Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Extensible Platform for Real-Time Feedback in Presentation Training

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

Mohamed Zahir

Promoter: Prof. Dr. Beat Signer
Advisor: Reinout Roels

Academic year 2015-2016

Abstract

Presentations are an important form of communication, mainly used to deliver a message or to persuade people. Identified centuries ago, presentations are practically used everywhere nowadays, in business environments, educational environments, and more. Since not everyone is born as a good presenter, a presentation can nevertheless be performed effectively by mastering the required communication skills. Therefore, we conducted a literature study in the field of communication, identifying the most important elements present in the communication process and communication aspects, both verbal and nonverbal. Unfortunately, presentations can be regarded as being stressful situations, inciting the speaker to perform the same mistakes unconsciously. Therefore, using informative feedback has shown its effectiveness, allowing the speaker to adapt their behaviour. Since human feedback is not always affordable, sensor technologies have proven their potential, providing real-time feedback to the speaker. Current multimodal feedback systems are limited in the sense that communication aspects, input and output modalities are identified beforehand. However, a presentation should be adapted to the context and to the audience's background, which implies that there is no "fixed" set of rules that guarantees a good presentation for all scenarios and cultures. Therefore it is important that a multimodal feedback system is extensible at the level of input and output modalities as well as the detection rules that trigger feedback.

This thesis proposes an extensible platform providing real-time feedback to the speaker. The speaker is less constrained and able to customise the system depending on the presentation's context and needs. Moreover, extensibility has been identified as being the most important factor of the system, allowing developers to build plug-ins for the inclusion of different input and output modalities, as well as plug-ins for detecting and providing feedback on communication aspects.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

First, I would like to express my deep sense of gratitude to both my promoter, Prof. Dr. Beat Signer, and my supervisor, Reinout Roels, allowing me to conduct my research. Thank you for always being available, patient, and for your precious guidance and support throughout this thesis.

I would also like to thank my family and friends for their support during my academic years which made me able to achieve this important goal.

Contents

1 Introduction

1.1 Research Context

The importance of public speaking has been identified at least 2500 years ago, especially during the ancient Greek ages where the art of public speaking or rhetoric was used as a means for persuading people [9]. Even if the art of public speaking has changed during the centuries, it still remains important in business environments nowadays and in our daily life [10]. Since the goal of the speaker is to deliver a message to an audience, it is of great importance that it has to be done effectively, and this by mastering the needed communication skills (verbal and nonverbal communication skills) [11]. Not everyone is born as a good presenter. However the needed skills can be improved by making use of the different materials available, such as courses, seminars, books, etc. [12]. Nevertheless, practicing alone will not guarantee improvement, since the possibility of making the same mistakes without realising it can occur. Therefore, feedback can be used to avoid speakers from making the same mistakes. Although its effectiveness has been shown in learning [13], it is not always affordable to acquire human feedback during presentations [12]. The advances in real-time sensor technologies have proven their potential for these circumstances [14] and could therefore be used for building a powerful system designed to provide automatic feedback in real-time during presentations without any human intervention, which is one of

the main goals of this thesis.

1.2 Problem Definition and Justification

Multimodal interaction has shown its potential compared to other interaction styles [15] by offering a more intuitive and natural way of interaction between humans and machines, and by extending the spectrum of applications in which the traditional WIMP-style user interface found its limitations. Multiple researchers have taken presentations to the next level, by developing multimodal systems capable of measuring the speaker's nonverbal communication skills in order to rehearse and improve the quality of presentations.

Even if studies have provided empirical findings for the design of multimodal systems, practically none of the presentation tools offering feedback to the speaker allow their systems to be extensible in certain ways. Studying communication reveals that there are no fixed rules for a "good" presentation, since it depends on culture, context, language, and so forth. Unfortunately, current available systems tackle only certain aspects of oral communication which are defined in advance, limiting the power of multimodal systems in the field of presentations.

Therefore, this thesis presents *Presentation Mate* (PRESMA), a framework designed to provide feedback to the speaker in real-time during presentations. Taking advantage of existing multimodal systems and technologies, *PRESMA* allows the extensibility of input modalities, nonverbal communication skills (rules and plug-ins) and output modalities, allowing the tool to be customisable according to the speaker's conception of an ideal presentation. A basic set of plug-ins and communication aspect rules are provided based on research work, also provided in this document for anyone interested in the study of Human-Computer Interaction and Communication.

1.3 Objectives and Research Method

The main goal of this thesis is to design an extensible system capable of providing informative feedback to the speaker during presentations. Before being able to conceptualise the system, it is important to understand the issues that could occur during presentations, and which kind of systems are more suitable for assisting the speaker during a presentation session. Therefore the research methodology of this thesis was performed as follows:

- A study on Multimodal Systems in general in order to get an insight on how this kind of system are more suitable for assisting speakers during presentations.

- A literature study in the field of Pedagogy, Psychology, and Communication to identify and understand the communication process. This will provide more insight on the issues and difficulties that could occur while performing presentations.

- Afterwards, conducting an investigation, analysis and comparison of existing Multimodal Systems designed to provide informative feedback during or for the rehearsal of presentations. This will help to identify the gaps, which will be useful for the design and implementation of the proposed approach in this thesis.

- Finally, a proof of concept, *PRESMA*, based on the findings of the conducted research.

1.4 Structure of the Thesis

First, a literature study in the field of Human-Computer Interaction is conducted is Chapter 2, more specifically about WIMP and post-WIMP interfaces. This serves mainly to accentuate the importance of the emergence of Multimodal User Interfaces and Interaction Techniques. Based on this literature study, the core concepts present in multimodal systems, such as multimodal fusion and fission as well as the CASE and CARE modelling frameworks are explained, making it able to understand how meaningful information about users can be extracted from input modalities.

In the first part of Chapter 3, a literature study in the field of Pedagogy, Psychology, and Communication is provided, explaining the basic concepts of the communication process as well as the important communication aspects. The second part focusses more on the field of Computer Science, more specifically about existing multimodal systems providing informative feedback to speakers. The working of these systems are analysed and compared, making it possible to identify their advantages and inconveniences used to motivate the proposed approach of this thesis.

Chapter 4 explains the architecture of the *PRESMA* and the technologies used to successfully achieve this project.

The implementation of the system is divided in Chapter 5 and Chapter 6. The first part explains the core code of the system, allowing the inclusion of input and output modalities, as well as the detection of communication aspects. The second part consists of applying the core concepts defined in Chapter 5 by the means of plug-ins, such as the inclusion of the Microsoft Kinect sensor, nonverbal communication aspect detectors, and output modalities.

Finally, Chapter 7 concludes this thesis with the main contribution, as well as a number of possible future work.

2

Background

Computer technology has in the past decades made great advances and took an important place in everyone's daily life. Computer users are nowadays not mainly computer professionals, such as programmers or designers, but also *discretionary* users [16]. Opening this kind of technology to a broader spectrum of users did not only have advantages in terms of processing large amounts of information, but also caused users to become angry and frustrated due to the designers' lack of user knowledge and understanding. Both computer and cognitive science are needed to develop tools, techniques, methodologies, to learn how to build efficient computers that are easy to use [16]. *Human-Computer Interaction (HCI)*, or often called *Man-Machine Interaction*, became rapidly important with the emergence of computers [17, 18]. It is a cross-disciplinary area *"concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them."* [18]. Booth [16] suggests that ten major disciplines contribute towards HCI (as shown in Figure 2.1) serving to understand the user's needs, developing software and hardware interaction techniques, creating intelligent and adaptive, implementing usable and efficient interfaces, and so forth [16, 17, 18].

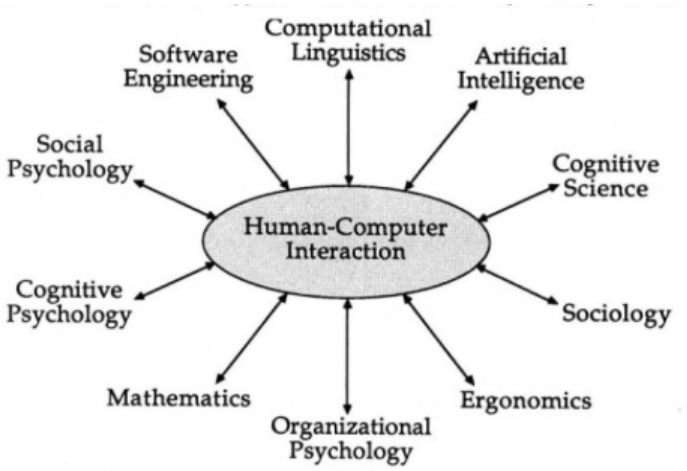


Figure 2.1: An overview of the disciplines involved in HCI [16].

Since the study of Human-Computer Interaction is relevant to this thesis, the following sections discuss the evolution of user interfaces and interaction techniques, and presents multiple kinds of user interfaces as well as their effectiveness in terms of usability, functionality, and performance.

2.1 User Interfaces and Interactions

A user interface or *UI* refers to means in which the user interacts with a machine by using input devices, such as a keyboard, mouse, touchscreen, digital pen, game controller and so forth [19]. During the past centuries one may notice that user interfaces have significantly changed, going from user interfaces using mechanical components to user interfaces allowing the user to interact with machines on a more natural way [20, 21]. The prediction made in 1962 by Douglas Engelbert, mostly known as being the inventor of the mouse, stating that *"In 20 or 30 years, you'll be able to hold in your hand as much computing knowledge as exists now in the whole city, or even the whole world."* [22], has become true with the evolution of hardware devices, which allows software to be embedded practically everywhere. Even though the current methodology for designing user interfaces remains powerful, it nevertheless does not fulfil all the requirements for new future computing devices and requires therefore a new way of thinking about user interfaces and interaction [23].

Figure 2.2: The IBM 029 batch system [24].

2.1.1 The Evolution of User Interfaces

Since in the early 20th century computing devices were not as powerful as nowadays microwaves, user interfaces were considered to be overhead and the processor was utilised with as less overhead as possible [24]. The user interfaces were also too complex and only expert users were allowed to use them. A good example is the *batch computing system*, as presented in Figure 2.2, where punched cards were used for describing the program and used as input of the user interface. Attached printers were used as output media in order to generate the results of the program.

Evolved from the batch computing systems, *command-line interfaces* present a kind of request-response interaction style. Such systems, as presented in Figure 2.3, were mainly composed of a keyboard for requesting demands by typing commands in textual forms and, depending on the system, a screen or a printer for presenting the results. A possible disadvantage that can be attributed to such systems is that it required the user to spend serious learning time for mastering the commands (possible requests), but a real-time and interactive way of working was offered, which was not the case in the previous systems. Software was no more designed to be computation-intensive but also presentation-intensive [24]. Even if the user interfaces have evolved quite a lot during these last years, the command-line interfaces have still survived and are to be found in recent operating systems [25].

In the 1960s Douglas Engelbert, inspired by his experiences with early graphical displays and by Vannevar Bush's *As we may think* vision [26] of

Figure 2.3: The VT100 command-line [24].

what we today may call hypertext, presented a demo of a hypertext system composed of a multiple-window graphical interface, which was controlled by a three-button mouse [24]. The user experience took another dimensions that manufacturers started to design systems based on the Xerox PARC WIMP (windows, icons, menus and a pointing device) graphical user interface. In 1984, the Macintosh popularized the WIMP graphical user interfaces (as presented in Figure 2.4) and later copied by Windows with the Personal Computer (PC), which made computing devices more accessible to the public [27]. Even though WIMP interfaces have not changed a lot [25, 27] during the last decades, they remain still successful and important, because:

- It became possible for young children, managers and non-professional users at home to become comfortable with computers. The WIMP user interface has provided a way to learn easily to work with and to gain knowledge because of the fact that the look and feel of these user interfaces are consistent compared to previous generations of user interfaces that focussed only on the performances and functionalities of applications [25, 27].
- It is a kind of abstraction that hides the most technical issues from the application to the user. In this way even the most unexperienced users can perform complex tasks [27].

Figure 2.4: The Apple Macintosh (1984). Image taken from http://www.computerhistory.org/timeline/1984/

- They have been explored, examined and decorticated in the last two decades, such that design principles have been defined for making them easy to understand and to learn [27].

Nevertheless, even if WIMP user interfaces have been successful and important through the last decades, they are not fit enough to match future computing systems due to some Human-Computer Interaction limitations [23] and carry a number of disadvantages, which are [25, 27]:

Following the disadvantages of the WIMP user interfaces, one may notice that new thinking about user interfaces is necessary [25, 27, 23]. User interfaces do not need to use toolbars, menus and forms anymore but must rely on more natural way of interaction, such as, for example, gesture recognition or even speech recognition, which are called *post- or non-WIMP user interfaces* or *next generation user interfaces* [25, 27, 28]. This fourth generation of user interfaces must at least contain one interaction technique that is not dependent of classical 2D widgets (e.g. menus or icons) [25]. Compared to traditional WIMP user interfaces that inherit automatically from a serial, turn-taking dialog between the user and the computer with a single input and output stream, non-WIMP user interfaces must instead support continuous multi-mode interaction between the user and the computer by making use of multiple parallel, asynchronous channel devices [29, 30]. This style of interaction are natural and easy to learn because it reflects the user's pre-existing interaction skills with non-digital real world and real world objects [29, 31]. However, such kind of interaction are more difficult to build, share

- The more complex an application is, the harder the interface becomes, which decreases the user's effectiveness of learning.
- Certain applications are so large that users do not want to upgrade and are satisfied with small subset they know.
- Users spend too much time manipulating the user interface that the application itself.
- Expert users are frustrated of performing too much "point-to-click" operations and prefer keyboard shortcuts.
- Mousing and keyboarding are not suitable for all the users, especially for disabled users.
- It becomes difficult to perform 3D tasks with conventional 2D control widgets.
- Not suitable for shared tasks. WIMP user interfaces are designed to to be used by a single person.

Figure 2.5: Disadvantages of WIMP style user interfaces.

and reuse, since they require inventive low-level programming approaches and high skills [29]. Also speaking in terms of interaction, non-WIMP interaction has to be focussed on both new kinds of interactions (e.g.: virtual reality, mixed reality, augmented reality, tangible interaction, ubiquitous and pervasive computing, mobile interaction, and etc.) and the output presentation of information [27, 31].

The following sections present different kinds of next generation or post-WIMP user interfaces that are the subject of research and even available on the market, and provide a more natural way of interaction between humans and machines. Unfortunately, there are plenty of different types of next generation user interfaces available nowadays [27], but only a few of them will be discussed to provide a better understanding of what to expect from this fourth generation of user interfaces.

2.1.2 Tangible User Interfaces

Tangible User Interfaces (TUI) are interfaces that provides a physical form to digital information and computations allowing them to be manipulated by human hands. Such interfaces are often called "graspable" user interfaces and are built to support the sophisticated developed human skills for sensing and manipulating their environments [32]. According to Burghart [27] easy to manipulate real life objects are used to delegate the representations of data from a visualisation on the screen, which makes the interaction more intuitive compared to traditional GUIs (using a keyboard and a computer mouse) that are inconsistent with the way humans interact with the environment within they live [32]. While GUIs are serving as a more general interface to emulate various tools using pixels, TUIs are a new way of materialising Mark Weiser's [33] vision of Ubiquitous Computing [32].

The key idea of TUIs, as shown in Figure 2.6, is to provide input devices that act as a bridge between the physical and digital world. Such input devices are both a tangible representation of the coupled digital information and a way of controlling that digital information. An intangible representation is added to the model to visualise the digital information to compensate the limitations of physical objects, like for example, changing the physical properties of physical objects.

Multi-touch surfaces are often used with tangible user interfaces to compose what is called *hybrid surfaces* or *Hybrid User Interfaces* (HUI). Such interfaces combine multiple types of interaction techniques to form one interface [27, 34], like the *ReacTable* and Microsoft *Surface* [34] shown in Figure 2.7 and Figure 2.8.

"Radical Atoms" is a further step made by Ishii et al. [35] stating that all

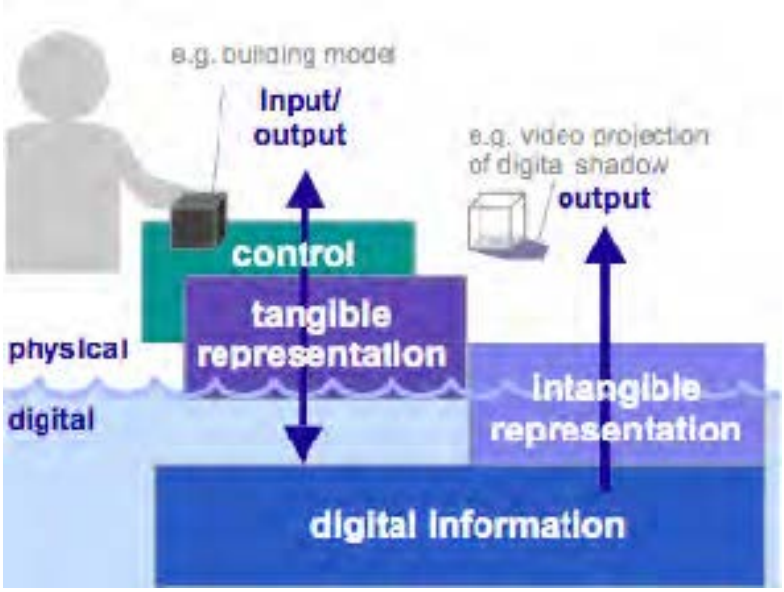


Figure 2.6: The key idea of Tangible User Interfaces [32].

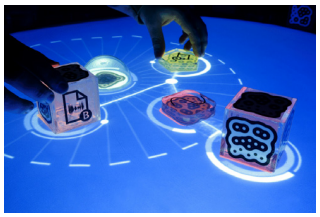


Figure 2.7: ReacTable hybrid surface [34].

Figure 2.8: Microsoft's hybrid Surface [34].

digital information may in the future be manifested in the physical world. The main idea of this assumption, as presented in Figure 2.9, is based on the iceberg metaphor stating that with GUIs the user can only see information from the digital world through a screen, just like the iceberg that can only be seen through the surface of the water. TUIs are a further step allowing the user to interact with the digital world through physical objects, like the top of the iceberg that sticks out of the water. Finally, Radical Atoms would be a manifestation of digital information in the physical world based on hypothetical dynamic physical materials that are transformable and reconfigurable [32, 35].

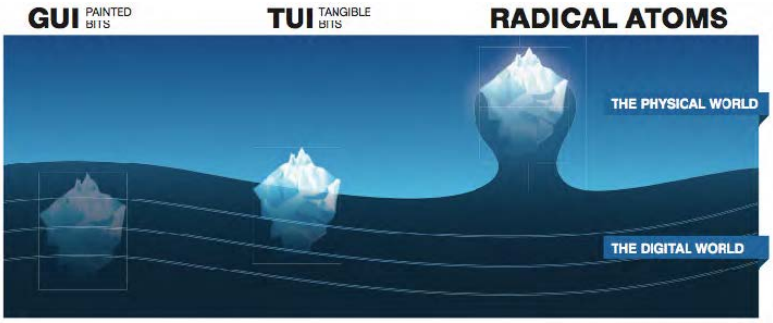


Figure 2.9: Radical Atoms. The iceberg metaphor [35].

2.1.3 Organic User Interfaces

Organic user interface (OUI) is a rather new research area working on non-planar, non-flat and flexible organic displays [36]. They allow users to interact with organic objects with both hands, like in the real world. The feeling of texture and temperature is incorporated into HCI and thus open new ways of interaction [36, 27]. Although OUIs share similarities with TUIs, the first one combines the input and output in one device (no spatial separation). This implies that digital information is manipulated directly without the use of physical controllers compared to TUIs. Interactions become more continuous and natural, allowing users to use everyday's gestures, like for example, bending, folding or crumpling a paper [36]. Designers are no more obligated to waste space for graphical buttons, which offers benefits to small displays.

Although organic user interfaces are difficult to obtain or not yet to be found on the market, investigating the user's experience with such devices becomes difficult. Vertegaal and Poupyrev [37] describe three interaction principles for the design of organic user interfaces:

- *"Input Equals Output: Where the display is the input device."* Due to the fact that no spatial separation is present between the input and output device, the display serves both as a manipulator (input) and for the presentation (output) of digital information. The user is able to interact with the display by dynamically changing its form, touching the surface or by even integrating small sensors within the display [36, 37].

- *"Function Equals Form: Where the display can take any shape."* The flexibility property of the displays allows users to change the shape of the display, which imply the designers to develop flexible layouts that adapt to every possible form of the display [36, 37].

