BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program TW

Developing methods for integrating assistive software by example

By: Leonhard Hauptfeld

Student Number: 1510768031

Supervisor: Ing. Martin Deinhofer, MSc

Vienna, January 16, 2018

FH University of Applied Sciences
TECHNIKUM WIEN

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Vienna, January 16, 2018 Signature

Kurzfassung

In einer sich rapide entwickelnden Welt entstehen täglich neue experimentelle Softwaretechnologien. Manche davon etablieren sich, manche sind schnell veraltet. Durch den Nutzen dieser Neuerungen in dem Feld der Assistiven Technologien bieten sich jedoch große Möglichkeiten an. Dafür müssen neue Komponenten aber aufgrund der schnellen technischen Entwicklung ebenso schnell in existierende Assistive Toolkits integriert werden. Diese Arbeit beschäftigt sich aus diesem Grund mit dem Prozess der Integrierung neuer Software in ein AT-Toolkit. Dabei werden zuerst eigene Methoden für die Integrierung von Software in einem Assistive Technologies-Umfeld entwickelt und besprochen. Diese enthalten auch für diesen Kontext angepasste Wege zur Analyse von Quell- und Zielsoftware inklusive Erkennung von Gemeinsamkeiten sowie Erweiterbarkeitskonzepten. Aus den Ergebnissen dieser Analyse wird ein Konzept der Implementierung erstellt. Weiters werden mehrere State-of-the-art Technologien wie die Programmiersprachen Java, C++, Python und die Datenaustauschformate XML und JSON anhand ihrer Interoperabilität und Nutzen im assistiven Umfeld untereinander gegenübergestellt und bewertet. Anschließend werden die Ergebnisse im Zuge einer Integrierung von Gestenerkennung mit Intel RealSense-Technologie in das AT-Toolkit "Assistive Technology Rapid Integration & Construction Set" angewendet. Abschließend wird die Effektivität der Methodik anhand der erfolgten Implementierung bewertet und auf positive bzw. negative Aspekte hin analysiert. Aufgeteilt nach Anwendungsfällen werden die Methoden diskutiert und bewertet sowie "Best Practices" aufgelistet.

Schlagworte: Assistive, Technologie, Software, Integration, Einbindung, Bewegungserkennung, Computer Vision

Abstract

In today's rapidly advancing world, new software technologies emerge at an enormous pace. Some may develop to be the next great standard, while others perish rapidly into obscurity. The uses for these emerging technologies in the field of assistive technology are often manifold but the rapid pace of technology demands fast adaption of this software to existing toolkits, or they themselves might soon disappear. This paper brings up the key challenges involved with this implementation and adaption process and possible methods for solving them. First, custom approaches to integration of software are developed, considering the assisstive technologies context. These include custom ways of analyzing source and target software and collecting data, such as similarities and interconnectivity options. Furthermore, state-of-the-art technologies, such as the programming languages Java, C++ and Python, as well as the data exchange formats XML and JSON, are weighed against each other for their usability in assistive technologies. The results of these discussions and the methods developed are then applied to an integration of gesture recognition utilizing Intel RealSense technology into the AT-Toolkit "Assistive Technology Rapid Integration & Construction Set," or "AsTeRICS." The paper concludes with a rating of effectiveness, as well as positive and negative aspects for the methods discussed, using the aforementioned implementation as a guideline. Sorted by application, the different methods and processes are discussed and lists of "best practices" are developed.

Keywords: assistive, technology, software, integration, motion recognition, computer vision

Acknowledgements

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

1 Introduction

1.1 State of the art

1.1.1 AT-Frameworks

AsTeRICS is a framework for building AT-Solutions that is based entirely on different plug-ins and their interaction. Possible plug-in types are sensors and actors. It is written in Java, utilizing native C++ libraries where necessary or advantageous.

The runtime application is implemented as a framework utilizing the OSGi platform concept by the OSGi Alliance. It therefore provides a middleware layer to all the sensor and actor plugins that they can use to connect to the main application. These plugins represent the component model and provide their services to the main application over the middleware library.

