what they're thinking. As the interface and concepts involves is novel it will be necessary to spend some time introducing language features and concepts. There is a balance to be made between explaining parts to allow the participant to attempt more involved interactions and not explaining parts to avoid skipping over usability problems. This balance will be achieved by using three types of example:

A. Completely researcher lead where the research builds a complete example from scratch.
B. Simple modification to an incomplete example built by the researcher.
C. Complex modification to an incomplete example built by the researcher.

In a larger project it would be preferable to give the participants more time to interact with the system in a way driven by the participant themselves rather than the structured format used. This would be useful as it would cover more forms of interaction and do so more thoroughly. For example there is limited scope for exploratory design in this design. However, the study introduces a lot of novel concepts as well as a novel interface and so it would be unrealistic to expect the participants to be able to pick up enough in the time available to make this work.

Measurements

- Audio recording.
- Screenshots taken for each interactive part in the structured section and throughout the unstructured section.
- The number of stages through the study the participant chooses to do. This would be particularly relevant for a future study with children.

Procedure

Introduction

I'm going to take you through some examples of what the environment can do. I'll start with simple examples and move onto more complex ones. To begin with I'll construct complete examples and as we go through I will invite you to have a go at completing examples.

This is a think aloud study which means we're interesting in what you're thinking when you interact with the system. Now, I'd like to emphasise that we're not assessing you, what we're looking for is usability problems in the system. So for example if you see the handle on a door and you think that it's shaped in a way that suggests you should pull it but it's actually a push door then that suggests there is a usability problem in the way the designer of the door choose to design the handle.

It can feel quite unnatural to talk through what you're thinking and so I'll encourage you throughout to keep talking.

I will explain some parts of the environment however will deliberately not explain all parts. When you come across something you don't understand or perhaps is confusing then it's really useful if you tell me that. You are welcome to ask questions at any time however I may choose to answer them later or perhaps may choose not to answer them to keep the test fair across participants.

(A) Example 1 – Video

Motivation: I like to watch videos on YouTube.

BUILD EXAMPLE 1. PLAY.

(B) Example 2 - Volume

Motivation: I like to listen to my music loud so I'm going to increase the volume.

BUILD EXAMPLE 2. PLAY.

(B) HOW ABOUT YOU HAVE A GO AT DECREASING THE VOLUME.

Changing an atomic value. Using play.

(B) Example 3 - Speed

Motivation: I really like "A DRAMATIC SURPRISE ON A QUIET SQUARE" however it takes a while to get to the best bit so I'm going to play it fast.

BUILD EXAMPLE 3. PLAY.

(B) HOW ABOUT YOU HAVE A GO AT PLAYING THE SAN DIEGO FIREWORKS 2012 VIDEO SLOWLY.

Inserting an atomic video value from the palette into an existing scene. Changing speed.

(A) Example 4 - Offset

Motivation: I'd to skip to the best bit of Felix Baugartner's Supersonic Freefall video however I don't want to play it fast and miss the good bits. I'm going to play it from an offset of 60 seconds.

BUILD EXAMPLE 4. PLAY.

(A) Example 5 – Video collections and search

Motivation: The environment allows you to interact with YouTube. I'm going to be at the Cambridge Science Festival in a few weeks so I would like to watch some videos to prepare.

BUILD EXAMPLE 5. PLAY.

(B,C) Example 6 – Related videos

Motivation: I'm a big fan of Kid President and his Pep Talk. I'd like to see what else he's been up to. In this scene I'm selecting a random video from a collection of videos which are related to Kid President's Pep Talk Video.

BUILD EXAMPLE 6. PLAY.

(B) HOW ABOUT YOU HAVE A GO AT PLAYING A VIDEO RELATED TO CAMBRIDGE HARLEM SHAKE.

(C) HOW ABOUT YOU HAVE A GO AT PLAYING A VIDEO RELATED TO A VIDEO SELECTED FROM A SEARCH.

(A) Example 7 – Scenes

Motivation: I like the Worlds Largest Rope Swing video but I don't just want to watch that video, I'd like to watch bits from it and related videos. I can do this by adding other scenes. These break the performance up, like scenes in a play. In fact you can think about what you're doing as writing a script for the computer to perform, like how a writer writes a screenplay to tell actors what to do.

BUILD EXAMPLE 7. PLAY.

(A) Example 8 – Variables

Motivation: If I want to change the video I have to change it in a load of place, I'd like to save myself work by setting it in one place and referring to it later. I'm going to take the previous example and use variables instead.