- *"Form Follows Flow: Where displays can change their shape."* This principle states that interaction can take place actively or passively. By twisting, bending, pulling, etc. the display, the user will passively supply input to the application. While in the future, computing devices will be able to actively present data by automatically changing their physical shapes [36, 37].

2.1.4 Ubiquitous Computing

Computing devices have nowadays taken an important and inseparable place thanks to the improvements in microprocessor technologies, making them more compact to support, organise, and mediate our daily activities [38]. The idea behind Weiser's *Ubiquitous Computing* [33] is to take advantage of embedded, wireless, and flexible software architecture technologies to help organise and mediate social interactions in every situation [38]. The concept of ubiquitous computing is based on advances from both *pervasive* and *mobile* computing, as shown in Figure 2.10.

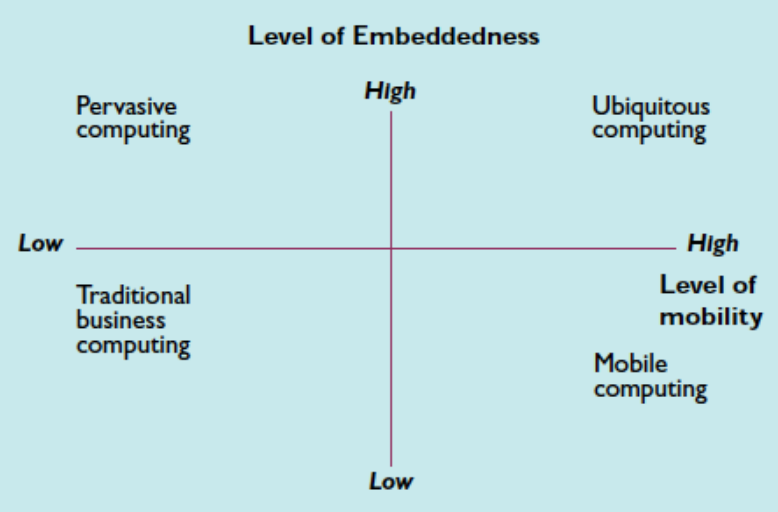


Figure 2.10: The two dimensions of Ubiquitous Computing [38].

By making computing devices more mobile, users are capable of carrying computing devices with them, which enhances the reasoning and communication capabilities of humans independently from the device's location. Accessing such a level of mobility is either done by reducing the size of the devices or by allowing them to be accessible over a network by light-weighted devices. Pervasive computing, which is the second dimension needed for making computing devices more invisible, is based on a mutual dependency involving

both devices and environments to be intelligent. Computing devices should be capable of acquiring information from the environment in which they are embedded, while the environment should be capable of detecting these devices [38].

2.1.5 Virtual, Mixed & Augmented Reality

Virtual Reality (VR) technology offers unique experiences to users, such as exploring molecules or even the interior of vehicles before the production takes place [39]. Users are immersed inside a computerised or simulated environment, called a *Virtual Environment* (VE), which does not allow them to see the real world around them [40, 41]. Sensors, such as head-mounted displays, sensor gloves, and much more are used to provide the possibility to the user to interact naturally within the environment and to transform enormous quantities of complex data into *"graspable illusions"* [40, 42]. Even if virtual reality offers the possibility to gain knowledge which would not be possible by using conventional ways, it is important to notice that it may cause some confusion, especially when users confuse unreal information acquired by the means of virtual scenarios with real knowledge. In fact, an abstract level need to be realised to eliminate the confusion between true and false, real and false data [41].

Augmented Reality (AR) can be seen as a variation of Virtual Reality (VR), but allows instead the user to see the real world supplemented with superimposed or composited virtual objects. Augmented reality can be thought as an intermediary world between Virtual Reality (VR), where the world is completely synthetic, and telepresence, where the world is completely real [40]. According to Azuma et al. [40], AR systems should not be limited to specific technologies, such as Head-Mounted Displays (HMDs), monocular or binocular systems and monitors, but should rather have the following three characteristics:

- AR systems should combine real and virtual;
- AR systems should be interactive in real-time;
- AR systems should be registered in 3D.

Combining the reality with virtual 3D objects can be useful to display information that cannot be perceived by human senses. Since computing devices are used as an *Intelligence Amplification* [43, 40] to make tasks more easier to perform for users, AR applications have proven to be very effective in

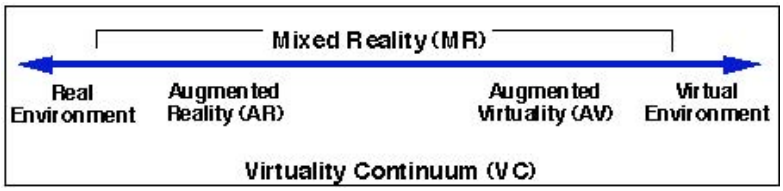


Figure 2.11: A representation of "virtual continuum" [44].

several domains, such as in the medical environment, military environment, or repair and maintenance environments [40].

Mixed Reality (MR) is according to Tamura et al. [45] *"a kind of virtual reality (VR) but a broader concept than augmented reality (AR), which augments the real world with synthetic electronic data"*. MR covers the space on the virtual continuum (as shown in Figure 2.11) from AR to AV, and allows users to see the real world and virtual world objects at the same time [44]. Users are able to see each other, to communicate between them, and to manipulate virtual information on an intuitive manner [46]. Multiple single user MR interfaces have been developed enabling persons to interact with the real world using non conventional ways, like for example projecting kinds of "X-Ray" images on the patient's body, while doctors perform biopsy tasks. The potential of MR is not only limited to advanced single user Human-Computer Interfaces, but is also ideal for Human-Human collaborative interfaces mediated by computers [47].

2.1.6 Brain-Computer Interfaces

The human brain is one of the most fascinating complex signal processing machines that is capable of extracting information from a variety of environmental signals, such as sound, sight, touch, and smell. The brain possesses a great number of neurones, all operating in parallel, which makes it an unique architecture. These facts have attracted the interest of scientists and engineers to develop means (interfaces) for measuring neural signals and decoding them [48]. A *Brain-Computer Interface* (BCI) is a system that should be capable of [48]:

- Measuring neural signals from the brain;
- Decoding neural signals by the means of methods/algorithms;
- The decoded signals must be mapped with a behaviour or action.

BCI systems have found their places in tremendous applications, especially in clinical applications serving as a means of communication for individuals with neurological diseases. BCIs can be divided in two major classes, invasive and noninvasive systems. The first class of BCIs consist of implanting electrodes in humans, while noninvasive systems are better suited for situations where surgeries are not an option [48]. This kind of interface has shown its potential by allowing monkeys and humans to move robotic arms and mouse cursors by simply activating their neural population responsible for the natural arms movements [49].

2.1.7 Similarities Between Post-WIMP User Interfaces

Even if nowadays different groups of post-WIMP interfaces exist, there still lie some similarities between them [27]. Jacob et al. [31] proposed a framework for Post-WIMP user interfaces called *"Reality-Based Interaction"* (RBI) that has as goal to unify a large subset of these new emerging interaction styles, which are based on the pre-existing knowledge of the users with everyday and non-digital real world objects. They provided a basis for interaction with machines that can almost be applied to almost all people and cultures. As presented in Figure 2.12, the four themes representing the basis of the RBI framework are:

- *"Naive Physics"* (NP): refers to the human's common sense knowledge about the basic physical principles of the physical world, such as the gravity, velocity, and so on. User interfaces simulate or use directly properties of the physical world and should therefore use metaphors, like for example the illusion of gravity, mass, and inertia in the case of the Apple iPhone, or physical constraints, in the case of TUIs, to guide the user while performing interactions.

- *"Body Awareness & Skills"* (BAS): refers to the fact that people are familiar with and understand well their bodies. They are aware of the relative positions of their limbs and possess the needed skills to control and coordinate them.

- *"Environment Awareness & Skills"* (EAS): refers to the people's skills for interacting with and orienting within environment surrounded by objects. In the case of VR, performing tasks relative to the body within a virtual world should be done using a representation of the user's body.

- *"Social Awareness & Skills"* (SAS): refers to the awareness people have to detect the presence of others, and to the developed skills to communicate, both verbal and non-verbal, to exchange objects, and to collaborate with them. Certain TUIs provide enough space and input devices to support collaboration, while other interfaces, such as VR provide a digital environment allowing users to communicate and interact with each other via avatars.

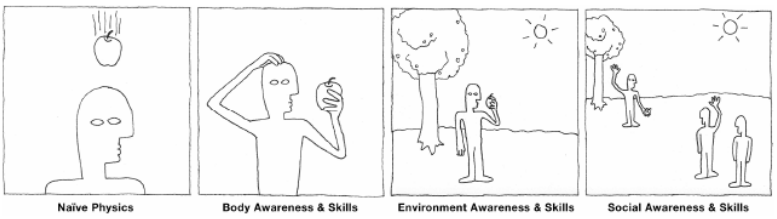


Figure 2.12: The four RBI themes [31].

2.1.8 Conclusion

User interfaces have evolved a lot during the last century, going from complex interaction styles using commands to WIMP user interfaces providing a more easy and user-friendly way of working with computers. Although WIMP user interfaces are still popular and offered the possibility to a broad range of users to use computers, it unfortunately does not fit all the HCI requirements [23]. As stated by Van Dam [25], a new way of thinking about user interfaces is required to provide a more natural way of interaction between the computing devices and users, instead of the traditional keyboard and mouse. Since then, a lot of research has been done to support the call for such a change. New post-WIMP interfaces, such as Tangible User Interfaces, Augmented, Mixed and Reality Based Interfaces, and much more have seen the light to tackle the problems of the traditional WIMP style interfaces.

Multimodal User Interfaces (MUI) are user interfaces that allows users to use their human senses as input modalities for interacting with machines on a more human way. According to Dumas et al. [15], such interfaces possess two unique features that makes MUIs distinguishable from the others:

- *"The fusion of different types of data".*
- *"Real-time processing and temporal constraints imposed on information processing".*

Since this kind of user interfaces are relevant to this thesis, an in-depth explanation of this new class of interface is provided in section the next section.

2.2 Multimodal Interfaces and Interaction

The way humans interact with the world is inherently multimodal in nature [50, 51]. The multiple human senses or sensing modalities are used both sequentially or in parallel for exploring environments and confirming expectations by providing us tons of information to support interaction. The interaction between humans and machines was in the last decades mostly done by the means of unimodal communication through single input channels [50]. Since the WIMP style user interfaces do not scale the Human-Computer Interaction (HCI) needs of future computing devices [23], multimodal interaction has developed significantly in the past years to fulfil the needs by offering a more "human" way of interacting with computers, by using the human senses [17, 15]. According to Turk [50] multimodal interfaces *"describes interactive systems that seek to leverage natural human capabilities to communicate via speech, gesture, touch, facial expression, and other modalities, bringing more sophisticated pattern recognition and classification methods to human-computer interaction."*. Such user interfaces are preferred by the users over unimodal systems where single-based recognition systems are used since they are more transparent, flexible, robust and reliable than other human-machine interaction means, and not to forget that they become more easier to learn and to use [52, 15, 53] . Since the input modalities of multimodal interfaces differ from one to another, systems become easily customisable and usable for a large spectrum of users (impaired or not). Efficiency is not the only advantage that multimodal interfaces offers in comparison to unimodal interfaces, but it has shown that reliability, error handling and precision in visual tasks have been improved, which can expand computing devices to more other challenging levels [15].

No one can deny that Bolt's *"Put That There"* system (see Figure 2.13) in 1980 was groundbreaking in terms of multimodal systems, since it was the first system to deal with more than one input channel, allowing speech and gesture recognition to work in parallel [54, 55, 28]. The user, sitting in a chair, could move objects (basic shapes) presented on a wall-sized display by simply saying the *"put that"* command, while pointing to an object, and using the voice command *"there"*, while pointing to a destination. Multiple systems started to emerge in the late 1980s, allowing users to interact by using spoken or typed natural language, speech and gesture recognition eye gaze and was

Figure 2.13: Put That There multimodal system [54].

even the opportunity to bring new modalities, such as haptic and mobile computing environments to expand the traditional desktop environments and opening doors for going beyond the WIMP-style [50].

2.2.1 Multimodal User Interfaces vs. Graphical User Interfaces

As presented in Table 2.1, Multimodal User Interfaces (MUI) differ in many ways from the traditional Graphical User Interface (GUI). One of the major characteristics of the GUI is that it is often based on the Window, Icon, Menu and Pointing device (WIMP) style and is practically used in all kinds of operating systems [15]. The user can control a cursor displayed on the screen by the means of pointing device (e.g. a mouse) in order to present information represented by icons in windows. The GUI manages only one input stream, since this interaction style requires only a single physical input device. On the other hand, MUIs are systems using a combination of input modalities and should therefore be able to handle multiple input streams [15, 29, 30, 50], like in the case of the *"Put That There"* system [54], where the parallel or multithreaded working of the input has shown a great importance, since the system had to compute the coordinates on the exact same time the user said one of the two voice commands [28]. Interpreting data for controlling GUI is a deterministic process since the position of the pointing device can be used

Differences	Graphical User Interface	Multimodal User Interface
Input Stream	Single	Multiple
Interpretation of Data	Atomic, Deterministic	Continous, Probabilistic
Processing Commands	Sequential	Parallel
Architecture	Centralised	Distributed and Time-sensitive

Table 2.1: Differences table between GUIs and MUIs [15].

or even characters typed on keyboards. This principle can not be applied in MUI because of the fact that such interfaces are dealing with input streams that must be continuously interpreted by probabilistic recognisers [15, 29, 30]. The processing of multimodal inputs happens in parallel due to the fact that multiple probabilistic recognisers are needed for the interpretation of data and are time sensitive compared to GUI. Distributed architectures are often used for MUI since algorithms demanding high resources have to be used and for dealing with the synchronisation of parallel interpretation of data [15].

Multimodal User Interfaces do not only provide the needed technical support for building much powerful systems than most GUIs based on the WIMP style, but solves a lot of their disadvantages (as discussed in 2.1.1 and listed in List 2.5). As presented in Table 2.2 , complex tasks are more suitable on MUIs, since they allow users to adapt their multimodal behaviour [56] and provide a more natural way of interaction instead of spending too much time manipulating user interfaces than the application itself [50]. Also the spectrum of users becomes much broader than traditional GUIs with mouse and keyboard interactions. Systems are easily customisable and adaptable depending on the users' needs, impaired or not [15, 50]. Multimodal interaction does not only provide extra freedom for 3D environments [57], but allows also users to work and collaborate on shared tasks [58, 59] in the same environments instead of using desktop application usually designed for single users. MUIs allows with their multithreaded input and output streams the combination of multiple media, which may supplement each other by using one medium as an input stream for acquiring commands and another stream for acquiring data, as used in the *"Put That There"* system [54]. This offers several advantages from a user interface perspective, since the user

	Graphical User Interfaces	Multimodal User Interfaces
Suitable for complex tasks	✗	✓
Manipulation of UIs	✗	✓
Interactions	✗	✓
Performing 3D tasks	✗	✓
Shared and collaborative work	✗	✓

Table 2.2: Advantages of MUIs over GUIs.

is provided a less constraint and moded feeling than with the traditional mouse interaction and will allow more precise recognition due to the natural behaviour of the human [28, 60].

2.2.2 Design of Multimodal Systems

The big issue regarding MUIs is that building such systems can be really challenging, since the design principles used for standard computing environments do not apply in multimodal systems and that even if such systems provide much more freedom compared to WIMP user interfaces, one may notice that each multimodal system may have different design decisions, since the combination of modalities, the targeted users or even the application tasks may differ from one to another [50, 61]. Researchers have tried to solve this problem by providing a set of guidelines and myths that have proven to be useful for designing multimodal systems [50]. The *"Ten Myths of Multimodal Interaction"* by Oviatt [62] has played a key role by identifying computational myths regarding how people interact with multimodal interfaces and by replacing these misconceptions with empirical findings. The ten myths of multimodal interaction includes [62, 50, 63, 64]:

- *Myth #1: "If you build a multimodal system, users will interact multimodally".* Even if multimodal systems support multiple input modes does not mean that users will interact multimodally. Users tend to intermix unimodal and multimodal interaction, but prefer the last one, since such interaction type is much more predictable depending on the action that must be performed.

- *Myth #2: "Speech and pointing is the dominant multimodal integration pattern".* Speech and pointing are a more intuitive combination of input modalities and suits better for deictic tasks, where users must select objects and perform operations on it. The reason why this combination of interaction seems to be dominant is because they are used by most interfaces and are a good alternative for the traditional WIMP style user interfaces.

- *Myth #3: "Multimodal input involves simultaneous signals".* Users act multimodally with the real world by using different modalities simultaneously, which does not require to perform individual multimodal inputs at the same moment.

- *Myth #4: "Speech is the primary input mode in any multimodal system that includes it".* Speech is a dominant media because of the fact that it is mainly used by humans as an input channel for communication. Nevertheless, it does not mean that it should be used as a primary modality in each multimodal interface. In certain cases like noisy environments or privacy concerns, speech may not be the best option, and must not be a requirement while designing a multimodal interface.

- *Myth #5: "Multimodal language does not differ linguistically from unimodal language".* One of the advantages of multimodal interaction is that it allows tasks to be simplified by using multiple input modes by taking away all the complexities provided by unimodal languages. Regarding the case of the *"Put That There"* system [54], using only speech would result in a much more complex task, while users are likely to avoid linguistic complexities.

- *Myth #6: "Multimodal integration involves redundancy of content between modes".* Multimodal integration help to support a better user experience because redundant input modes can be used to reinforce each other. Nevertheless, multimodal integration should take in consideration the complementary nature of multimodal input for a better usability. Non-duplicated information acquired by the multiple communication modes should be put together on a complementary way rather than redundantly.

- *Myth #7: "Individual error-prone recognition technologies combine multimodally to produce even greater unreliability".* Multimodal inputs requires the use of recognition technologies that are designed to understand the input received from the users. Unfortunately, such recognition systems are error-prone, but can be overcome by combining the

input of multiple recognition-based together to improve the accuracy, and providing a more reliable user interface. The users tend also to use the more effective mode instead of the more error-prone input media (?), because they naturally know when and how to choose the most effective one.

- *Myth #8: "All users' multimodal commands are integrated in a uniform way".* Users can all have their own way of using a multimodal interface due to their individual differences and interaction strategies. Therefore, the multimodal integration scheme should be flexible in a way that it should be able to detect these differences and adapt the system to the users' preferences.

- *Myth #9: "Different input modes are capable of transmitting comparable content".* From a usability perspective, it is important to understand that not all input modalities are equal. Input modalities have all their strengths and weaknesses regarding the information the user wants to convey. Since every mode is unique, it becomes important to understand the input modalities and in to know for what tasks they are ideal for.

- *Myth #10: "Enhanced efficiency is the main advantage of multimodal systems".* Multimodal systems provides much more other substantial advantages than efficiency and speed, but are flexible and allows the users to interact with the systems the way they want. A well designed multimodal interface provides a level of generality that enables the users tot support a variety of different tasks, applications and environments.

Reeves et al. [65] stated that new interaction paradigms and guidelines were needed in order to facilitate the design of multimodal systems to provide a more transparent, flexible and efficient way of human-computer interaction. The two main goals of these guidelines are to achieve a kind of interaction closer to the way human communicates with each other, and to increase the robustness of such interfaces by using redundant or complementary information. The following six guidelines should be taken in consideration for the design of multimodal user interfaces [65, 50]:

- *Requirement Specification.* Designers should be aware of the users' psychological characteristics, experience level, cultural background, physical attributes, etc. to design systems for the broadest range of users and context of use. The best modality or combination of modalities should therefore be supported to anticipate the changing environments

(e.g.: private office vs. driving car). Multimodal systems should also address privacy and security issues. In situations where speech is used as an input mode, providing a non-speech alternative in a public context could prevent users to divulge sensitive information or intimate information.

- *Designing Multimodal Input and Output.* Designers should take in consideration the foundation of cognitive science (on intersensory perception and intermodal coordination) to maximize human cognitive and physical abilities. Therefore, intuitive interaction needs to be supported based on the capabilities of humans to process information. Modalities should be compatible with the user preferences, context, and system functionality. Additional modalities should only be added if they improve the performances for a given user and context, for example allowing gestures instead of speech in noisy environments.

- *Adaptivity.* Designers should take in consideration the needs and abilities of different users, as well as the different context in which the multimodal interface will be used. The individual differences of the user (for example, age, preferences, skill impairment) can be captured and used to determine the settings of the interface.

- *Consistency.* The multimodal interface should remain consistent in terms of presentation and prompts even if different input and output modalities are used to perform the same task. Therefore the output of the system must be independent from the used input modality.