1.1.2 Computer Vision

Various computer vision libraries for research exist, but OpenComputerVision (OpenCV) has established itself as the dominant library used for most computer vision applications, becoming almost synonymous for the technology. It contains a vast selection of structures and algorithms for loading, storing and transforming computer vision relevant data.

OpenCVs core is implemented in C, making it a very resource-efficient library. It is available for every major operating system and processor architecture. Official bindings with documentation exist for C++, Python and Java with many more unofficial ones for almost every other important programming language.

1.1.3 Programming languages

Java

Java[6] is a programming language invented by Sun Microsystems, now owned by the Oracle Corporation, which also supports its most prominent implementation. It compiles into non-native Java Bytecode that runs on a Java Virtual Machine (JVM). As a result, any compiled Java Code can run on any hardware that runs such a JVM, making Java almost entirely platform independent.

It features a very strict object oriented programming model with, in comparison with similar languages like C#, little convenience features, making it one of the more challenging program-

ming languages for development. Additionally, the platform independence of the Java Virtual Machine means that Java programs are quite more resource intensive than those of natively executed languages[2].

Java can be coupled to native Machine Code written in C or C++, giving it additional versatility at the cost of complete platform independence. As such, bindings exist for many popular C/C++ libraries, including OpenCV. The OpenCV binding to Java is officially supported by the core OpenCV team.

C++

C++ is a programming language defined and standardized by the International Organization for Standardization (ISO)[3]. There are multiple implementations of this standard (each differing slightly), the most prominent ones being as part of the free GNU Compiler Collection project and Visual C++ by Microsoft.

Python

Python[7] is an interpreted programming language with a simple syntax, partially derived the language "ABC", a simple language with the original purpose of teaching children programming. Despite its simple appearance it is closely tied in with C and C++, with many libraries being direct ports from these programming languages. Prominent examples of this include OpenCV (supported by the core development team), QT and many more. Furthermore, it also has a number of libraries for scientific computing available exclusively to it, such as the hugely popular numpy.

Being an interpreted language, Python is not as fast as most compiled languages. However, single methods of the C/C++ libraries will perform almost as fast as their native counterparts, since they just refer back to their native assemblies. As such, program logic is mostly implemented in Python, while performance critical algorithms are written in C/C++.

The versatility coupled with the easy syntax makes Python a very attractive technology to use for prototyping and scientific computation.

1.1.4 Data exchange formats

JSON

JavaScript Object Notation is a text-based data exchange format based on the object definition syntax in the programming language JavaScript. It is defined by the Standards RFC8259[4] and ECMA-404[1]. The simple syntax with only 4 object types (string, number, array, object) makes it easily human-readable while the same object syntax is native to a few programming languages (like JavaScript and Python) leading to fast and easy processing.

JSON data is most commonly transmitted via the Hypertext Transfer Protocol (HTTP) from a web service utlizing the Representational State Transfer (REST) model. The exact format

of JSON and the means of exchange are not standardized, leading to vastly different types of usage and often completely incompatible web services merely able to parse the data, not process it. No acknowledged standards exist. An attempt at providing standardized JSON schemas has been made through the JSON-Schema project[5].

XML

The Extensible Markup Language is also a text-based data exchange format specified[8] by the World Wide Web Consortium (W3C). It uses a system of tags and attributes to represent hierarchical data.

Its wide use in commercial applications gives it a large advantage over JSON when it comes to standardization. There is an official XML Schema recommendation and a Document Type Definition (DTD) standard by the W3C. Furthermore, web services can be described in the Web Services Description Language (WSDL), which is a derivative of XML. A standardized way of transforming XML exists in the XML Stylesheet (XSLT) standard. XLST provides the means to transform any XML document from one web service into a completely different XML arrangement for use with another.

2 Method

2.0.1 Integration methods

Analyzing target software

This proposed method starts with an analysis of the technology used to build both the target software and the technology to be integrated at different tiers.

At the core level, this is the programming language used to implement it. Integration is naturally easiest when these are the same or very similar. A method used in one program to encode an output is guaranteed to have an accompanying decoding method in the other program.