BUILD EXAMPLE 8. PLAY.

(B,C) Example 9 – Arithmetic and variables

Target: I like adventure sports but want to get straight into the action. I'm going to use arithmetic and variables to play the videos from half way through.

BUILD EXAMPLE 9. PLAY.

(B) HOW ABOUT YOU HAVE A GO AT CHANGING IT SO THE VIDEOS WILL PLAY FOR A QUARTER OF ITS DURATION.

This is significantly more challenging. The participant needs to identify they can use get variable expressions in the duration slot.

(C) HOW ABOUT YOU HAVE A GO AT CHANGING IT SO THE PROPORTION OF THE VIDEO THAT IS PLAYED CAN BE CHANGED IN ONE PLACE ONLY.

This is also significantly more challenging. The participant needs to identify that a number can be stored and used in later examples.

(B,C) Example 10 – Titles and comments

Target: I like to know what I'm watching and what other people are saying about it. I'm going to use a different type of scene to display the title and comments of one my favourite mountain biking videos.

BUILD EXAMPLE 10. PLAY.

(B) HOW ABOUT YOU HAVE A GO AT CHANGING IT SO YOU HAVE TO WRITE OUT THE VIDEO IN ONE PLACE ONLY.

Need to use variables.

Motivation: At the moment I have to look at three comments but sometimes I'd like to look at more and sometimes I'd like to look at less. I'd like to be able to control this.

(C) HOW ABOUT YOU HAVE A GO AT USING SOMETHING CALLED A REPEAT SCENE. YOU MIGHT LIKE TO CREATE A NEW SCENE TO EXPERIMENT WITH WHAT THIS CAN DO AND THEN TO LOAD EXAMPLE 10 SO THAT YOU CAN MODIFY IT TO USE THE REPEAT SCENE TO ACHIEVE THIS CONTROL OVER THE REPETITION OF THE COMMENTS.

This is particularly challenging. In this task we're not explaining to the participant directly what the repeat scene does but showing them how they can find out what something can do just by experimenting. The key thing is we're introducing a form of interaction called exploratory design where the desired end state is not known in advance. The participant knows they want to repeat the comments but they don't know what a repeat scene does so they have to work that out and then have to work out how to incorporate that into the example.

(A) Example 11 – Smart Music Player

I like listening to music on my computer or on YouTube but neither of them react to what I'm interested in. I'd like a Smart Music Player that reacts to my taste.

OPEN EXAMPLE 11. PERFORM.

(A) HOW ABOUT YOU HAVE A GO AT WORKING OUT WHAT THIS DOES AND HOW IT WORKS.

This is the most complicated example. The idea here is whether the participant can work out what it's doing. This isn't trivial because it introduces language features they haven't come across. The participant may want to work out what individual bits do first by perhaps deleting other bit and then build it up. This would be an example of computational thinking, breaking the problem of understanding the whole into understanding individual bits.

Appendix B - Proposal

Computer Science Project Proposal

Video Processing Language for the Raspberry Pi

C. J. Eadie, Girton College

Originator: Dr A. Blackwell

19th October 2012

Project Supervisor: Dr A. Blackwell
Director of Studies: C. Hadley
Project Overseers: Dr S. Clark & Dr P. Lio

Word count: 976

B.1. Introduction and work to be undertaken

The Raspberry Pi is a cheap credit card sized computer developed with the intention to provide a platform for programming experimentation and teaching of basic computer science in schools.

The Raspberry Pi has very good graphics performance for its size and cost, however most users are not able to exploit its video processing capabilities. They have not yet been well explored by the community and so APIs, documentation and examples are early in development.

The Raspberry Pi has a strong educational focus, however there are currently no projects offering video processing experimentation.

This project seeks to provide an environment for children to explore video processing by creating a visual programming language and a development environment for video processing. It will also provide an example to other developers in accessing the video processing capabilities and so make future video applications more feasible.

B.2. Starting point

A number of visual language syntaxes have been designed for similar signal and images processing tasks in the past including data flow, functional and constraint languages.

In preparation for this proposal I have performed preliminary research in conjunction with Dr A. Blackwell, Dr R. Mullens (co-founder of Raspberry Pi foundation) and Alex Bradbury (actively involved with Raspberry Pi development).