- *Feedback.* The users should be aware, by the means of feedback, of which modalities are made available to them without being overloaded with lengthy instructions that could distract them from performing tasks correctly. Therefore, it is important to provide intuitive feedback messages to the users by, for example, using descriptive icons, and notify them whenever a certain action needs to take place (for example, speech).

- *Error Prevention/Handling.* The multimodal interface should be clear and easily discoverable to provide a good error prevention and error handling. Other possibilities to avoid errors are to provide enough assistance to the user by the means of help functions, allowing the users to undo a previous action or command, or even integrate complementary modalities to allow users to select the one they feel more comfortable with.

2.2.3 Principles of Multimodal Interaction

When it comes to multimodal human-computer interaction, it is important to understand the basic knowledge regarding the input and output modalities implied in the interaction between the *interfaces* or *machines* and *humans*.

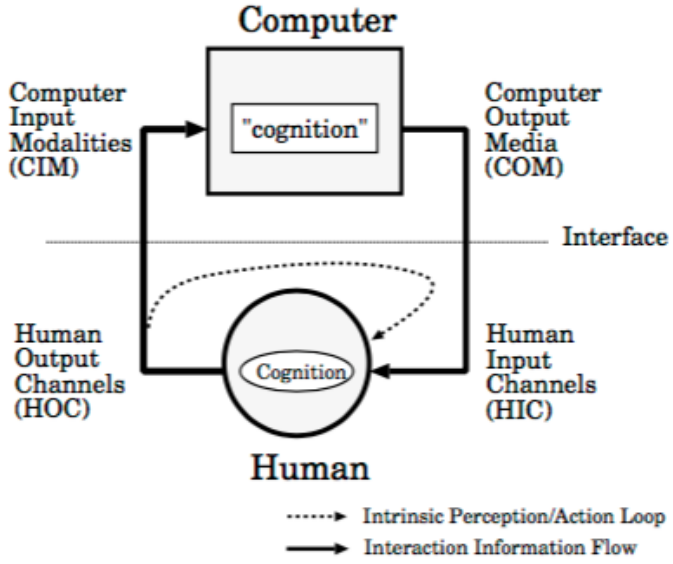


Figure 2.14: Model for the identification of basic processes in human-computer interaction [66].

According to Schomaker et al. [66] there are at least two physically separated *agents* involved in the multimodal human-machine interaction processes, one human and one machine, but they are able to exchange information through a number of information channels. As depicted in the scheme shown in Figure 2.14, the model describes the four input and output channels, as well as the involved agents. The *Human Input Channels (HIC)* and *Computer Input Modalities (CIM)* describes the input, while *Computer Output Media (COM)* and *Human Output Channels (HOC)* define the output channels or feedback [64]. The model involves two basic processes:

- The *Perceptive process* connects the human input (HIC) and the computer output (COM) together.
- The *Control process* connects the human output (HOC) and the computer input (CIM) together.

Each of the involved agents possess a cognitive or computation component that processes information acquired from the input channels in order to

prepare the output. These intermediate cognitive levels are influenced by intentional parameters, either implicitly by design, in the case of non-intelligent agents, or explicitly, as in the case of humans or more sophisticated agents [66]. Since the elaboration of a model describing the complexity of the human cognitive apparatus is difficult even hindering, the design of interfaces can only be inferred, since it can not be observed directly, and be supported by empirical results [64].

Dumas et al. [15] propose an interaction model inspired by the Norman's action cycle [67], which suits more for multimodal human-machine interaction. The model, as shown in Figure 2.15, identifies important concepts that should be considered for the design of multimodal systems, such as: the *fusion* of multiple input modalities, and *fission* to generate a message to the user depending on the context. In the human-machine interaction loop (see Figure 2.15) the first state, which is the *decision* state, consists of the human forming consciously, or unconsciously, an intention in order to interact with a machine. In order to satisfy the goal, the human selects the communication means in the *action* state to transmit the message. In the *perception* state, the machine makes use of modules, which can be in the form of sensors, to capture the transmitted message. Depending on the different information collected in the perception state, the machine's *interpretation* state goal is to give meaning to these, typically done by the fusion of multimodal messages. Action is then taken in the *computational* state depending on the developer's defined rules. The machine's *action* state will generate and transmit an answer to the user according to the meaning extracted from the interpretation stage. Depending on the user's profile and context of use, the *fission* engine will choose the adequate output modalities for the transmission of the message. In the human's *perception* state, the user is able to perceive the changes in the world performed by the machine's action state, allowing the interpretation of the generated message in the *interpretation* state.

The architectural details of what a multimodal system should be composed of is illustrated in Figure 2.16. This figure reflects the software components used by the machine in order to make the interaction model described before work. *Recognizers* are used to perceive the input modalities, which outputs will be used by the fusion engine. The *fusion engine* is responsible for interpreting the inputs and to forward it to the *dialog manager* responsible for the identification of dialog state, the transmission of a message through the fission component, the transition to perform, and/or the communication of the action to the application. The *fission engine* return the message to the user by choosing the most adequate output modality, or even combination

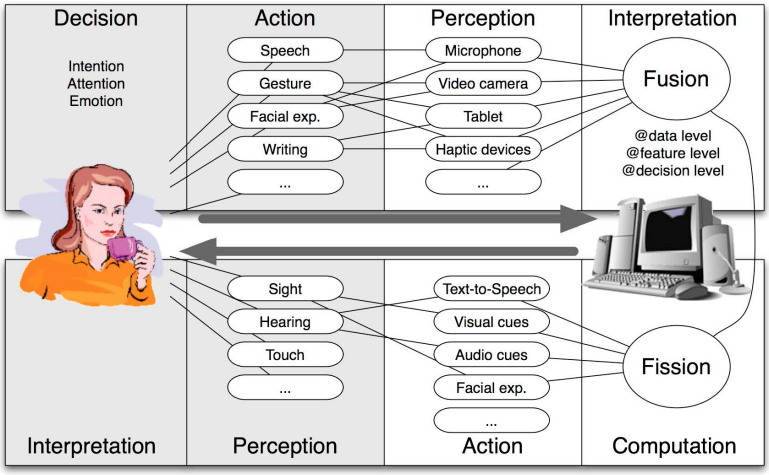


Figure 2.15: Dumas' proposed multimodal human-machine interaction model [15].

of output modalities, depending on the context and user's profile tracked by the *context manager*.

2.2.4 Multimodal Fusion

Fusion engines have received a lot of attention due to the benefits they provide regarding various multimedia tasks [68]. By processing multimodal data provided by a set of input modalities, such engines are able to obtain valuable insights. Unfortunately, the fusion of multiple modalities is a complex task, since it depends on the different characteristics of the involved modalities, such as: the different frame rates of audio and video modalities, the processing time required by the used modalities, the correlation of modalities, and so forth [15, 50, 68].

Figure 2.17 represents a classification of the types of multimodal interfaces according to Nigay and Coutaz (1992) [69]. A multimodal interface can be categorised along a three dimensional space, defining the use of modalities, fusion, and levels of abstraction. The *Use of Modalities* dimension defines whether the modalities from a specific multimodal interface can be used in *parallel*, or *sequentially*, by the users. The *Fusion* dimension defines the combination of different modalities. If a multimodal interface uses a fusion engine for the combination of different types of data, the attributed value will be *combined*, otherwise, *independent* in the case of the absence of fusion. The last dimension, *Levels of Abstraction*, shows that all the multimodal interfaces

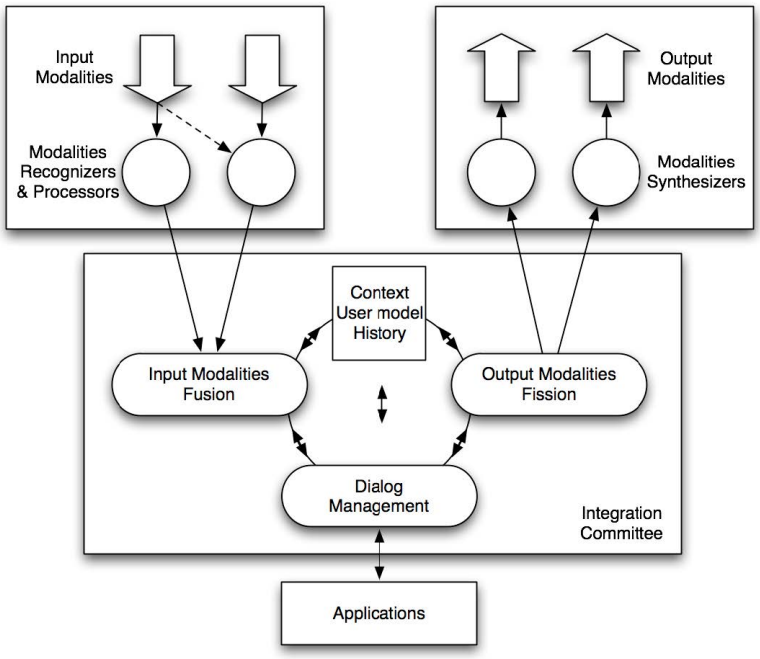


Figure 2.16: The architecture of a multimodal system [15].

have as value *meaning*, making it able to categorise the multimodal interfaces in to four CASE categories [50, 70, 71]:

- *Alternate*: when modalities are used sequentially by the users **with** fusion.
- *Exclusive*: when modalities are used sequentially by the users **without** fusion.
- *Synergistic*: when modalities are used in parallel by the users **without** fusion.
- *Concurrent*: when modalities are used in parallel by the users **with** fusion.

As depicted in Figure 2.18, the fusion of different input modalities can operate at three levels: data level, feature level, or decision level [15, 71, 72]. Fusion executed at the *data level* operate directly on the input streams provided by the input modalities [71]. Such kind of fusion is used when the fusion engine deals with multiple signals coming from similar input modalities (e.g.: two webcams), providing no loss of information but a high susceptibility of noise and failure due to the absence of pre-processing [15]. *Feature Level*

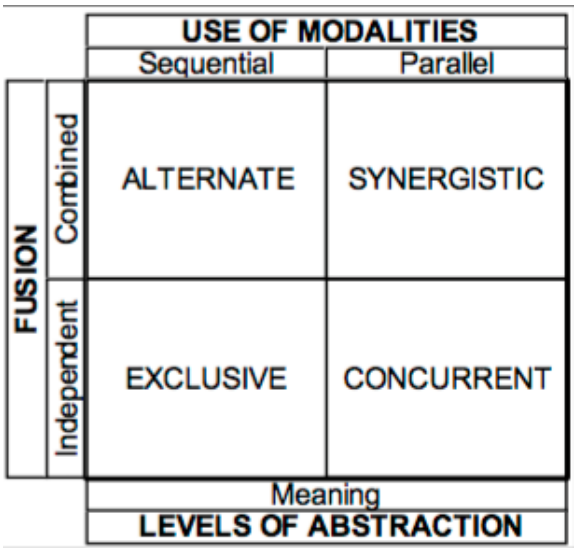


Figure 2.17: The classification of multimodal interfaces types [69]. Figure taken from [70].

fusion extracts patterns and characteristics, using adaptive systems, when tightly-coupled, or time synchronised, input modalities need to be fused. Even though it handles better noise than the previous type of fusion, it nevertheless can loose information [15, 71]. Fusion at the *Decision Level* is mainly used for the fusion of loosely-coupled input modalities. Since the data is pre-process data, it provides a low loss of information and failure sensitivity [15].

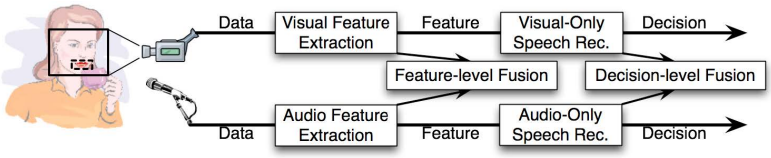


Figure 2.18: The three levels of multimodal fusion [15].

2.2.5 Multimodal Fission

According to Foster (2002) [73] multimodal fission is *"the process of realising an abstract message through output on some combination of the available channels"*. The generated message should be adequate to the context provided by the available output modalities, making it a delicate task [15]. Multimodal fission consists of three main tasks [15, 73]:

- Message Constructing or *"Content selection and structuring"*: consists of the task used to design the overall structure of a presentation.
- Output Channel Selection or *"Modality Selection"*: consists of the task responsible for the selection of output modalities depending on the context, the user's profile, and the characteristics of the output modalities.
- *"Output Coordination"*: consists of ensuring the coherence and synchronisation of a presentation when multiple output modalities are combined.

2.2.6 The CARE Framework

While the CASE model focuses more on the combination possibilities of modalities at the fusion engine level, the CARE properties, **C**omplementarity - **A**ssignment - **R**edundancy - **E**quivalence, instead focuses more on the user level [15, 74]. The CARE framework provides a way of characterising and assessing multimodal interaction from both the user and the system perspectives [75]. *Equivalence (s, M, s')* states that modalities of a particular set of modalities M can all lead to the desired state *s'* from state s [74, 76]. In other words, since all the modalities are equivalent, choosing one of them at a time would have the same result [15, 74, 75]. For example, asking the departure of a flight by clicking on a button or by using speech [75]. *Redundancy (s, M, s', tw)* states that reaching the desired state *s'* from state s can be achieved by redundantly using modalities (from the set M) within the same temporal window *tw* [74, 76]. Since all the modalities have the same expressive power [76], they can be used simultaneously to reach the same goal. For example, using speech and direct manipulation redundantly to ask for flights from a particular city [75]. *Complementarity (s, M, s', tw)* states that modalities from a set M are to be used in complementary within the same temporal window *tw* in order to achieve the desired goal *s'* [74, 76]. In the case of Bolt's "Put That There" system [54], gestures and speech need to be used in order to attain the target state. Finally, *Assignment (s, m, s')* states that only one modality m can be used to achieve the desired state *s'* from state s, due to the absence of choice [74, 75].

3
Related Work

Presentations are found everywhere serving as a means for persuading people or to deliver a message to an audience. Being a form of communication, it is important to understand the basic principles of the communication process in order to deliver a message efficiently. Multiple areas, such as pedagogy, psychology, education, and so forth, are involved in the study of communication providing crucial information to improve the quality of the delivered message. However, the use of display and sensor-based technologies have proven their potential in certain cases. The first part of this chapter presents the basic principles of communication in the field of Pedagogy, Psychology and Education, by identifying the most important elements involved in the communication process. More specifically, an in-depth explanation of important nonverbal communication skills is provided, since most of them are measurable by sensors.. The second part focusses more on the field of Computer Science, and presents multimodal systems measuring nonverbal cues and providing informative feedback to the speaker.

3.1 Communication

Communication is an important study area and cannot be regarded as a phenomenon which simply *"happens"*, because every activity involving participants negotiating their roles in the communication process do use a form of

communication, which can be direct (consciously) or indirect (unconsciously) [77, 78]. Moreover, communication skills are very important since they do not only affect personal and organisational effectiveness [77, 79, 80], but are regarded as one of the most important characteristic of a job candidate [77, 81]. Since communication is an important topic regarding this thesis, the following sections will be devoted to the explanation of the communication process, as well as the important communication skills that need to be mastered in order to provide a message efficiently and of high quality, and finally the effectiveness and importance of feedback in the communication process.

3.1.1 The Communication Process

Communication is the process of transmitting a message, which is information and common understanding, from one person to another, or even to multiple persons [77, 78]. The word communication takes its name from the Latin word *communis*, meaning *common*, and states that there is no communication unless the exchange of information results in common understanding [77]. As shown in Figure 3.1, the important actors and elements present in the communication process are identified.

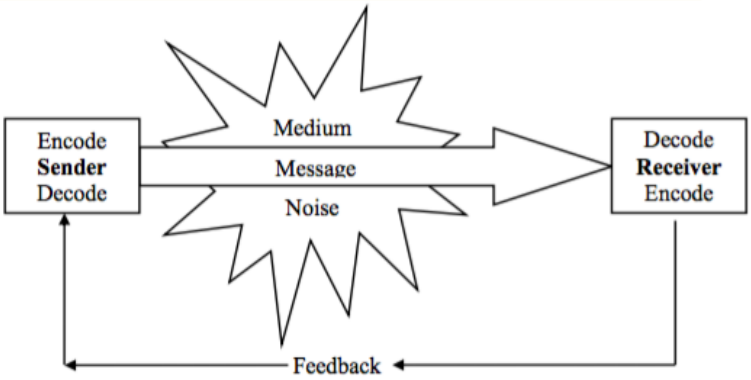


Figure 3.1: The communication process [77].

At least two common elements are to be found in every communication exchange, which are the *sender* and the *receiver(s)*. The sender is the person who initiates the communication to convey an idea or concept, while the receiver(s) is the individual to whom the message is sent. The *message* is *encoded* by the sender in the form of verbal, nonverbal, or written language by simply selecting words, symbols, or gestures [77]. It is important that the form of the message should be appropriate to the *medium* or *communication channel* (e.g. face-to-face conversation, e-mail, telephone call,

etc.), which will be used to convey the message. The receiver(s) then *decodes* the received message to understand its meaning and significance. Nevertheless, *noise* is anything that could create misunderstandings at any stage of the communication process. This is why the potential of misunderstandings should be minimised and the communication effectiveness should be maximized by choosing the appropriate communication channel, understanding the audience, and encoding the message on the best way to avoid different perceptions of the message, language barriers, emotions, interruptions and attitudes. *Feedback* from the receiver(s), through verbal and nonverbal reactions, allows to determine whether the sender's message has been received and understood correctly in order to be able to correct any misunderstandings as soon as possible [77, 78].

Many reasons could be the cause of communication failure. When noise (see Figure 3.1) exists in one of the multiple elements of the communication process in any way, the message may not be clear enough and understandable as the sender intended. Understanding and overcoming these types of noises or *barriers* lead to a clear and concise message. It is of great importance that the communicator must be aware of the following barriers and try to reduce their impact in the communication process [77, 78, 82]:

- *Language or Semantic Barriers*: Even if the communication is performed in the same language, the semantic or the meaning of the terminology (words) used may act as a communication barrier. The same words may have different meanings depending on the audience and the use of expressions may be misinterpreted or even offensive.
- *Psychological Barriers*: The psychological state of the communicator may have an impact on the message in the communication process. A stressed, angered, or communicator with low self-esteem will influence the way the message is sent, received and perceived by the receivers.
- *Physical Barriers*: Physical distractions, such as telephone calls, drop-in visitors, the geographical distance between the communicator and the receiver(s), may have an impact on the effectiveness of communication.

3.1.2 Verbal Communication

Verbal communication is the sharing of information with others using speech [83, 84]. Usually, verbal communication happens through face-to-face conversations, but meetings, conferences, phone calls, and written communication

	Verbal Communication	**Nonverbal Communication**
Oral	Spoken Language	Laughing, Crying, Coughing, etc.
Non Oral	Written Language/Sign Language	Gesture, Body Language, etc.

Table 3.1: Comparison between verbal and nonverbal communication regarding spoken communication [84].

can also be regarded as being other forms of verbal communication understandingNonverbalComm. Many people tend to misinterpret the definition of verbal communication. Moreover, they assume verbal communication to only refer to spoken communication, which is not the case. Table 3.1 highlights the different kinds of communications and categories. Even if laughing tends to belong to the verbal communication category, it is not considered to be a word and is therefore classified as a form of nonverbal communication [84]. According to [84], verbal communication is *"an agreed-upon and rule-governed system of symbols used to share meaning"*, where symbols are used for the encoding and decoding of arbitrary representations of ideas, emotions, thoughts, or actions. Since verbal communication is *rule-governed*, agreed-upon rules must be followed to give sense to the shared symbols.