A level above this lie any direct extensibility interfaces. This includes all methods of directly attaching your own code to the target software. Examples for this are loading of shared libraries (dll, so, etc.) or Java JAR files.

The last level to analyze is any networking components. The application might expose a REST API via a local HTTP server or other information via a custom protocol and a local socket.

These levels are sorted from lowest to highest level interfacing from a technical standpoint. This is by no means a way to rank their usability, as the other software might have better support for the networking components or another layer might have shortcomings in a certain technology that makes it unable to support usage for assistive technology. For example, a JavaScript extension for an application is much more suited for interfacing via the networking layer than ones on a lower level. However, if the target software does not implement some kind of security protocol on that layer, it might not be suited for exchanging medical data.

Those are some reasons why the integration of software, especially complex technology, might not be extremely straightforward and why it is important to choose the right technology.

Choosing the right technology

The simplest proposed methodology is to find the lowest level interface that the software to be integrated supports at its own lowest level, and use these end points to integrate the two pieces of software. This crude approach might be well suited if the two pieces of target software are very similar in nature. A directly extensible C++ program would not use a web layer to extend another C++ application when it can be directly connected via shared library plugin extension and the target software supports that method.

Another point to consider is the difference of closeness to hardware between the target software and the software to be integrated. If the target software is of a higher level it us usually

easier to extend it with a lower level language because high level programming languages are built on low level technology. A good example for this is the integration of native C++ code into Java applications using the Java Native Interface. With this in mind, keeping the level of the software to be integrated at a higher level is more desirable, due to high level programming languages usually being easier to develop with and the loss of high level functionality when couple with low level technology, like easy platform independence when using the Java Native Interface.

The existing technology available to the programming language to be integrated is also important. While it is not usually possible to directly change things like dependencies about the target software, the software to be integrated can often be quite freely extended.

TODO:
Way of attachment
Platforms / Development platforms
choice of dev tools
application to asterics+opencv+realsense

2.1 Prerequisites

2.1.1 Utilized software

For C++ development, the IDE "CLion" by IntelliJ was used, together with the integrated CMake build system. The libraries used in the native C++ library were OpenCV, librealsense and the Java Native Interface.

For Java development, the IDE "IDEA" by IntelliJ was used. The build system of the ASTeRICS framework is Ant, and the integration of Ant into IDEA was set up for the build process. The only external dependencies of the Java program is the ASTeRICS middleware and the native library developed for the recognition.

2.1.2 Required hardware

2.2 Implementation

2.2.1 Implementation of Handtracking

RealSense and OpenCV

The open source library "librealsense" by Intel used provides an easy means of accessing the RealSense camera directly from C++ code. A pipeline object is created and used in a loop to get the newest image data. For this project, only data from the camera's depth sensor is

needed. The newest frame of depth data is received every loop iteration and converted to a standard OpenCV Matrix ("Mat") using a built-in librealsense method.

```
[...]

// Declare RealSense pipeline, encapsulating the actual device and sensors
rs2::pipeline pipe;
// Start streaming with default recommended configuration
pipe.start();

isRecognizing = true;
int previousFingers = 0;

using namespace cv;
while (gesture_visualizer.is_open() && isRecognizing)
{
    rs2::frameset data = pipe.wait_for_frames(); // Wait for next set of frames
        from the camera
    rs2::frame depth = color_map(data.get_depth_frame());
    //rs2::frame depth = data.get_depth_frame();

    // Query frame size (width and height)
    const int w = depth.as<rs2::video_frame>().get_width();
    const int h = depth.as<rs2::video_frame>().get_height();

    // Create OpenCV matrix of size (w,h) from the colorized depth data
    Mat image(Size(w, h), CV_8UC3, (void*)depth.get_data(), Mat::AUTO_STEP);
    [...]
}
```

Code 1: RealSense image capture and conversion

The default capture mode of the depth sensor outputs a BGR (Blue, Green and Red color components) image in 8 bit depth, as indicated by the 8-bit depth 3 channel ("CV_8UC3") format of the matrix. For easier processing, a grayscale image is more suited. Therefore, a color scheme is set using a "colorizer" class built into the librealsense library.

```
// Declare depth colorizer for pretty visualization of depth data
rs2::colorizer color_map;
// Use color scheme option 2 (grayscale, distant black, close white)
color_map.set_option(RS2_OPTION_COLOR_SCHEME, 2);
```

Code 2: RealSense image colorizer

The resulting captured image in OpenCV Matrix format is then passed to the library's own "recognizer" class for further processing.