The Raspberry Pi supports OPENMAX for media applications, OPENGL ES for 3D graphics and OPENVG for 2D graphics. There is limited documentation for using these technologies on the Raspberry Pi, however there are some substantial open source projects that provide examples. In particular, OMXPLAYER (a video player made specifically for the Raspberry Pi) and the XMBC build for Raspberry Pi (a media centre).

B.3. Substance and structure of the project

As the video processing capabilities of the Raspberry Pi are a significant unknown, four options for end user programmable functions have been identified:

I. Live video transformations.
II. Video mixing.
III. Compositing video data with 2D and 3D animation.
IV. 3D animation without video.

The core major work items are:

1. Determining a set of end user programmable functions through an evaluation of the video processing capabilities of the Raspberry Pi.

2. Developing an API to access the end user programmable functions identified in (1).

3. Designing a visual language syntax. This will involve a review of existing visual languages, consideration of the tools for developing an editor and specification of the syntax to control the functions identified in (1).

4. Developing an editor for the visual language.

5. Developing an interpreter for the visual language. This would integrate the language selected in (3) with the video processing capabilities accessed through the API developed in (2).

6. A small structured user study with adults to evaluate the user experience and record usability problems.

The project could be extended with the following major work items:

A. Extending (6) to a large structured user study with children.

B. Adding the ability to use external content sources, such as YouTube videos.

C. Creating an environment where the user can explore data structure and algorithms through analogy to content source structure. This would extend the video processing experimentation features to general-purpose computer science experimentation.

D. Adding dynamic composition of programs, where changes to the program are immediately applied to a running instance.

B.4. Success Criterion

The main success criterion will be the successful creation of sample programs in the visual language that demonstrate video processing behaviours from a test suite of end user scenarios.

In addition, the studies described in "Substance and structure of the project" will be used to scientifically evaluate the usability of the visual language.

B.5. Plan of work

The work will be divided into two-week packages with the intensity specified to account for other activities.

Package 1 **22/10/2012** **-** **04/10/2012**	**Intensity:** High. **Milestones:** • Prepare work environment. • Revise and research theoretical background. • Develop demos of key technologies.
Package 2 **05/11/2012** **-** **18/11/2012**	**Intensity:** High. **Milestones:** • Identify and measure video processing capabilities. • Identify set of end user programmable functions.
Package 3 **19/11/2012** **-** **02/12/2012**	**Intensity:** Low. **Milestones:** • Develop API for each end user programmable function identified.
Package 4 **10/12/2012** **-** **23/12/2012**	**Intensity:** High. **Milestones:** • Review existing visual languages. • Review tools to assist development of editor. • **Specify visual language syntax.**
Package 5 **31/12/2012** **-** **13/01/2013**	**Intensity:** High. **Milestones:** • Develop graphical editor for manipulation of visual language. • Substantially complete development of interpreter for visual language.
Package 6	**Intensity:** High. **Milestones:**

14/01/2013 - 27/01/2013	• Complete interpreter development. • Integrate visual language, editor, interpreter and API for end user programmable functions. • Complete written Progress Report. • Present oral Progress Report.
Packages 7 and 8 **28/01/2013 - 24/02/2013**	**Intensity:** Medium These packages are for working on extensions. See "Substance and structure of the project" for details.
Package 9 **25/02/2013 - 10/03/2013**	**Intensity:** Medium. **Milestones:** • Prepare evaluation experiments.
16/03/2013	**Cambridge Hands on Science – Crash Bang Squelch Event – Extension Opportunity**
Package 10 **11/03/2013 - 24/03/2013**	**Intensity:** Medium. **Milestones:** • Perform evaluation experiments.
Package 11 **25/03/2013 - 21/04/2013**	**Intensity:** High. **Milestones:** • Complete first major draft of dissertation – all sections outlined, key sections complete.
Package 12 **08/04/2013 - 21/04/2013**	**Intensity:** High. **Milestones:** • Complete second major draft – all sections complete.
Package 13	**Intensity:** Low. **Milestones:**

22/03/2013 - 05/04/2013	• Complete final draft.
Package 14 06/05/2013 - 17/05/2013	**Intensity:** Very low. **Milestones:** • Submit dissertation early to concentrate on examination revision.

B.6. Resource Declaration

I have secured access to Raspberry Pi's and supporting hardware through the Computer Laboratory.

I will develop on my personal computer (MacBook Pro running Mac OSX) and directly on the Raspberry Pi.

The small user study in the core scheme of work will require adult participants. The larger user study extension would require child participants.