3.1.3 Nonverbal Communication

Communication is not so much in *what* is said, but more in *how* it is said [85]. Besides the *explicit* meaning of words, information, or message, communication includes also *implicit* messages expressed through *nonverbal* behaviours. Such nonverbal cues, intentional or not, contribute to the verbal communication process and interpretation of information by any behavioural or expressive communication channel, such as gestures, eye contact, facial expressions, body posture, and also the physical distance between the sender of the message and the receiver(s) [83, 86, 87]. According to Argyle [88], nonverbal behaviour possess five primary functions:

- *"Expression of emotions mainly through the face, body, voice."*
- *"Communication of interpersonal attitudes."*
- *"Accompany and support speech."*

- *"Self-presentation."*
- *"Rituals."*

Nonverbal communication is perceived by the audience as being part of the message. Unfortunately, the interpretation of body language and nonverbal cues is not an easy task, since nonverbal communication is not a language with a fixed meaning and is influenced by the context (e.g.: culture, place, people, and etc.) in which the communication process occurs [86]. Nonverbal communication has proven to be significant, especially in conveying messages and forming judgements about people [89]. Even when mixed signals force the receiver to make a choice between verbal and nonverbal parts of the message, the receiver(s) tend to choose, most often, the nonverbal aspects [89, 90]. Many forms of nonverbal communication exists. Developing an awareness about these forms of communication can be helpful in improving communication abilities and increasing the effectiveness of a message, by showing that the speaker is self-confident, capable, and controlling the situation [89]. Nonverbal communication includes [91, 86, 92, 93, 89, 94]:

Gestures and Postures *"are the frequent, continuous movement changes that happen in the body while speaking, walking, even sleeping"* [95]. While gestures are vital expressions (actions) of *parts* of the body, and postures the activity of the whole human body at once [95], both of them reflect individual thoughts and regulate communication [94]. Gestures can be divided in *speech independent* and *speech related* gestures [93]. The first one defines gestures that can be understood independently from speech, since they possess a well-known verbal translation (e.g.: gestures used to represent "okay", "peace", etc.). While the second one accompanies speech and is used to enhance verbal communication [93]. Using gestures to emphasise spoken words can ensure an animated and lively conversation, otherwise, the conversation can be perceived as boring or tense. It is also preferable to not exaggerate and to use appropriate gestures depending on the context (e.g.: being aware of the meaning of gestures with respect to the cultural backgrounds of the listeners) [83]. Postures therefore reflect the emotional states, attitudes, and intentions of the speaker [92, 93]. It is important to sit or stand with an erect posture when communicating with others, to lean towards the listener(s), and to not turn your back when speaking [83].

Eye Contact or Eye Gaze refers to the eye movements and can indicate the degree of interest, engagement or involvement of the listener(s) and speaker(s). On the other side, a lack of eye contact may suggest nervousness, detachment, fear, or that a person is hiding something [92, 89]. Eye

contact occurs 10 to 30% of the time during a conversation [94] and can help to regulate the flow of the conversation. Differences in cultural backgrounds may lead to breakdowns in conversations [94]. Therefore it is vital to use the appropriate pattern of eye contact, regarding both culture and context [89].

Argyle [88] stated that ***facial expressions*** are the most important non-verbal communication channel for expressing emotional states or attitudes. Researchers agreed to categorise facial expression in *anger, happiness, surprise, fear, sadness, anger* and *disgust*. Interpreting facial expressions should be done in a careful manner, for example, a frown could be the effect of a headache rather than from the difficulty of a task [89]. Pleasant facial expressions are powerful cues for transmitting messages including friendliness, happiness, and so forth. Smiling can, for example, let others perceive the speaker as approachable, friendly and allow the message to be remembered by the receiver(s), while a frown or grimace will send a negative message [83].

The voice of the speaker serves as a communication channel, or medium, during conversations or presentations. The audience is not only listening to the words the speaker is saying, but rather on *how* it is said [93]. Mastering the ***vocal elements*** or ***paralinguistic***, such as speech rate, volume, tone, and voice inflections, will grab the attention of the audience by adding interest and meaning to the shared messages [89, 94, 83]. It is also important that the paralanguage aligns with the accompanied verbal message [94]. Not everyone possess the same voice as a broadcast reporter, nevertheless tactics can be employed to use vocal techniques effectively [83, 85, 96]:

- **Speech Rate** is an important aspect of verbal communication and determines the number of spoken words or syllables per time unit. [91]. Speaking consistently at the same rate can lead to ineffective communications comparing to a varying speech rate, which will help to maintain the interest of the audience. Speaking quickly in combination with an enthusiastic tone can be used to excite the audience. In the case where information needs to be absorbed, a low speaking rate is more preferable. However, a good speech rate for the English language lays between 100 to 150 words per minute [85, 97].

- The voice **pitch** defines the frequency of speech. A more authoritative and influential character is given to the voice when the pitch is lowered, while raising the pitch would suggest a question or uncertainty. Even if both pitches are useful, it is important to not take either to an extreme or to shift the pitch of the voice in an uncontrolled manner.

- The voice **volume** affects the audience's ability to hear and understand the speaker. The speaker should speak loud enough in order to provide a comfortable feeling to the audience. A much louder volume may annoy and disturb the audience, while speaking too softly makes the speech hard to hear. By varying the volume of the voice occasionally, the speaker is able to add character to the speech. It is preferable to raise the volume when a particular word or idea needs to be emphasised, and to lower the voice to allow the audience to concentrate on the speech.

- **Pauses** are powerful means, since they help to break up the flow of information and to allow the listeners to process, interpret, and understand the meaning of what was said. They can be used to gain the attention of the audience before an important message or conclusion takes place. Figure 3.2 shows that a speech becomes interesting and effective when a low pitch is combined with a varied pace and occasional pauses.

- To improve the audience's understanding, the speaker should **articulate** clearly each pronounced word, phrase, and sentence. By speaking clearly, the speaker enhances the quality of the speech, and conveys a feeling of intelligence, confidence, and competence. Another important factor is to avoid unnecessary words, such as fillers (e.g. *"uhm"*), and the repetition of words.

Proxemics refers to the *personal space* of an individual, and *territory*, which refers to a large private area controlled by an individual [93, 89, 94]. Invading another person's territory may cause uncomfortable situations and create communication barriers (noise). The extent to which people react whenever their personal space is encroached, depends on multitude of factors, such as culture, sex, age, etc. [89, 94]. Another important aspect of proxemics is the ***distance*** that people tend to put between themselves or others, since it can indicate the difference in power, feelings or attitudes [89]. It is important that the person with the highest status in an interaction must generally appropriate the distance regarding the relationship with the listener(s) and to control the level of approach [89, 83].

Physical Appearance is generally the first nonverbal message acquired by the receiver(s) and plays an important role in the interpretation, credibility and understanding of the message [89, 94]. These static nonverbal cues, such

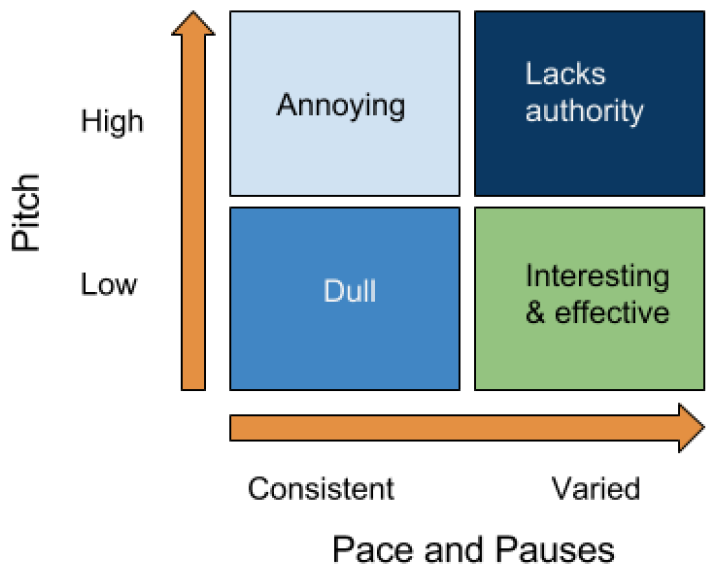


Figure 3.2: Effective use of vocal elements [83].

as attractiveness, body physique, clothes, and so forth, can develop judgements about people and impact the way people perceive others as similar to themselves. Attractiveness has shown to provide more advantages in most of the aspects of life [93, 89, 94]. Even if certain physical properties cannot be changed easily, a convenable dress could play a significant role [89].

Investigating the importance of nonverbal communication aspects for presentations can sometimes lead to misinformation, because of the multiple sources using the 7%-38%-55% rule, also known as the *Mehrabian Myth*, wrongly. Multiple research conducted by Mehrabian [98, 99] have led to the conclusion that the importance of verbal communication (spoken words) is only 7%, while 38% for para-verbal communication (voice tone), and 55% for nonverbal communication (body language). The rectification provided by Mehrabian in his website [100] states explicitly that this equation is only applicable when the communicator is talking about their feelings or attitudes, and is thus not applicable to all forms of communication.

3.1.4 The Importance of Feedback

A communication process can be in one or two ways. *One-way communication* is a communication process that does not use any feedback, and flows

only in one direction, namely from the sender to the receiver(s). It is known to be a faster way of communication, but lacks in terms of clarity because the others are not taken into consideration [101]. *Two-way communication* occurs when the notion of feedback is present in the communication process. Feedback can be regarded as *"information provided by an agent (e.g., teacher, peer, book, parent, self, experience) regarding aspects of ones performance or understanding."* [13]. The feedback message can be in the form of a verbal or nonverbal response, external or internal (like self-examination), and allows the sender to adjust their message or even to consider a different approach [101, 102]. According to Lunenburg [77], a two-way communication involving feedback is more effective and desirable compared to a one-way communication process. It is in human nature to unconsciously look for feedback messages giving sense to all actions [101]. Moreover, without feedback, no possible way of knowing whether the message has been understood correctly would exist [102]. However, feedback must be delivered properly to reinforce and motivate people to improve, rather than putting the communicator on the defence [89]. Therefore, guidelines are available to provide effective feedback [89]:

- Feedback should be specific by highlighting specific events instead of providing only general advice.
- Feedback should offer ways to resolve the occurred problems.
- Feedback should be provided in a positive way, even negative feedback.
- Feedback should be problem oriented instead of people oriented.
- Feedback should be descriptive and not evaluative.
- Feedback should be well timed.
- Feedback should not cause cognitive overload by presenting too much information the user can use.

3.1.5 Conclusion

Communication has shown its importance in multiple facets of people's daily life. It is practically used everywhere, in the business environment, at school, and so on. Nevertheless, delivering a message of quality requires to master multiple communication skills, verbal and nonverbal. Unfortunately, this is not an easy task in terms of public speaking, since there exist no "fixed" rules defining what a good presentation actually is. Therefore, it is important

that the speaker should be aware of the identified communication skills, and should understand the audience, in order to adapt their behaviours depending on the presentation context.

3.2 Multimodal Feedback Systems

3.2.1 Presentation Sensei [1]

Presentation Sensei [1] is a presentation training system that provides, in real-time, recommendations to the speaker for improving their behaviour during presentations. Therefore, to improve the quality of the delivered message, they only focus on five aspects of nonverbal presentation aspects, which are:

- The speaker should not speak too slow or too fast (speaking rate).
- The speech should not contain too much fillers (e.g.: *"euuuh"*).
- The speaker should have enough eye contact with the audience.
- The presentation should have a time limit.
- The speech should not be too monotonous.

The motivation behind selecting only these five aspects is because it is nowadays still difficult to detect all the presentation aspects, while these five exist in recent speech and image processing technologies. Nevertheless, for acquiring the best performances with the system, it is better to perform the presentation in environments such as rooms with no noise and no visual obstacles.

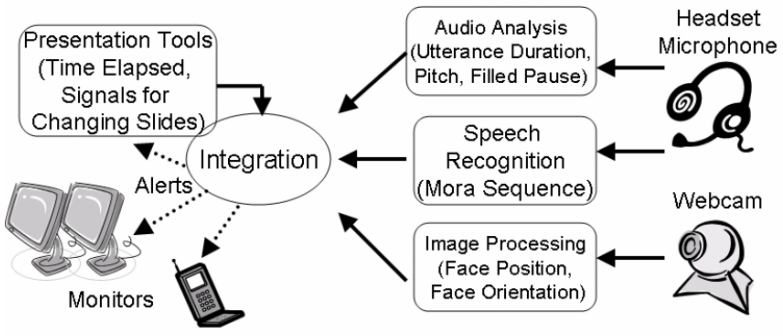


Figure 3.3: Architecture of Presentation Sensei [1].

The architecture of Presentation Sensei, as presented in Figure 3.3, consists of a "star"-like network where several modules are connected to a central module. Each module, except the integration module, is responsible for tackling one or more presentation skills aspects as explained previously. The audio analysis module is responsible for detecting the utterance, filled pause and the pitch of the speaker by the means of a microphone. It measures the voice of the speaker every 10 seconds and sends continuously this data to the integration module for processing purposes. The system can then recognise whether the speaker should add more pauses or is speaking monotonously.

The Julian speech recognition engine, integrated in the Speech Recognition module, has as purpose to count the number of moras or in other words, the number of Japanese syllables and to send this data to the integration module for determining whether the speaker should adjust its speaking rate or not. Since it only supports mora-based languages such as Japanese, supporting other languages would result in the need to use other speech recognition engines that support the speaker's language.

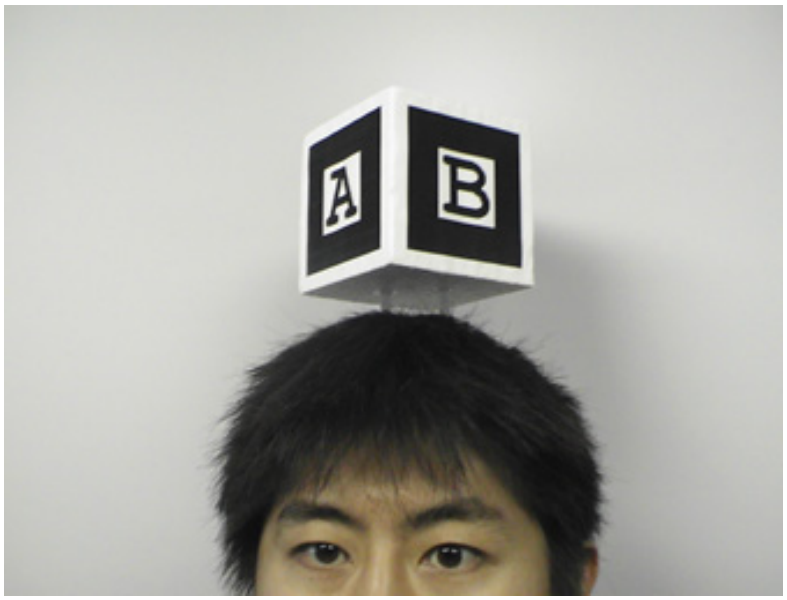
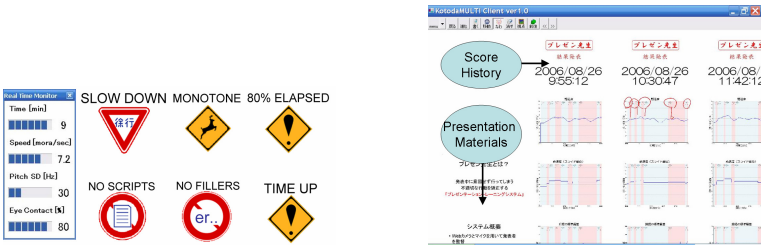


Figure 3.4: The AR toolkit for detecting and tracking the head of the speaker [1].

The Image Processing module is used to determine whether the speaker is looking to the audience or not by tracking the position and the orientation of the face. Two methods are proposed, one that makes use of the AR toolkit and a more advanced one that uses Sub-space and Support Vector Regression (SVR) method. The AR toolkit, as presented in Figure 3.4, is a

Figure 3.5: The Sub-space and SVR method for tracking the head of the speaker [1].



(a) Online feedback. The short term statistics on the left side and the icons on the right side.

(b) Offline feedback. Rehearsal history of the speaker.

Figure 3.6: Presentation Sensei feedback modes [1].

special visual marker that the speaker should wear on its head. The second method (as shown in Figure 3.5) is more advanced, since it only requires a series of images of the speaker's head for computing the eigenvectors for each orientation of the face. Therefore, the orientation of the face can be detected without making use of a marker.

Two important features are available in Presentation Sensei, which provides the speaker with the possibility of choosing between the online and offline mode. The online mode can be used for training purposes or during the presentation itself and is composed out of an user interface (as shown in Figure 3.6a) that provides short term statistics, while icons are used as "alerts". The offline mode (as shown in Figure 3.6b) shows the statistics of

the whole presentation in the form of charts, representing the rehearsal of the speaker, after the presentation, allowing the speaker to review the whole session afterwards.

3.2.2 Presentation Trainer [2]

The Presentation Trainer proposed by Schneider et al. [2] is a system that can be used for training purposes. It allows speakers to improve their nonverbal communication skills by practicing while receiving feedback from the system. Only the basic nonverbal communication aspects are covered by the Presentation Sensei, which are:

- The body postures. Using the right body postures during presentations gives the impression to the audience that the speaker is confident, open and attentive. Therefore, they propose to stand up in an upright position in front of the audience with the hands in front of the body above the hips.
- Use of gestures. Using gestures in combination with speech during the presentation can have as advantage that some contents of the delivered message can be strengthened and give a better idea to the audience. Nevertheless, the system does not identify specific gestures, but only determine whether gestures are being used or not.
- Voice volume. Having a good volume during the presentation enhance the clarity of the message and ensures the attention of the audience.
- Use of pauses and phonetic pauses. Using pauses during the presentation can only have advantages, since it gives the opportunity to the audience to refocus on the future contents of the presentations.

The user interface is designed in a way that it does not cause cognitive overload, while the speaker can get the maximum of the system to improve its skills. As presented in Figure 3.7, the user interface is in fact a “mirror” of the speaker themselves where feedback is shown in real-time. Since it can happen that several feedback events could be triggered at the same time, the Presentation Trainer will only show one feedback event at the time and wait at least six seconds before firing the next feedback event. Selecting the right feedback event can be done by assigning relevance scores to feedback events or by using the First-In-First-Out principle. The last method displays the feedback event until the mistake is corrected. The user interface does not consist only of a Graphical User Interface (GUI) on the screen, but uses also

a wristband that produces vibrations whenever a feedback event is triggered by the system.

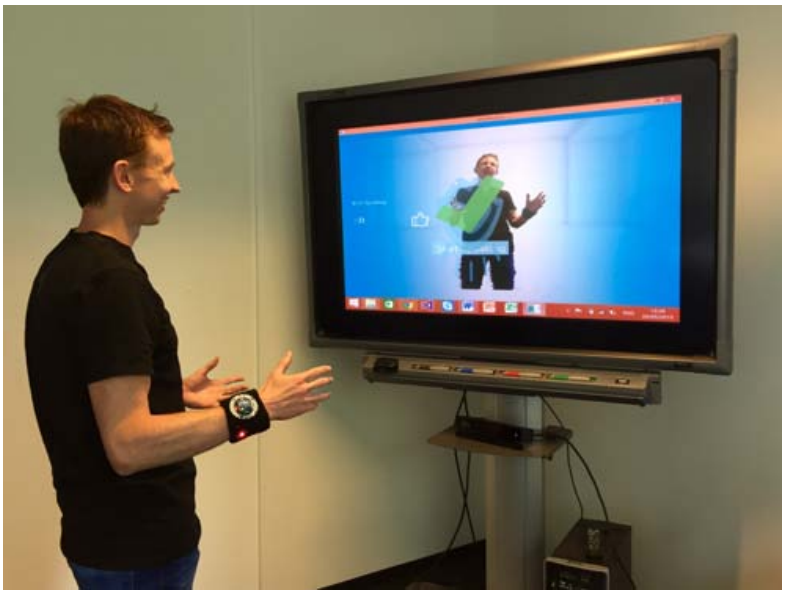


Figure 3.7: The Presentation Trainer user interface and setup [2].

Feedback events can be divided into corrective and interruptive feedback. Corrective feedback is used to alert the speaker in real-time in order to change its behaviour. As presented in Figure 3.8a, the system displays the right behaviour to the speaker to correct the identified mistake. It can happen that the same mistakes can be repeated several times or that identified mistakes can stay for too long without being corrected. Such mistakes triggers interruptive feedbacks (see Figure 3.8b) which will not only alert the speaker but will also stop the program and display the reason.

In order to be able to detect the previously-cited nonverbal communication skills, the Presentation Trainer uses the Microsoft Kinect for Windows V2 sensor with its SDK for tracking the voice and body of the speaker. The architecture of the system as presented in Figure 3.9 is really advanced and allows the inclusion of more sensors. The bottom layer or the Sensor Modalities layer is responsible for the communication between the Kinect sensor and the system. It contains not only the current coordinates of the body joints and the audio values of the speaker, but is also composed out of flags for indicating whether certain events are fired or not. All these data are used by the judgement maker for generating presentation actions, which are actually any types of nonverbal communication mistakes. The rule analyser decides based on the state of the program and the performance of the

(a) The corrective feedback event. (b) The interruptive feedback event.

Figure 3.8: The Presentation Trainer feedback modes [2].

speaker which feedback event should be triggered. The operational states receives such feedback events and forwards them to the output channels.