Gesture recognition

While it is not the purpose of this paper to research and explain gesture recognition using the OpenCV framework, parts of it are relevant to the later discussion of integration and as such a short summary of the recognition technique is given here.

The "recognizer" class is where all of the image segmentation and hand detection occurs. Its method "get_hand_model" takes the Matrix depth frame and outputs a "hand_model" structure, containing all the necessary info about how the hand was detected and, as a final result, how many fingers were detected as extended. The structure is as follows:

```
1 struct hand_model{
2    int num_fingers;
3    std::vector<cv::Point> hand_contour;
4    std::vector<int> hand_hull_indexes;
5    std::vector<cv::Vec4i> hand_defects;
6    std::vector<int> finger_defects_indexes;
7    cv::Mat display_frame;
8 };
```

Code 3: Hand Model Structure

The following steps are then applied to get the hand model from the passed Matrix:

Conversion The grayscale BGR matrix is converted to a single grayscale channel

Segmentation The arm is segmented from background noise. This is achieved by getting the median value of a small center region, and then removing all depth data within a certain threshold of that median. Gaps caused by an insufficient threshold are filled in and everything is colored gray. The segment of the image touching the middle (the hand) is then filled white and all remaining gray background noise is removed. The result is an image with a white hand contour in the middle.

Defect Recognition Contours are drawn around the white segments in the frame. The contour with the biggest area is selected as the probable shape of the hand. It is smoothed to remove noise and a hull is drawn around it. The hull is a contour drawn around the outside of the hand, not accounting for "valleys" in the actual contour. This hull is then analyzed for convexity defects, which represent those valleys.

Hand Processing The convexity defects are analyzed for the angle of the "valley". If it is smaller than a threshold, it is counted as an extended finger. Otherwise, it is discarded.

All the above mentioned processes can be found in the "recognizer.cpp" class file, split into appropriately named methods. Comments in those methods describe the functionality in greater detail.

Tracking and Information Window

TODO: choosing between target software UI / own UI
TODO: Results, discussion and ratings

2.2.2 Integration into framework

Creation of AsTeRICS plugin

Plugin Configuration and Options

Java Native Interface

3 Results

3.1 Technology integrated

3.2 Method effectiveness

4 Discussion

4.1 Steps for approaching integration

4.1.1 Analyzing extensibility concepts

4.2 Combining different programming languages

4.2.1 Virtues and choosing languages

4.2.2 Difficulties and compensation

Bibliography

[1] ECMA INTERNATIONAL: *ECMA-404 – The JSON Data Interchange Syntax*, 2017.

[2] FOURMENT, M. and M. R. GILLINGS: *A comparison of common programming languages used in bioinformatics*. PubMed Central®, US National Library of Medicine, 2008.

[3] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *C++ language specification*.

[4] INTERNET ENGINEERING TASK FORCE: *RFC 8259 – The JavaScript Object Notation (JSON) Data Interchange Format*, 2017.

[5] JSON SCHEMA PROJECT: *JSON Schema Website*.

[6] ORACLE CORPORATION: *Java, Official Website*.

[7] PYTHON SOFTWARE FOUNDATION: *Python Programming Language, Official Website*.

[8] WORLD WIDE WEB CONSORTIUM: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, 2008.

List of Figures

List of Tables

List of Code

List of Abbreviations

AT Assistive Technology

W3C World Wide Web Consortium

JSON JavaScript Object Notation

XML Extensible Markup Language

AsTeRICS Assistive Technology Rapid Integration & Construction Set

OpenCV Open Computer Vision

IDE Integrated Development Environment

REST Representational State Transfer

API Application Programming Interface

HTTP Hypertext Transfer Protocol

A Anhang A

B Anhang B