3.2.3 RHEMA [3]

Nowadays, technology is becoming more advanced, powerful and designed in a way that intelligence can be embedded in practically all everyday objects. A good example is Google Glass [103], which is a pair of wearable eyeglasses (as shown in Figure 3.10) running on the Android Operating System and lets the user interact in multiple ways thanks to the embedded sensors, allowing both gesture as well as voice-based input. An augmented reality experience is offered to the user by showing relevant information using audiovisual content [104].

RHEMA [3] is a smart user interface that uses the advantages of Google Glass to provide speakers with real-time feedback during presentations. Glass records the speech of the speaker and displays a feedback message, as presented in Figure 3.11, whenever it is needed. The feedback message is displayed in a way that it becomes less intrusive, minimises distraction and avoid cognitive overloads. Google Glass is compared to other wearable displays as a cheaper and light-weighted comfortable alternative, but it is nevertheless not designed for running computationally intensive applications due to heat issues. In order to avoid such problems the researchers opted for an architecture (see Figure 3.12) that makes use of a local server for processing the audio data sent by Glass and returning a result back to the glasses. The tasks of the server consist in analysing the audio data for determining the loudness and the speech rate of the speaker. The TCP (Transfer Control Protocol) is used for transferring data from the Glass to the local server since other protocols such as UDP (User Datagram Protocol) and RTP (Real-time Transport Protocol) causes unreliable packet delivery and overheating issues.

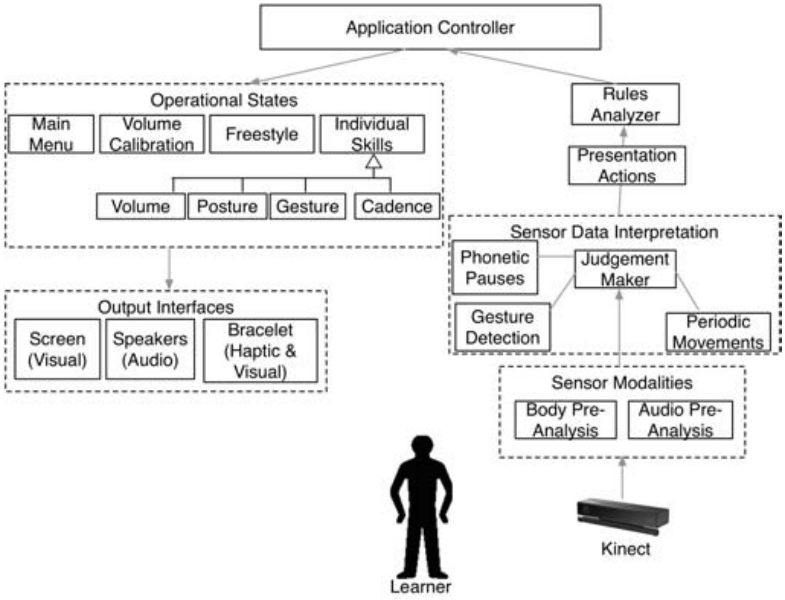


Figure 3.9: The Presentation Trainer architecture [2].

3.2.4 A Real-Time Feedback System for Presentation Skills [4]

Kopf et al. [4] provide a low-cost system that could benefit students and young professionals by giving direct real-time feedback on their behaviour while practicing presentations. The low-cost system is composed out of a GUI and the Microsoft Kinect that can easily be carried out in safe environments, which provides a comfortable sensation to the presenter, even to the more shy ones. The system takes two kinds of feedback in consideration: the first one is used to provide direct feedback on the presenter's presentation style, such as the movement, gestures and eye-contact with the audience, while the second one provides additional feedback showing statistics about the duration of each slide and the presenter's speech rate. The audio and video data during a training session is recorded so that it is made possible for the presenter to study it for finding ways to improve the speech, gestures and other presentation skills.

Figure 3.13 shows the main view (GUI) of the feedback system that visualises the captured data from the Microsoft Kinect and makes it easy for the presenter to configure the system. Functionalities are made available for playing and recording streams from the sensor as well as evaluating the current presentation for presenting feedback about the speech rate, slides duration, gestures and even the viewing of the presenter (as shown in Figure 3.14).

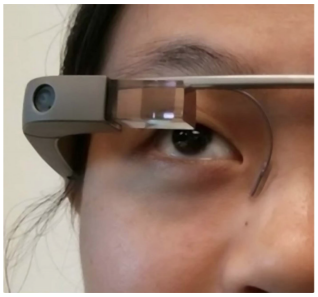


Figure 3.10: Using the Google Glass as a secondary display [3]

Figure 3.11: Feedback message displayed by Rhema on the Google Glass [3].

The gesture recognition module, which is the most important part of the system, is used to identify three essential behaviours of the presenter, which are the viewing direction of the presenter, open gestures and hands below the waistline. To determine whether the presenter is looking to the audience, the distances from the left and right shoulder to the Microsoft Kinect are computed and used with the assumption that they must practically be similar in the case that the presenter is looking at the audience. This simple approach can trigger errors in some cases (e.g.: whenever the presenter's face is oriented to the walls and not the shoulders) but is preferred to a more accurate but high computational demanding face detection algorithm.

Analysing the duration of the slide is done by using the RGB stream of the Microsoft Kinect and by measuring the amount of pixels changes. In order to avoid significant pixels while the presenter is moving, the silhouette from the presenter is computed and the occluded pixels are ignored as presented in Figure 3.15 .

The speech rate of the presenter is computed by analysing the audio stream from the Microsoft Kinect and by counting all the recognised words with the Microsoft Speech Engine.

3.2.5 Feedback System for Presenters Detects Nonverbal Expressions [5, 6]

Nonverbal expressions have shown their significance during presentations, since the use of the body and vocal elements are important for the delivery of a message. Since self-practicing presentations in front of a mirror or by videotape is time-consuming and not accurate enough, Nguyen et al. [5, 6] designed an automatic multimodal system capable of analysing nonverbal expressions in order to assist the speaker by the means of natural and infor-

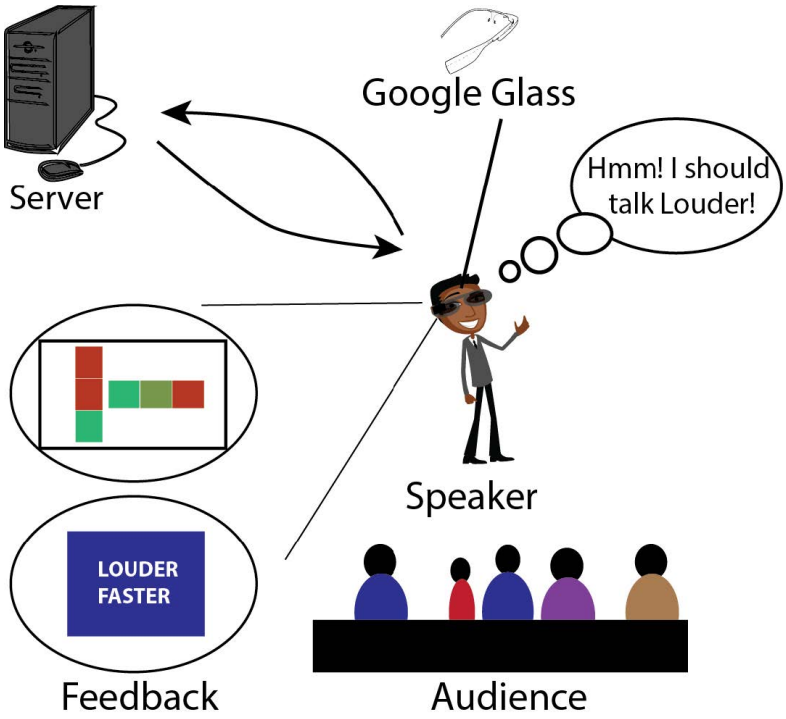


Figure 3.12: The Rhema architecture [3].

mative feedback. The architecture of their approach proposed in Figure 3.16 consists of three important modules:

- The *Capture Device* module, composed of the Microsoft Kinect, is used to capture the speaker's information in a non-intrusive manner, instead of using wearable sensors. The Microsoft Kinect captures the speaker's body movements and voice, both enough for the processing of nonverbal expressions.

- The *Emotional Signal Detection* module is the "heart" of the system. This module serves to analyse the acquired voice and body movements information acquired by the Microsoft Kinect to extract both good and bad nonverbal speaker expression.

- The *Performance Assessment* module is finally used to provide appropriate feedback to the speaker based on the detected expressions.

The Emotional Signal Detection module identifies only two nonverbal communication aspects, which are *posture* and *gesture* analysis. In order to make the distinction between good or bad nonverbal expressions, a database

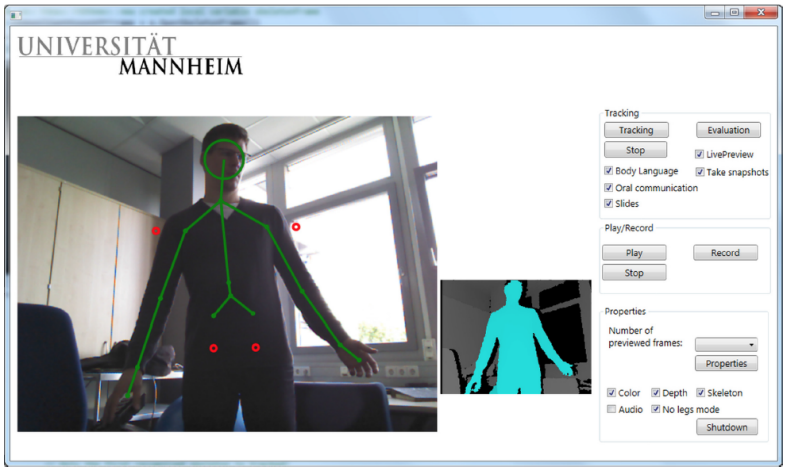


Figure 3.13: The GUI of the feedback system [4].

collecting short recorded presentations is used. These annotated samples, composed of a colour and depth image (as shown in Figure 3.17), have been recorded to serve as a reference point for the analysis of nonverbal expressions. The analysis of gestures and postures is performed separately by the system. The nearest-neighbour algorithm has been used to classify the good and bad postures using the information available in the database. In the case of gesture analysis, they implemented a method for the interpretation of gestures based on the Laban Movement Analysis (LMA) language [105]. The speaker's gestures are automatically mapped to Laban's parameters serving to detect the good and bad gestures. The detected speaker's postures and gestures are assessed through a scale going from "Bad" to "Excellent" by the Performance Assessment module, in order to provide a simple visual feedback message shown on the computer screen.

3.2.6 Multi-sensor Self-Quantification of Presentations [7]

Gan et al. [7] proposed a system that makes its distinction, compared to other multimodal systems, in terms of the type of sensor technology they use. The Multi-sensor Self-Quantification Framework, used for presentations, works only with wearable sensors (audio, video, and motion sensors), or in combination with existing ambient sensors (audio, video, and depth sensors), which makes it, according to the authors, the first wearable sensor system used to quantify the speaker's performances during a presentation.

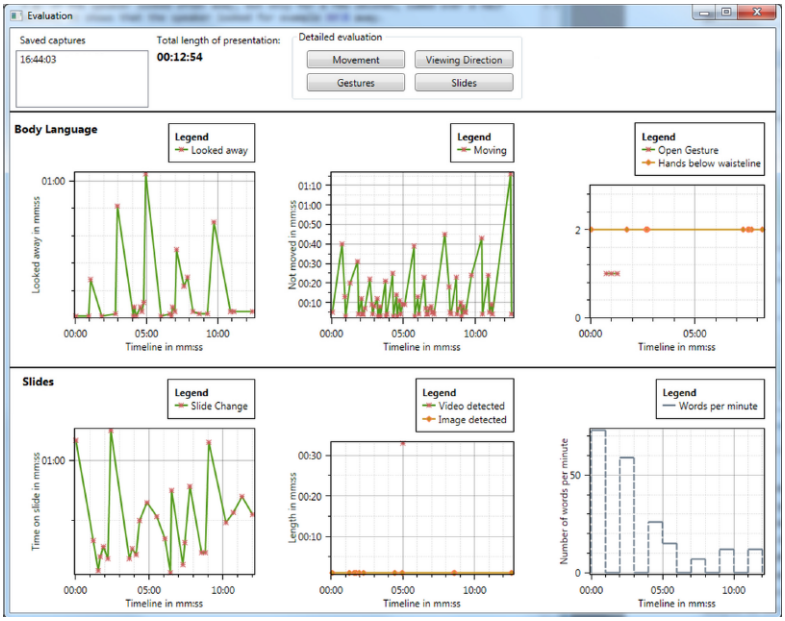


Figure 3.14: Statistics and feedback about the presenter's behavior from the recorded session [4].

The methodology used to quantify the speaker's performance, is similar to the way presentation skills are evaluated in the field of psychology, namely, by a *rubric*. A rubric, where every presentation skill is associated with different levels of performance quality using a *score*, can be used by an evaluator (human or machine) to provide informative feedback. Since providing the speaker with a score instead of a semantically useful messages such as, for example, "speak louder", is not efficient enough, the authors proposed their own assessment rubric, which contain semantically meaningful assessments while also benefiting the analytic-algorithms for the performance analysis. The proposed rubric, shown in Figure 3.18, consists of three hierarchical layers:

- The *category* layer is a high-level categorisation of presentation skill aspects.
- The *concept* layer divides each category into more detailed presentation aspects.
- The *state* layer contains the semantically meaningful state of each concept (for example, in terms of speech rate, the states would be: *slower*, *faster*, and so forth).

Figure 3.15: The silhouette of the presenter [4].

The **Vocal Behaviour** category identifies three *paralinguistic* behaviours, or concepts, which are *speaking rate*, *liveliness*, and *fluency*, in order to assess the vocal quality of the speaker. In this case, liveliness is defined as being the intonation, rhythm and loudness of the speaker's voice, while fluency refers to the speaker's speech smoothness in which syllables, words, and phrases are combined together. The raw audio data acquired by the sensors is processed by PRAAT [106], a software library for analysing speech, which is used by the system to quantify vocal behaviour.

Body language, such as gestures, facial expressions and body postures, are often used by the speakers to strengthen a message, while also making the presentation more dynamic. The **Body language** category identifies only *body movements* and *gestures* as concepts. The detection of facial expressions was not an option, since algorithms used to detect facial expressions are not accurate enough. Similar to the vocal category, a body language category contains three states for each concept used for the quantification process, which are: *excessive*, *normal* and *insufficient*. Therefore the skeleton tracking provided by the Kinect sensor is used.

The **Engagement** category is used to evaluate the level of attention of both the speaker and the audience. For the *speaker's attention*, the *screen*, *audience*, *script* and *"others"* states were identified, listing the objects in

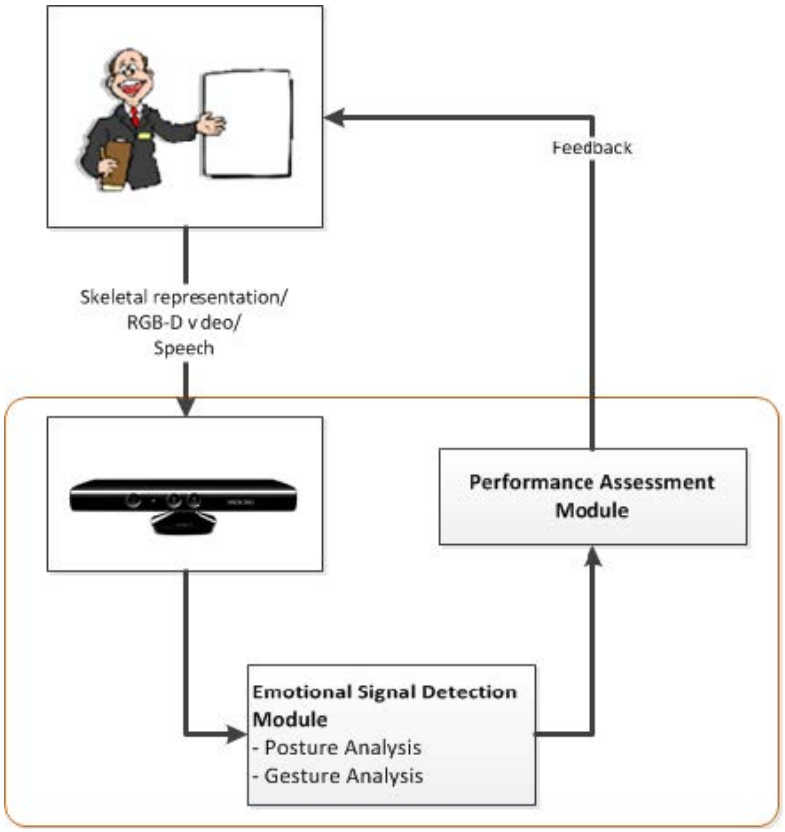


Figure 3.16: Architecture of the system [6].

which the speaker can pay attention during the presentation. On the other side, the *audience's attention* concept defines the engagement of the audience by the means of three states: *no attention*, *attention without feedback* and *attention with feedback*. The last state can be in the form of interaction between the speaker and the audience, or just by nodding the head. Therefore, three Google Glasses where used, one for the speaker and the others for the audience.

The **Presentation State** category, with *presentation state* as concept, characterises whether the presentation is in *presentation* state, or in *question answering* (QA) state.

In order to be able to quantify the speaker's performances identified by the categories and concepts, the framework uses multiple sensors in order to capture the speaker's and the audience's behaviour. As shown in Figure 3.19, the framework assumes that the data of each included sensors can be mod-

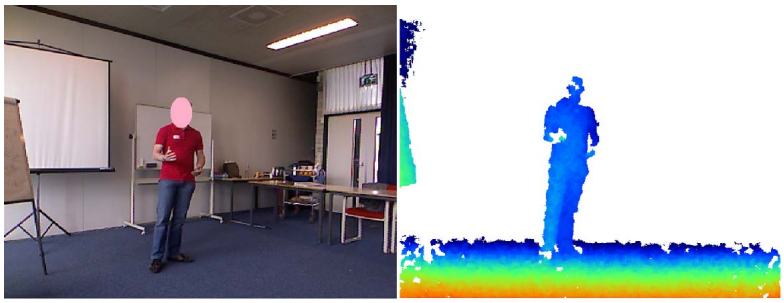


Figure 3.17: Samples stored in the database. Left, the colour frame; and right, the depth frame from the Microsoft Kinect [6].

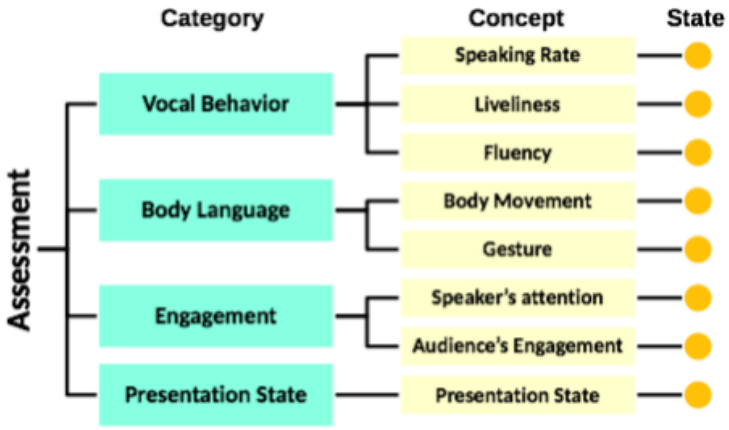


Figure 3.18: Assessment rubric used to quantify the speaker's performances [7].

elled as a collection of features, making it able to extract specific features from specific modalities m. This would help the classifier to learn, in order to identify the state of the corresponding concept, used to quantify the identified performances. The *Acoustic Feature*, *Visual Feature*, *Depth Feature*, and *Motion Feature* are all Machine Learning algorithms used to target specific performance categories.

Nevertheless, even if this framework does not provide feedback messages to the speaker in order to adapt its behaviour in real-time, it provides a generated analytics report for each presentation session. The report consists of three parts:

- As shown in Figure 3.20, the speaker is able to visualise the captured footage of all the included video devices. The moment of the presen-

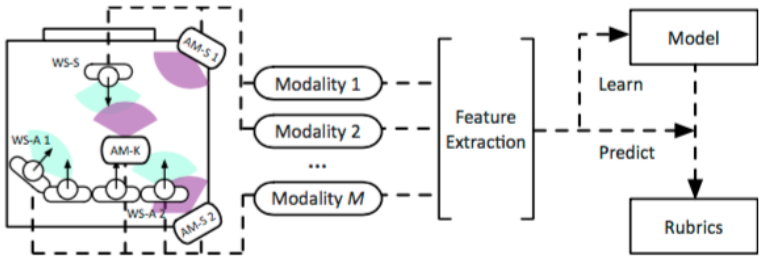


Figure 3.19: The framework with the sensor configuration [7].

tation, highlighted by the red line in Figure 3.21, corresponds to the frames shown in the video footage.

- The time-series analytics shown in the left part of Figure 3.21, consists of the identified presentation concepts. Notice that the level of the line charts have three distinct states, low, middle or high. The red marks present in the yellow graphs consists of visualising where the speaker pays attention to.
- The piecharts on the right side of Figure 3.21, allows the speaker to quickly examine the accumulated status from the identified concepts.

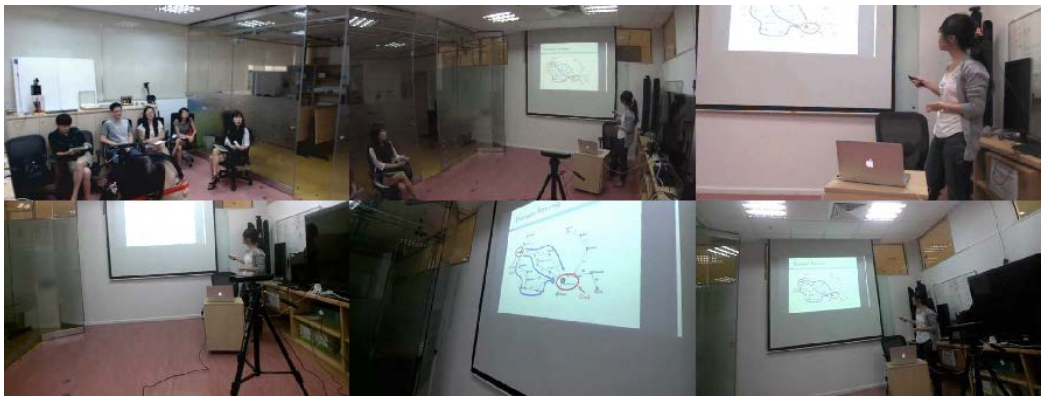


Figure 3.20: Snapshots captured from the video devices [7].

3.2.7 Multimodal Public Speaking Performance Assessment [8]

Wörtwein et al. [8] propose a multimodal framework combined with an interactive virtual audience used to automatically assess the speaker's nonverbal

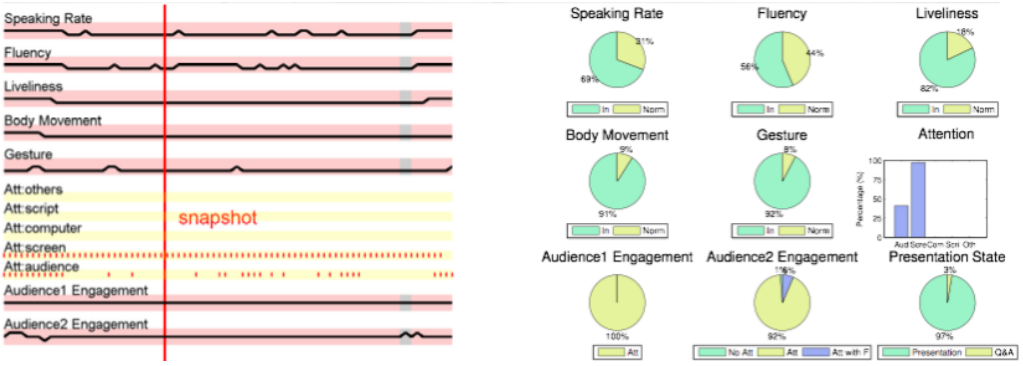


Figure 3.21: The generated analytics [7].

behaviours. The artificial virtual audience is based on the Cicero platform [107], which is capable of providing feedback according to the perceived multimodal speaker's behaviour. As shown in Figure 3.22, the virtual figures were configured in order to provide feedback to the speaker according to their performance. The provided feedback messages, given in the form of multiple nonverbal messages, are:

- Leaning forward and being attentive.
- Standing straight on their chairs.
- Leaning backwards.
- Nodding or shaking their heads.

Depending on the performance of the speaker, whenever the virtual characters' behaviour consist of leaning their body forwards or nodding regularly their head means positive feedback. On the other hand, negative feedback would result in leaning their body backwards or regularly shaking their head. Notice that the threshold values used to trigger the nonverbal feedback messages differ from one character to another. Since they are randomly sampled, every character will not behave simultaneously. The nonverbal communication aspects used to assess the speaker's performance are:

- Eye Contact
- Use of gestures
- Body posture
- Flow of speech

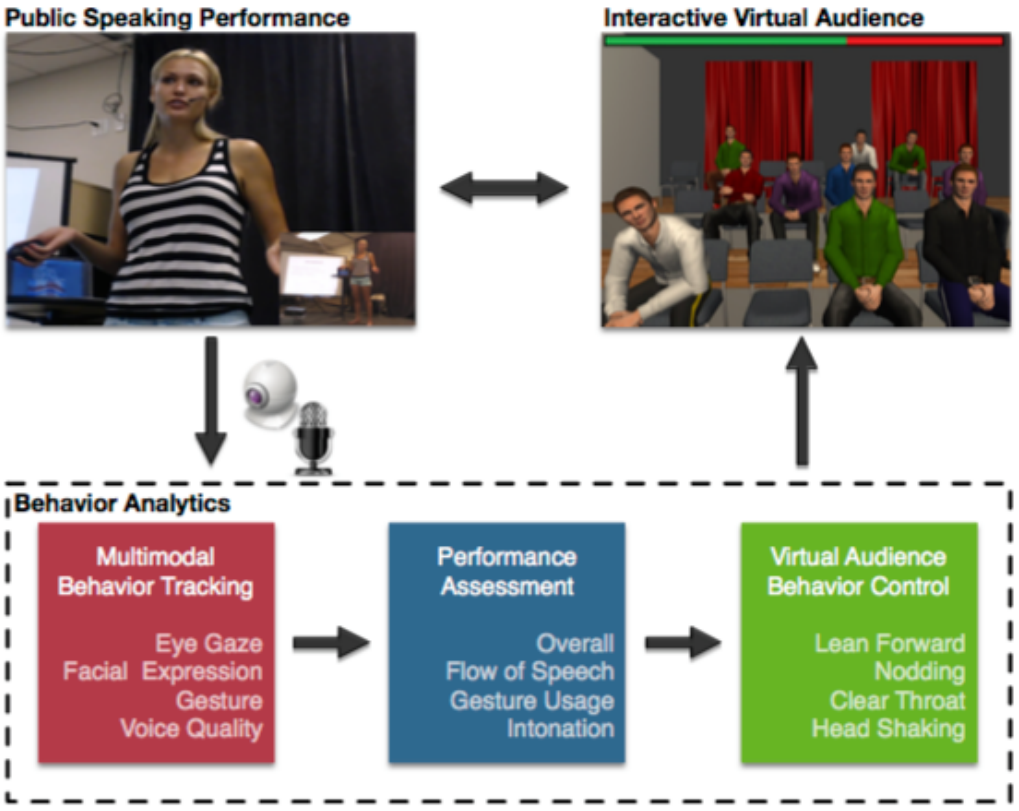


Figure 3.22: The interaction loop used to track the nonverbal behaviours allowing the virtual audience to provide audiovisual feedback [8].

- Intonation
- Use of pause fillers

In principle, the *Performance Assessment* is divided in two parts, one responsible for the assessment of the speaker's acoustic or paralinguistic behaviour, while the other for the assessment of visual behaviours. The *Acoustic Behaviour Assessment* is responsible for processing the captured audio signals with the COVAREP toolbox [108], providing robust and tested open-source tools for speech analysis. Analysing the voiced from the unvoiced parts of the audio signals, provides an estimation of the speech rate and pauses, while the intonation can be assessed by computing the speaker's voice intensity in dB. On the other side, the Microsoft Kinect is used to by the *Visual Behaviour Assessment* to obtain the skeleton data of the speaker's joints (shoulders, elbows, hands and wrists) to measure the use of gestures. The Emotient's FACET SDK [109] is used to extract the emotions of the speaker,

such as anger, sadness and contempt, while the OKAO vision-based technology [110] is used to get an estimation of the eye-contact ratio. During a presentation session, the speaker is able to get an overall assessment of their performance by the means of a bar, as shown in Figure 3.22 right above the interactive virtual audience.

3.3 Summary of the Related Work

This section presents a summary of all the related work that has been done so far regarding the previously presented unimodal and multimodal systems. First, a comparison of the previously discussed multimodal systems has been done (in Table 3.2) based on essentials criteria. Then an analysis of the table is provided where the advantages and inconveniences of the compared systems are discussed, which leads to the conclusion of the conducted research.

3.3.1 Comparison of the Feedback Systems

Table 3.2 compares the discussed unimodal and multimodal systems discussed in 3.2 based on headers/criteria that have been chosen to be very relevant for systems designed for such applications. For a better comprehension and interpretation of the comparison table, an explanation of all the criteria/headers is provided as well as their possible values in the following paragraphs.

Identified Presentation Aspects refers to the presentation skills that have been identified to be essentials and which are implemented in the system. A system is considered to have a *fixed* set of identified presentation aspects if it does not allow extensibility in anyway by the user or even by a programmer. On the other hand, the value *extensible* is used if it is indeed the case.

In terms of extensibility, the columns **Input Modalities** and **Output Modalities** are used to define whether the system supports only pre-defined input and output modalities, or allows them to be customisable. The *extensible* value is used in the case where extensibility is supported, otherwise the value *fixed*.

Context of Use stands for the situation in which the speaker can use the system. Systems designed only for training purposes are labeled with the *training*, while the value *real-time* is used in the case where the speaker is able to use the system during presentations. Nevertheless, it cannot be denied

	Extensibility					
Paper	**Identified Presentation Aspects**	**Input Modalities**	**Output Modalities**	**Context of Use**	**Data Processing**	**Thresholds**
Kurihara et al. [1]	Fixed	Fixed	Fixed	Hybrid	Manually	Fixed
Schneider et al. [2]	Fixed	Extensible	Fixed	Training	Rule Engine	Changeable
Tanveer et al. [3]	Fixed	Fixed	Fixed	Hybrid	Manually	Fixed
Kopf et al. [4]	Fixed	Fixed	Fixed	Training	Manually	Fixed
Nguyen et al. [5, 6]	Fixed	Fixed	Fixed	Hybrid	Manually	Fixed
Gan et al. [7]	Fixed	Extensible	Not Mentioned	Real-Time	Manually	Fixed
Wörtwein et al. [8]	Fixed	Fixed	Fixed	Training	Manually	Fixed

Table 3.2: Comparison table of multimodal feedback systems.

that a third option remains possible for systems suited for both training and real-time presentation purposes. Therefore the value *hybrid* is used.

Data Processing defines how the systems process raw data acquired by the input modalities used to measure the nonverbal cues of the speaker. It can either be done *manually* by using advanced algorithms, such as Machine Learning, by programming all the different situations, or by making use of a *rule engine*.

The information provided by the input modalities are mainly used to measure the nonverbal cues of the speaker. This information serves as benchmarks and are bounded between established minimum and maximum values. Exceeding these values would lead to the triggering of a feedback message. The **Thresholds** column identifies whether these established boundaries are *changeable* or *fixed*.

3.3.2 Analysis of the Comparison Table

One important thing to keep in mind is the fact that the conducted study on communication reveals that there exist no "fixed" rules defining a good presentation. Even if multiple researchers have identified important communication skills, the speaker's presentation approach must be adequate to presentation context. As shown in the comparison Table 3.2, all the presented multimodal systems identified *fixed* essential presentation aspects (nonverbal cues) beforehand. Since the **identified presentation aspects** are hard-coded in their systems, including new nonverbal cues is not an option. Moreover, these identifications are maybe based on their cultural backgrounds, making some of them useless depending on the audience and presenter, for example, the use of gesture detection for impaired speakers, or the use of eye-contact with an Asian audience. Since they all process raw data acquired by the sensors into meaningful data, in order to trigger feedback messages whenever needed, providing the possibility to use these meaningful data by plug-ins, would bring their system to another level.

Input modalities are used to capture data from the tracked speaker for the detection of nonverbal cues. Since, certain input modalities are not always affordable, it becomes important to allow the system to include different input modalities. Of course, they should be able to structure the measured information in a way that can be interpreted by the libraries and modules used for detecting nonverbal cues. Compared to Kurihara et al. [1], Tanveer et al. [3], Kopf et al. [4], Nguyen et al. [5, 6], and Wörtwein et al. [8];

Presentation Trainer [2] and Gan's quantification framework [7] are the only systems that allow the inclusion of input modalities. Even though they do not explicitly explain how this could be performed, they both stated that their architecture is indeed capable.

Output modalities are used to alert the speaker with feedback messages, by means of a display, phone, vibration (haptic feedback), and so forth. Providing a multimodal system allowing the inclusion of output modalities will open the system to a much broader spectrum of users (impaired or not). Moreover, allowing the user to choose the output modality, or modalities, can provide a comfortable feeling and not restrain the user to use specific output modalities. Unfortunately, all the proposed systems impose a predefined set of output modalities. RHEMA [3] displays feedback messages by the means of the Google Glass, Presentation Trainer [2] and Presentation Sensei [1] combine the display, in which the system is running, with haptic feedback. Finally, the systems proposed by Kopf et al. [4], and Nguyen et al. [5, 6] both use the computer display. There is no talk of output modalities with Wörtwein's [8] system and Gan's quantification system [7].

The context in which the systems can be used is really important, since some of them are designed for real-time purposes, while other ones for training purposes. The reason why Presentation Sensei [1], RHEMA [3], and Nguyen's feedback system [5, 6] are considered to be *hybrid*, is because their user interfaces are designed in a way that allows them to be used in both *real-time* and *training* situations. On the other hand, Presentation Trainer [2] and Kopf's real-time feedback system [4], provide both a user interface "mirroring" the speaker's body, which forces the speaker to concentrate on the user interface in order to read the feedback messages. This could cause unwanted distraction for the speaker, which could affect the quality of a presentation. On the other side, Gan's quantification system [7] is identified as being *real-time*, since the system is used to measure the performances of both the speaker and the audience. Finally, Wörtwein's system [8] can only be used for training purposes, since the provided feedback is given by an interactive virtual audience.

Processing meaningful data acquired by the input modalities can be a difficult task. Depending on certain presentation skills, a programmer would have to identify and use algorithms or hardcode all the different possibilities in order to provide the speaker with effective feedback. A simple solution to this problem would be to make use of rule engines, which would save a number of lines of code, provide some intelligence, ease the maintainabil-

ity, and centralise the logic of the system. Presentation Trainer [2] is the only identified system that uses a rule engine to process the data; while Presentation Sensei [1], Gan's quantification system [7], and Nguyen's feedback system [5, 6] uses Machine Learning techniques; RHEMA [3] make use of external tools like PRAAT [106]; Wörtwein's system [8] uses a combination of open source toolboxes and machine learning techniques; and finally, Kopf's feedback system [4] uses calculations to detect nonverbal cues.

Thresholds values serve mainly to define ranges in order to make a distinction between valid and non-valid detected communication skills. Allowing the user to adapt the ranges would make the system more flexible. Presentation Trainer [2] is the only multimodal system that allows the user to modify the thresholds at runtime. All the other ones make use of Machine Learning techniques to infer values, or use fixed values obtained from experimentations, or the literature.

3.3.3 Conclusion

Despite the fact that the presented systems in the comparison table have shown their effectiveness, most of them fall short in terms of extensibility. Being able to extend the functionality of a system would open much more alternatives to the users, instead of restraining them. Moreover, this would allow the users to customise the system according to their needs, making presentations more effective. Therefore, comparison Table 3.2 shows in consequence that a need for a flexible tool is evident. *PRESMA*, the proposed approach of this thesis, allows the inclusion of different input modalities, output modalities, logic, and identified presentation aspects, due to its advanced architecture, which draws its strength from the limitations of existing systems. The users are free to configure the system depending on the context of the presentation, and to their perception of what an "ideal" presentation should look like.

4
Presentation Mate

This part of the thesis presents the architecture of *PRESMA* in order to allow the inclusion of multiple input modalities, output modalities and communication skills (rules and plug-ins), providing as advantage the customisation of the system based on the speaker's need and the presentation context. The architecture is designed according to the conclusions drawn from the analysis and comparison of existing multimodal feedback systems.

4.1 Architecture

PRESMA makes its distinction over other multimodal systems in terms of extensibility. As presented in Figure 4.1, the system consists of multiple layers in order to provide a clear separation of concern and to allow certain parts of the system, such as the input modalities, output modalities, plug-ins, and rule sets to be extensible. The following sections explain the goals and ideas behind each of *PRESMA*'s layers.

4.1.1 Input Modality Layer

The *Input Modality Layer* (IML) is the layer responsible for the inclusion of input modalities. Since there exist a multitude of input modalities, it is practically impossible to define a standard way for the acquisition of data.

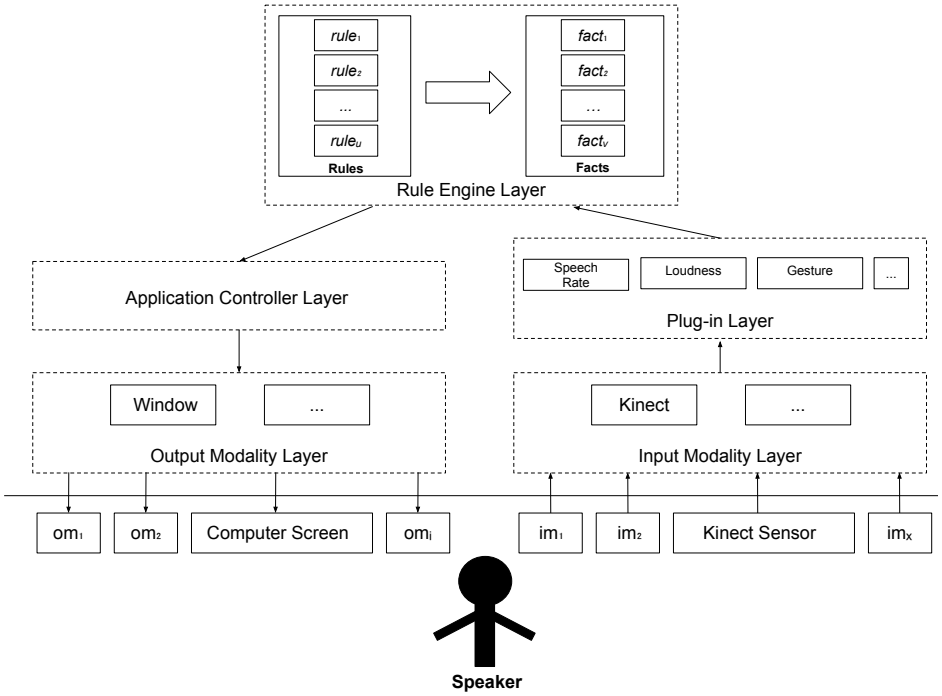


Figure 4.1: Architecture of PRESMA.

Acquiring data from a camera requires different techniques than, for example, a microphone. One way is to identify some "major" input modalities, and allow them to be compatible with *PRESMA*, but this approach will require constant updates whenever a new type of input modality would exist. This is why the IML should implement general and basic functionalities that could be applied to all kind of input modalities. The following identified functionalities should be supported by each included input modality, which are defined by a contract:

- **Start**: this functionality serves to start the input modality for the acquisition of data.
- **Stop**: this functionality serves to stop the input modality.
- **Add Subscribers**: serves to add, or "subscribe", plug-ins to an input modality in order to receive data from it. The input modalities serves only for the acquisition of data, while the plug-ins extracts meaningful information form them (e.g. skeleton positions of joints). The reason behind the separation of both concepts (input modality and plug-in) is explained in Section 4.1.2.

- **Is Possible Subscriber**: it is important that the plug-in receives data from the right type. Therefore, the plug-in should implement a specific method, in function of the input modality, for compatibility testing. For example, a speech plug-in would have to implement a `getSound()` method, allowing the Kinect modality to test for compatibility, and to be able to redirect the stream of sound data by calling this method at runtime.

- **Get Name**: each input modality should have a name serving as an identification in the graphical user interface.

A valid input modality would have to implement all the previous basic methods serving to make the system operable. Besides these methods, the programmer is responsible for gathering data from the input modalities and adding specific methods/functionalities by adding as much methods as wanted.

4.1.2 Plug-in Layer

The **Plug-in Layer** is responsible for adding detection logic to the system. Since the input modalities provide only *PRESMA* with raw data, the plug-ins therefore should give meaning to these data, for example, using the Kinect as input modality for the acquisition of audio data, and a plug-in for detecting words based on these data. Separating both layers would have as advantages that:

- Certain input modalities can only be instantiated once. Combining both concepts in one would result in combining multiple functionalities making the code complex and unreadable (e.g. combining gesture, speech, and so forth, in the case of the Kinect). It would also force the user to load all the functionalities (plug-ins), even when unwanted.

- By subscribing plug-ins to input modalities, the stream of data can therefore be redirected everywhere, and be used by a multitude of plug-ins (see Figure 4.2). Each plug-in could implement its own logic and be used/loaded by the user whenever wanted.

In order to allow the inclusion of detection logic by the means of plug-ins, each plug-in should therefore implement the following basic methods defined by a contract:

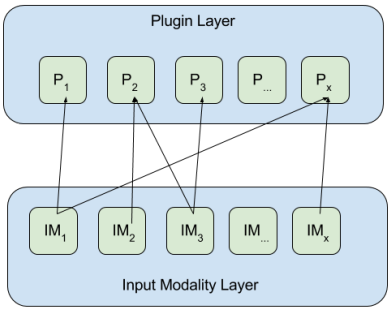


Figure 4.2: Redirection of data streams between the input modalities and plug-ins. *IM*: stands for Input Modality; *P*: stands for plug-in; and the arrow represent the stream of data.

- **Get Name**: serves as an identification of the plug-in in the GUI. By displaying the name of the plug-in, the user is able to make a distinction between the loaded plug-ins.

- **Get Configuration Page**: serves to configure the plug-in by the means of a GUI, which should be implemented by each loaded plug-in.

- **Configure Plug-in**: serves to configure the parameter(s), acquired from the Configuration Page of the plug-in, in the GUI. This would allow the user to parametrise the plug-ins in function of the context of presentation.

- **Get Facts**: facts serve to hold data from the plug-ins in the data memory used by the rule engine for the triggering of feedback messages. For a better understanding of the concept of facts and rules, Sections 4.1.3 and 5.1.3 explain the rule engine and Rete's algorithm in detail.

- **Add Fact**: serves to add, or in other words, subscribe a compatible fact to a specific plug-in. This will allow multiple facts, each tackling specific detection logics, to benefit from the stream of meaningful data generated by the plug-in.

- An input **modality-specific** method: every plug-in should implement a specific method, allowing it to be compatible with a specific input modality and to acquire a specific stream of data at runtime.

- **Compatibility Test**: serves to test whether a fact is compatible with a specific plug-in.

4.1.3 Rule Engine Layer

The rule engine is used to process the meaningful data acquired from the plug-ins and to trigger feedback messages whenever the data, stored in *facts*, satisfy certain *rules*. In other words, the rule engine gathers the data derived by the detection plug-ins and uses them to validate a set of conditions (rules). For example, a detection plug-in could count the amount of *"euhm"* (filler words) uttered per minute, and if there's a rule that says that the amount can not go over a certain threshold, the rule engine will automatically trigger an event so that the corresponding action can be taken. The goal is to allow the user to load a specific set of rules depending on the context of the presentation (e.g.: rule sets for business presentations, educative presentations, and so forth). Since the rules and facts are extensible in *PRESMA*, the user should combine compatible facts with plug-ins, in order to provide the required data to the rule engine for the triggering of rules. The rule facts should provide the following functionalities defined by a contract:

- **Data From Plug-in**: serves to acquire data from the plug-in and to store them in the fact itself, enabling the rule engine to fire feedback messages whenever the fact satisfies a rule.

- **Fire Feedback**: serves to fire a feedback message to the user whenever a rule is satisfied.

- **Configure fact**: serves to configure the fact with the thresholds defined by the users.

4.1.4 Application Controller Layer

The **Application Controller Layer** (ACL) is the intermediary layer between the Rule Engine and the Output Modality layers. Accessible to all the facts from the rule engine, the ACL gathers and redirects all the triggered feedback messages to the included output modalities, allowing them to present the message in any form to the speaker.

4.1.5 Output Modality Layer

The **Output Modality Layer** (OML) is the layer responsible for the inclusion of output modalities. All the loaded output modalities receive feedback messages from the Application Controller Layer, and present the information in a form specific to the output modality. In the case of *PRESMA*, a simple window displaying a feedback message to improve the behaviour of the speaker would be enough. The window should present one feedback message at the time with a specific duration defined by the speaker in order to avoid cognitive overload. Therefore, a valid output modality should provide the following functionalities defined by a contract:

- **Get Feedback Message**: serves to acquire feedback messages from the Application Controller and to present it to the presenter.
- **Get Configuration Page**: allows the user to configure the output modality by the means of a GUI.
- **Configure Output Modality**: serves to configure the output modality using the values defined by the user in the GUI (configuration page).
- **Start**: serves to start the output modality.
- **Stop**: serves to stop the output modality.

5 Implementation I: Architecture

This chapter is dedicated to the implementation of the *PRESMA*'s architecture presented in Chapter 4. Before explaining how each layer of the system works, the software technologies used to successfully achieve the implementation are first explained, providing enough insights and technical background needed to understand the technical parts.

5.1 Used Software Technologies

5.1.1 Managed Extensibility Framework

One of the main goals of *PRESMA* is to be a light-weighted and extensible application. **Managed Extensibility Framework** (MEF) [111] allows an application to be extended without touching the source code. The extensions or *plug-ins* are loaded and executed at runtime, via *composition*. Each MEF component, or part, defines both its capabilities and dependencies by the means of a "contract", often an interface. The contract interface defines basic methods to be implemented by the plug-ins, in order to look for plug-ins in the Dynamic Link Library (DLL) files satisfying the contract. This can be done easily by exploring the meta-data discoverable at runtime. MEF does not require hard and fragile dependencies, allowing to develop and test extensions independently from the application.

5.1.2 Reflection

Hardcoding is not an option for *PRESMA*, since it would limit the creativity of the programmers while developing plug-ins. The goal is to provide a kind of "freedom" to the programmer, while still defining the basic mechanism needed to run the tool correctly with MEF. Since everything is done at runtime, it becomes impossible to know in advance (while compiling the program) which modalities are compatible with the loaded plug-ins. **System.Reflection** [112] provides us the a way to access the type of an object, to invoke methods and access the fields and properties of loaded assemblies at runtime. Connecting input modalities with plug-ins would result in testing whether they are compatible or not by checking the correspondent methods.

5.1.3 Rule Engine

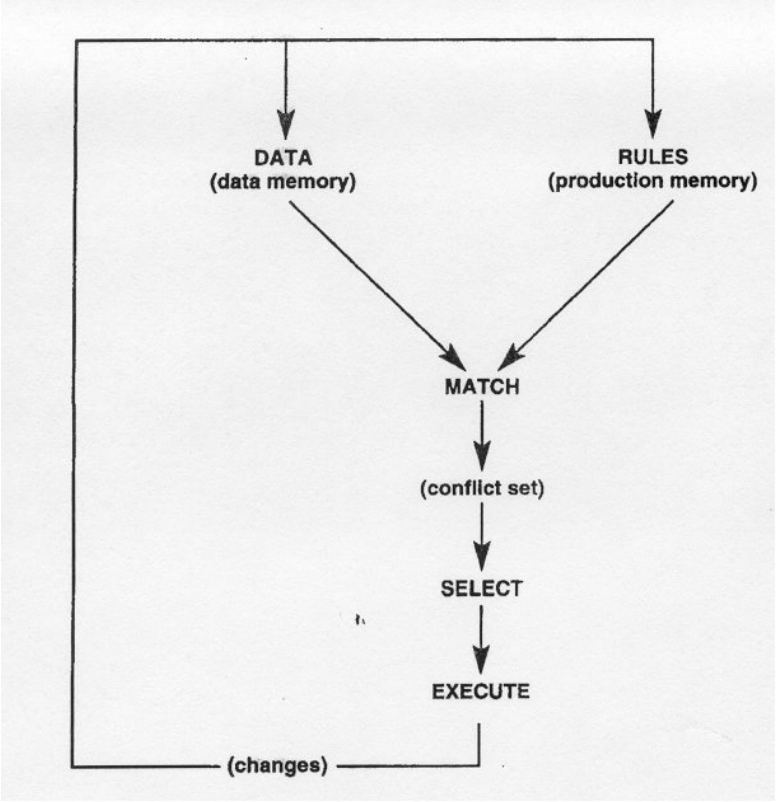
Practically most of the presented approaches in the related work Section 3.2 process the quantity of data acquired by the input modalities by:

- Defining all the possible situations manually for the triggering of feedback messages to the speaker.
- Using algorithms to generate feedback messages.

Using one of these techniques would have as repercussion that the source code becomes less maintainable, making changes in the future difficult. A **rule engine** could avoid a number of issues by expressing difficult imperative code into simple and easy to read declarative rules. By defining "what to do" instead of "how to do it", by means of rules, would result in less lines of complex code, and would decouple the data from logic compared to object-oriented languages [113]. Since all the logic is centralised in one specific place, it becomes easy to maintain and to modify. Therefore, the plug-ins would have to pass relevant and meaningful data to the rule engine, which will trigger the feedback messages whenever needed.

Several rule engines are available and compatible with the C#.NET programming language. Nevertheless, they are not all maintained and well documented (the case of Drools.NET [114]). **NRules** [115] is a simple and powerful production system rule engine based on the Rete's pattern matching algorithm [116]. The rule engine contains a database composed of *rules* defining the behaviour and *facts* containing data (acquired by the plug-ins) in the knowledge base. As presented in Figure 5.1, the rule engine examines the facts, in the data memory, and the rules, in the production memory, to

determine which rules must be triggered whenever they satisfy the facts. The rule engine performs a kind of conflict resolution to determine which rule will be fired. The rule is then executed.



nbol → is the

Figure 5.1: Production system type rule engine based on the Rete's algorithm [116].

5.1.4 C#.NET Programming Language

C#.NET is a static, type-safe programming language running on the Microsoft's .NET framework enabling developers to build secure and robust Windows client applications, web services, and more [117]. C#.NET has been preferred over other programming languages for the fact that it would allow a possible integration of PowerPoint in the future by the means of .NET plug-ins, which is not possible with other programming languages or with the Mac OS X environment. Since the Kinect and Managed Extensibility Framework are all technologies developed by Microsoft, this programming

language is incorporated with all the needed libraries to achieve the project successfully.

5.1.5 Windows Presentation Foundation

Windows Presentation Foundation (WPF) [118] is Microsoft's latest user interface framework used to create GUIs for desktop applications. The creation of rich user interfaces is simplified thanks to the **eXtended Application Markup Language** (XAML) [119], which is a declarative markup language used to create and declare UI elements. Moreover, it separates the GUI elements from the logic situated in the code-behind files.

5.1.6 Model-View-ViewModel Pattern

The **Model-View-ViewModel** (MVVM) [120] is Microsoft's MVC pattern variation used to make a clean separation between the user interface and the logic. As presented in Figure 5.2, the MVVM pattern has three conceptual parts: the model, the view and the view model:

- The *view* can be seen as the window containing all the UI elements that the user can see. A view is divided into two parts, the XAML part for defining and structuring the UI elements and the code-behind that does not contain any business logic and is practically always used to initialise de UI elements or even to process events when they occur.
- The *model* is typically the layer that deals with data, for example, classes used to interact with the database by performing SQL statements.
- The *view model* is an intermediate layer that lays between the model and the view. Its goal is to handle the view logic and to interact with the classes available in the model layer. The view model can gather data from the model layer and present it to the user by providing it back to the view.

5.2 Implementation of the Input Modality Layer

As explained in Section 5.1.1, the **Managed Extensibility Framework** is used by *PRESMA* to dynamically load input modalities at runtime. Therefore, every input modality should satisfy the contract by implementing the

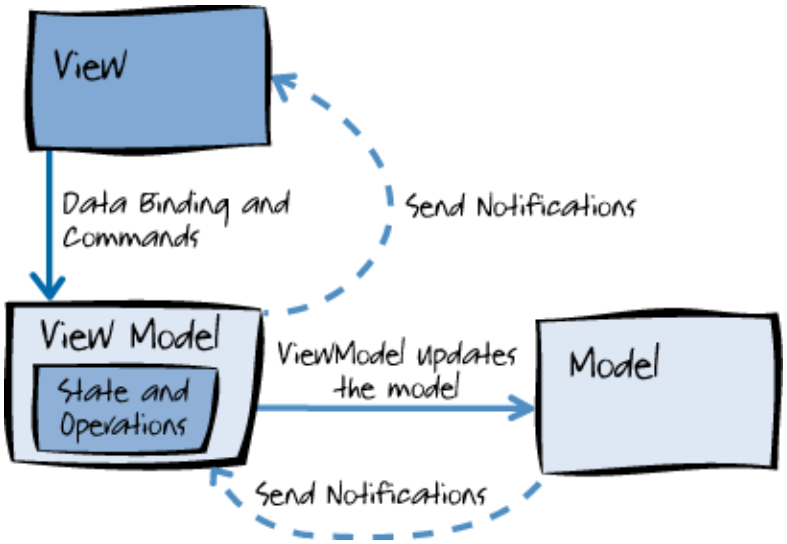


Figure 5.2: The MVVM-pattern [120].

methods of the interface **IInputModality**. Every method is important for the working of the system. For a better understanding of each method, section 4.1.1 explains the basic idea behind all of them. In order to build a new input modality compatible with the *PRESMA*, the following steps should be taken in consideration:

1. A new **Class Library** should be created in Visual Studio. This class library will incorporate all classes, interfaces, and so forth, needed for making the input modality operable.

2. There should be at least one **class** responsible for the implementation of the contract listed in Listing 5.1.

3. The class library project should be compiled, generating a valid **DLL** file.

4. Finally, the DLL file should be placed in a map where every input modality will be loaded at runtime by the framework.

Listing 5.1: The input modality contract.

```
public interface IInputModality
{
 //This method is used to start the input modality.
 void start();

 //This method is used to stop the input modality.
 void stop();

 //This method is used for adding the plugin to the list of subscribers.
 bool addSubscriber(IPlugin plugin);

 //This method is used to test whether the plugin can be added to the subscribers
 bool isPossibleSubscriber(IPlugin plugin);

 //This method is used to return the name of the Input Modality.
 string getPluginName();
}
```

Whenever *PRESMA* is executed by the user, the constructor of the class **InputModalityFramework** (listed in Listing 5.2) will automatically start looking in the directory *Plugins* for assemblies, by using *DirectoryCatalog*. The *AggregateCatalog* dictionary has the ability to collect all the exported assemblies and to make them available for further use.

Listing 5.2: The InputModalityFramework constructor.

```
DirectoryCatalog directoryCatalog = new DirectoryCatalog(path);
var catalog = new AggregateCatalog(directoryCatalog);
_Container = new CompositionContainer(catalog);
_Container.ComposeParts(this);
```

Calling the `ComposeParts(this)` method on the *CompositionContainer*, containing all the available imported parts of the system, will perform composition in order to match up the imports with the exports (e.g. the Kinect) satisfying the contract defined by the **IInputModality** interface. This will allow the *ImportMany* attribute specified with the type of the contract to populate the list *inputModalities* with input modalities (see Listing 5.3).

Listing 5.3: The *inputModalities* list containing the loaded input modalities.

```
[ImportMany(typeof(IInputModality))]
internal IEnumerable<IInputModality> inputModalities;
```

In order to fulfil the import process and to fill the container with valid exports, every input modality should declare the *Export* attribute in combination with the type of contract right above the classname (as shown in Listing 5.4).

Listing 5.4: The export attribute and the input modality contract.

```
[Export(typeof(IInputModality))]
public class KinectIM : IInputModality
{
 //...
}
```

Now that the imported input modalities are available and ready for use, each available input modality in the list *inputModalities* can be started

and stopped by the means of the implemented `startModalities()` and `stopModalities()` methods defined by the contract (see Listing 5.5).

Listing 5.5: Starting and stopping input modalities.

```
public void startModalities()
{
 foreach (IInputModality inputModality in inputModalities)
 {
  inputModality.start();
 }
}
public void stopModalities()
{
 foreach (IInputModality inputModality in inputModalities)
 {
  inputModality.stop();
 }
}
```

5.3 Implementation of the Plug-in Layer

The concept of importing the plug-ins into the system is the same as applied for the Input Modality Layer, with the exception of the contract which is different. The contract **IPlugin** (as shown in Listing 5.6) defines the basic methods serving to test the validity of a plug-in. The basic idea behind each of the methods is explained in Section 4.1.2.

Listing 5.6: The Plugin Contract.

```
public interface IPlugin
    {
     // This method is used to return the plugin name.
        string getPluginName();

        //This method is used to configure/initialize the plugins.
        void configurePlugin(Page p);

        //This method is used to get facts from the plugins.
        //Facts are used as objects containing useful data for the rule/rule engine.
        List<IFact> getFacts();

        //This method is responsible for adding facts to the list of facts
        void addFact(IFact fact);

        //This method is used to return the Page that will be used to configure the plugin and co.
        //The method should return null whenever no Page is used.
        Page getConfiguratorPage();
}
```

Since the same method is applied for the import of plug-ins, the **PluginLoader** class contains also the list *plugins* populated with plug-ins satisfying the contract **IPlugin** and declared with the *Export* attribute (see Listing 5.7 and 5.8).

Listing 5.7: The list of imported plugins.

```
[ImportMany(typeof(IPlugin))]
internal IEnumerable<IPlugin> plugins;
```

Listing 5.8: The export attribute and the plug-in contract.

```
[Export(typeof(IPlugin))]
public class GesturePlugin : IPlugin
{
 //...
}
```

5.4 Subscribing Plug-ins to Input Modalities

The challenge resides in allowing plug-ins to subscribe to input modalities in order to benefit from the stream of data generated by the last ones. Whenever the user configures the system by subscribing plug-ins to input modalities, the `controlCompatibilities()` method (see Listing 5.9) implemented in the **frmSubscriptionLinker** class will first test whether those two are compatible or not.

Listing 5.9: The compatibility checker method.

```
bool isPossibleSubscriber = true;
isPossibleSubscriber = selectedModality.isPossibleSubscriber(link.Key);

if (!isPossibleSubscriber)
{
 if (isCompatible)
  isCompatible = false;
 msgError += "Plugin " + link.Key.getPluginName() + " is not compatible with " +
       selectedModality.ToString() + Environment.NewLine;
}
```

The *selectedModality* is the input modality selected by the user that should add the plug-in *link.Key* as a subscriber to its stream of data. If the compatibility test fails, the user is provided with a feedback message *msgError*. In the other case, the plug-in will successfully be a subscriber of the selected modality and stored in the selected input modality. The `isPossibleSubscriber()` method is guaranteed to exist since it is defined by the **IInputModality** contract, making it therefore callable. Since the subscription process occurs at runtime, the only way for the system to perform the compatibility test is to use **Reflection**. The **CompatibilitySubscriber** class implements the method `hasMethod()` (see Listing 5.10) allowing the system to check compatibility at runtime. To be compatible, a plug-in should implement a method capable of acquiring the stream of data from a specific input modalities, in this case the selected one.

Listing 5.10: The method used to check the presence of a specific method with Reflection.

```
public static bool hasMethod(this object plugin, string methodName)
```

```
{
 try
 {
  var type = plugin.GetType();
  var method = type.GetMethod(methodName);

  if (method == null)
   return false;
  else
   return true;
 }
 catch (AmbiguousMatchException)
 {
  return true;
 }
}
```

5.5 Implementation of the Rule Engine Layer

The rule engine **NRules** used by *PRESMA* can be regarded as being the core of the system. The rule engine is provided with data from all the imported plug-ins and has as main goal to trigger feedback messages to the speaker whenever certain nonverbal communication aspects, measured by the plug-ins, are not applied correctly. The rule engine consists of two parts, namely the domain model and rules. The domain model, or *facts*, stores the meaningful data acquired by the plug-ins. The rule engine also contains a set of rules, which are checked against the domain model in order to trigger feedback messages where appropriate.

Introducing NRules into the project as simple as executing one command in the Package Manager Console of the Visual Studio Environment (see Listing 5.11).

Listing 5.11: The install NRules command.

```
PM> Install-Package NRules
```

After the installation of the rule engine, creating an instance of the **RuleEngine** class allows to benefit from all its functionalities. The method `createRuleEngine()` (see Listing 5.12) allows plug-ins to create a rule engine from an instance of the **RuleEngine** class and needs as parameter a list of facts, more precisely, the facts where the data will be stored from the plug-ins. Therefore, the plug-in can provide the rule engine with the needed facts by means of the `getFacts()` method imposed by the plug-in contract. The *repository* will load all the available rules of the type **IRule** in order to allow the Rete's network (internal structure of the rule engine) to know what the rules are and to easily match them with facts. After compiling the rules, the next steps consists of creating a session and to insert the facts in the rule engine's memory.

Listing 5.12: Creating a rule engine.

```
public class RuleEngine
{
  public void createRuleEngine(List<IFact> facts)
        {
  //Load rules (subtypes from type/interface Rule)
  var repository = new RuleRepository();
  repository.Load(x => x.From(typeof(IRule).Assembly));

  //Compile rules
  var factory = repository.Compile();

  //Create a working session
  session = factory.CreateSession();

  if (facts != null)
  {
   foreach (IFact fact in facts)
   {
    session.Insert(fact);
   }
  }
 }

 //...
}
```

The `updateFact()` provides the possibility to the plug-ins to update the data (variables) from the facts in the engine's memory. To avoid the system to freeze or to slower up, every time this method is called, a *thread* will take care of the update by calling the `updateFactData()` method (see Listing 5.13). After finishing trying to update the fact, the `session.Fire()` method fires the rules with the newly updated data in order to fire feedback messages to the speaker.

Listing 5.13: The thread responsible for firing the rules.

```
//...

public void updateFact(IFact fact)
{
 toUpdateFact = fact;

 othreadUPD =  new Thread(new ThreadStart(updateFactData));
 othreadUPD.Start();
 othreadUPD.Join();
}

private void updateFactData()
{
 //Update the fact
 session.TryUpdate(toUpdateFact);

 Thread.Sleep(10);

 //Fire the rules & rule engine.
 session.Fire();
}
//...
```

The **IRule** interface implemented by all the rules of the system does not serve as a contract in this case, nor as a way to force the implementation of specific methods, but is provided as a type (see Listing 5.14). In other words, since all the rules would have a distinct class name, implementing the **IRule** would ease the rule engine to load rules from the assemblies, by simply checking whether the parent class is indeed **IRule**.

Listing 5.14: The contract used for rules.

```
public interface IRule
{
 //This interface is used to load all the rules that implement this interface.
}
```

The facts therefore need to implement a specific behaviour since they all share two common functionalities, which are:

- The acquisition of data from the plug-ins.
- Firing a specific feedback message.

The **IFact** interface (see Listing 5.15) implemented by all the facts forces the implementation of two methods, namely `feedbackFromPlugins()` for the acquisition of data, and `fireFeedback()` for firing feedback messages. The first method has as parameter an array of objects containing the data from the plug-ins. The reason behind choosing an array of type **object** is that every type in C#.NET inherits direct or indirectly from it [121]. Which means that the array is able to store objects from all types, going from audio data, to images, and so forth. The only complication with this method of working is to impose, programmatically or by documentation, the positions of the data stored in the array, for example, *integer* x is stored in position one, while *string* y is stored in position two. When a fact is subscribed to a plug-in, the `configureFact()` method is called by the plug-in with the thresholds defined by the user in the GUI.

Listing 5.15: The contract used for facts.

```
public interface IFact
-{
 //Method used to be triggered from the plugins.
        void feedbackFromPlugins(object[] parameters);

        //Method used to fire feedback.
        void fireFeedback();

         //Configure the fact with this method
        void configureFact(object[] parameters);
}
```

Finally, whenever a rule is satisfied, the `fireFeedback()` method will be called triggering a feedback message to the speaker.

5.5.1 Subscribing Facts to Plug-ins

Since the rule engine is extensible, the user is able to add detection logic, or communication aspects to the system by developing or importing facts and rules implementing the **IFact** and the **IRule** contract interfaces. When the rule engine is initialised, all the assemblies available in the project, imported or not, will be analysed in order to create a Rete's network consisting of

rules and facts. Since a presentation should be adequate to the presentation context and needs, *rulesets* can be defined programmatically, in order to group one or more communication aspects destined to a specific kind of presentation session. For example, one may create a ruleset for business presentations, while another one for educational purposes. As presented in Figure 5.3, every ruleset should implement the **IFact** interface as super-type. Therefore, the rule engine will be able to allow the user to choose a ruleset adequate to a specific kind of presentation, and to load the facts with their corresponding rules in the Rete's network.

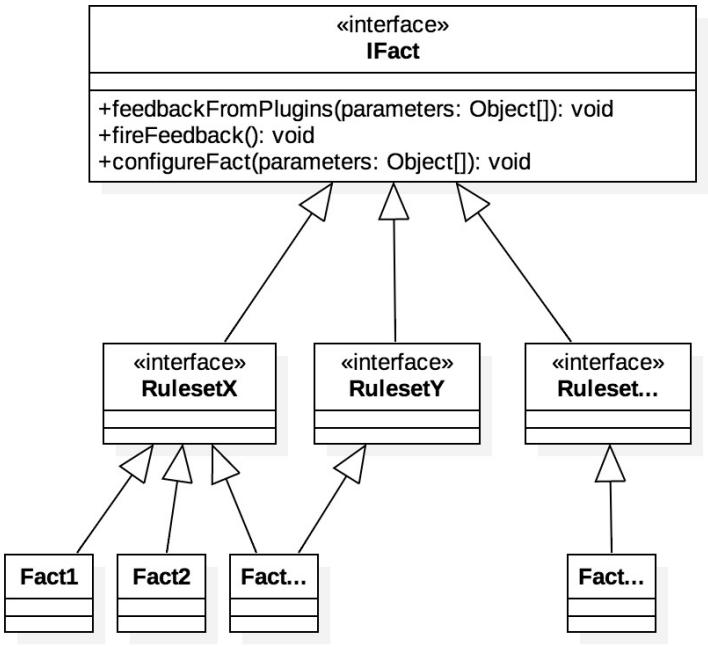


Figure 5.3: Class diagram representing the IFact contract, rulesets and facts.

Since the whole process is performed at runtime, facts are not hardcoded to plug-ins and need therefore to subscribe to their stream of data. Similar to how plug-ins subscribe to input modalities, explained in Section 5.4, the same method is applied for subscribing facts to plug-ins.

5.6 Implementation of the Application Controller Layer

The task of the Application Controller Layer consists of gathering the feedback messages obtained by the facts, and to redirect them to all the loaded output modalities. The layer consists of a *static* class **ApplicationController** accessible by all the facts. Whenever the feedback message is triggered by the rule engine, the `fireFeedback()` method implemented by all the facts, should call the Application Controller's `getFeedbackMessage()` method (see Listing 5.16). The message will then be redirected to all the imported output modalities, in order to present the feedback message in a particular form.

Listing 5.16: The methods used to receive and redirect feedback messages.

```
public static void getFeedbackMessage(object[] message)
{
 redirectFeedbackMessage(message);
}

private static void redirectFeedbackMessage(object[] message)
{
 if (outputModalityFramework == null)
  return;

 foreach(IOutputModality outputModality in outputModalityFramework.getOutputModalities())
 {
                outputModality.getFeedbackMessage(message);
 }
}
```

5.7 Implementation of the Output Modality Layer

The Output Modality Layer uses **Managed Extensibility Framework** to dynamically load output modalities at runtime. Therefore, every output modality should satisfy the contract by implementing the methods of the interface **IOutputModality**. Since the implementation of the Output Modality Layer is similar to the Plug-in Layer, Section 5.3 can serve as a technical basis, while the basic idea behind each method is explained in Section 4.1.5.

6

Implementation II: Plug-ins and Rules

The previous chapter described the architecture of *PRESMA*, which can be regarded as the framework that ties the various components of the system together. In order to demonstrate the working of the system, four nonverbal communication skills have been identified, which are the use of sufficient gestures during a presentation, providing sufficient eye contact, speaking loud enough, and having an appropriate speech rate. This makes the Microsoft Kinect a good candidate for this thesis, since the sensor will provide all the required data in order to be able to measure the identified nonverbal presentation skills, making it also the primary input modality. Since this part of the report focusses more on the application of the concepts explained in the architecture chapter, it provides also a strong basis for anyone wanting to extend the system.

6.1 The Microsoft Kinect Input Modality

6.1.1 Microsoft Kinect and Microsoft Kinect SDK

Human beings have multitude of senses, which allows us to see, touch, hear, etc. [122]. While computers do not possess the same abilities (senses) as human beings, it is nevertheless possible to simulate the senses by coupling computers with sensors. This will not only make computers or everyday's objects smarter, but also allows the development of real-time information systems [14]. Since one of the main goals of this thesis is to provide efficient feedback in real-time, sensors can be useful for measuring and detecting the physical properties [123] of the speaker, such as voice, gestures, heart rate, etc. and process this information to alert the speaker when needed.

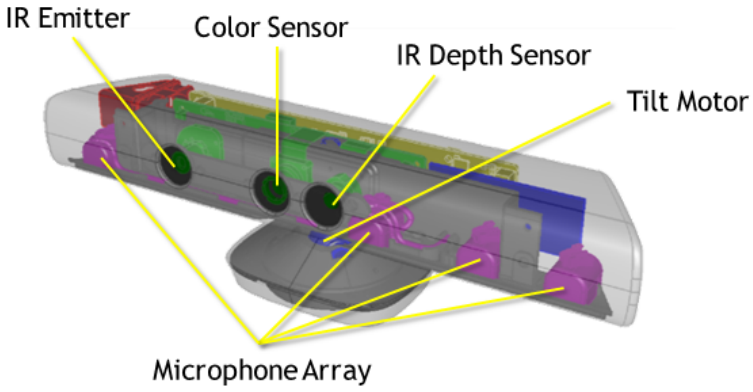


Figure 6.1: The Microsoft Kinect sensor components [124].

The Microsoft Kinect (version 1) can not only be used for game devices such as the Xbox, but it is also available with its Software Development Kit (SDK) in order to develop desktop applications while benefiting of all of its advantages [125]. As shown in Figure 6.1 , the Kinect is composed out of:

- An RGB camera for capturing and storing coloured images.
- An infrared emitter and an infrared depth sensor for measuring the distance between objects.
- A multi-array microphone for the recording of sound, determining the location and the direction of sound.
- An accelerometer for determining the orientation of the Kinect.

Thanks to the Microsoft Kinect SDK (version 1.8) [126], developers are provided with a sophisticated software library allowing the creation of rich and innovative applications and Natural User Interfaces, using gesture and voice recognition. As presented in Figure 6.2, the NUI library allows the application to access three types of streams of data:

- Audio Stream
- Color Stream
- Depth Stream
- Infrared Stream

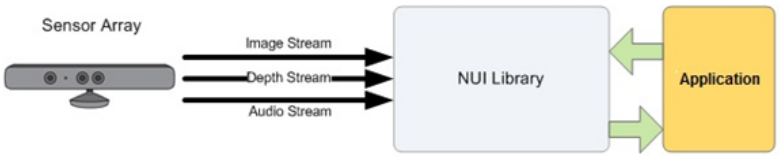


Figure 6.2: Interaction between Microsoft Kinect SDK and an application [126].

As mentioned before, the Microsoft Kinect includes a four-element microphone array for capturing high quality 16-kHz, 24-bit audio data. The **Audio Stream** allows an application to:

- Capturing audio from a specific direction
- Accessing raw audio data
- Identifying audio sources
- Perform speech recognition with captured audio data

The **Colour Stream** provides coloured image data acquired from the Kinect's RGB camera in different formats and resolutions. Depending on the bandwidth available from the USB connection, one may adjust the resolution of the frames, going from 1280x960 for a high-resolution image to 640x480 for a lower-resolution frame, in order to avoid the loss of image quality.

The pixels of depth frames acquired by the **Depth Stream** contain the cartesian distance, in millimetres, from the Kinect sensor to the nearest object with a particular (x,y)-coordinate as shown in Figure 6.3. The depth frame is available in three different resolutions (640x480, 320x240, and 80x60) and can identify up to 6 human figures.

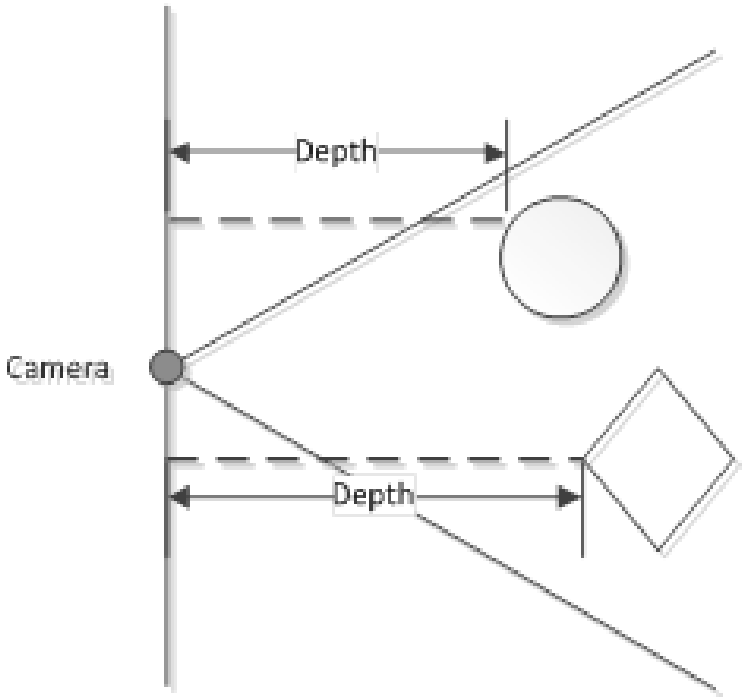


Figure 6.3: [USE BETTER IMAGE] Depth stream distances [127].

The **Skeleton Frame** contains 3D position data for up to two people. The position of each human skeleton joint (see Figure 6.4), expressed in meters, is available through (x, y, z)-coordinates.

The **AllFramesReady Stream** provides the application with all the active frames of the Kinect sensor. When new frames are captured and made available, an event will be fired, making it able to access the coloured image frame, depth frame and skeleton frame at once.

6.1.2 Implementation of the Kinect Input Modality

Even if the system supports different kinds of input modalities, the only input modality used so far is the Microsoft Kinect, since it provides the system

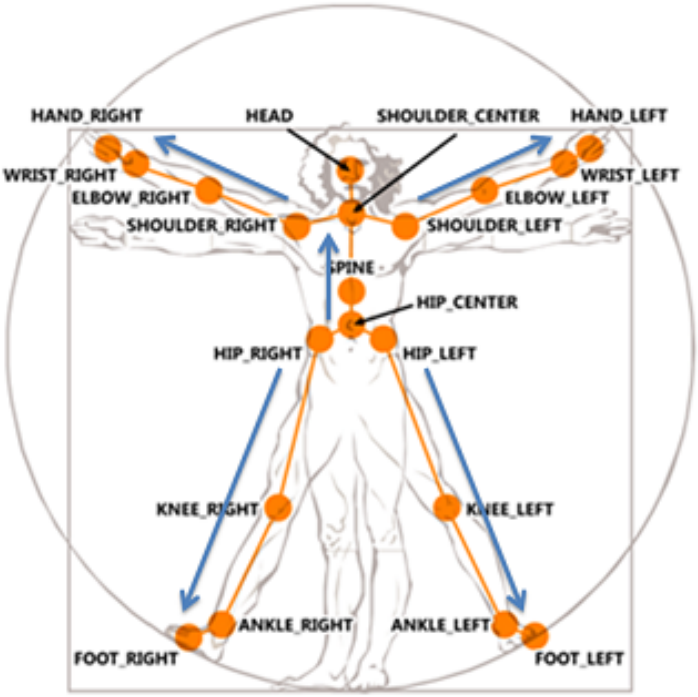


Figure 6.4: The human skeleton joints [128].

with audio, depth, and skeleton data about the speaker, allowing the plug-ins to work correctly and successfully. In order for the Microsoft Kinect to be loaded by the framework and to be recognised as being a valid input modality, the **KinectIM** class should therefore satisfy the input modality contract by implementing the **IInputModality** interface (listed in Listing 5.1), and declare the *Export* attribute (see Listing 6.1). Doing this will also force the class to implement the specified methods in order to make the system work correctly.

Listing 6.1: The Kinect sensor as a valid input modality.

```
[Export(typeof(IInputModality))]
public class KinectIM : IInputModality
{
 //...
}
```

The goal of the Microsoft Kinect is to generate the streams of data and to redirect them to plug-ins as presented in Figure 6.5. In order to do so, every plug-in should be subscribed to the appropriated stream of data. Therefore, the **KinectIM** class implements a *List* for every stream, allowing to store the subscribed plug-ins, which are:

- The list *audioSubscribers* holds the plug-ins requiring audio data, such as the Speech Rate and Loudness plug-in.
- The list *skeletonSubscribers* holds the plug-ins requiring skeleton data, such as the Gesture plugin.
- The list *allFramesReadySubscribers* holds the plugins requiring all the Kinect frames at once, such as the Sight plug-in responsible for the measuring of "eye contact" by the means of face tracking.
- The list *videoSubscribers* holds the plug-ins requiring the coloured images generated by the Kinect. Up to now, no plug-ins are using this stream of data, but could be useful in the future, especially while building a plug-in showing recorded sessions of the speaker annotated with with feedback messages.

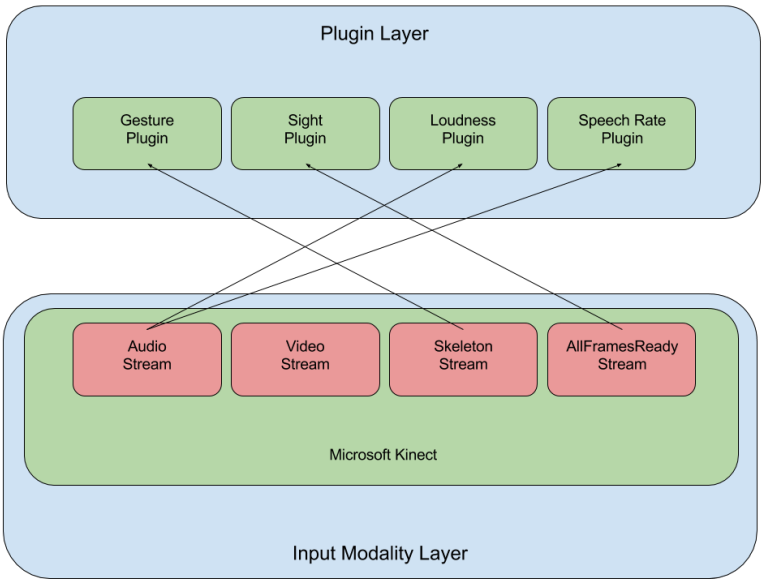


Figure 6.5: Redirection of the streams of data from the Microsoft Kinect to the plug-ins.

Compatibility Testing

Before being able to add plug-ins to one of the identified lists, the plug-ins must be compatible with the Kinect input modality sensor. Since the Kinect class is forced to implement the `isPossibleSubscriber()` method with a

plug-in as parameter, the programmer is therefore responsible for performing the compatibility test with the `CompatibilitySubscriber.hasMethod()` method using Reflection (see Listing 6.2). A specific plug-in is compatible with the Kinect sensor whenever it implements at least one of the following methods:

- The `getAudio()` method that will be called for sending audio data to the plug-in.
- The `getImageFrame()` method that will be called by the Kinect sensor for sending an coloured image frame.
- The `getSkeletonFrame()` method that will be called by the Kinect sensor for sending a skeleton frame.
- The `getAllFrames()` method that will be called by the Kinect sensor for whenever all the frames are ready.

Listing 6.2: The compatibly tests using Reflection.

```
public bool isPossibleSubscriber(IPlugin plugin)
{
 bool isCompatible = false;

 if (Compatibility.CompatibilitySubscriber.hasMethod(plugin, "getAudio"))
 {
                isCompatible = true;

                //Add to audio list
                if (null == audioSubscribers)
   audioSubscribers = new List<IPlugin>();

  audioSubscribers.Add(plugin);
 }
 if (Compatibility.CompatibilitySubscriber.hasMethod(plugin, "getImageFrame"))
 {
                isCompatible = true;

                //Add to video list
                if (null == videoSubscribers)
   videoSubscribers = new List<IPlugin>();

  videoSubscribers.Add(plugin);
 }
 if (Compatibility.CompatibilitySubscriber.hasMethod(plugin, "getSkeletonFrame"))
 {
                isCompatible = true;

                //Add to skeleton list
  if (null == skeletonSubscribers)
   skeletonSubscribers = new List<IPlugin>();

  skeletonSubscribers.Add(plugin);
 }
 if (Compatibility.CompatibilitySubscriber.hasMethod(plugin, "getAllFrames"))
 {
                isCompatible = true;

                //Add to All frames ready list
  if (null == allFramesReadySubscribers)
    allFramesReadySubscribers = new List<IPlugin>();

  allFramesReadySubscribers.Add(plugin);
 }
 return isCompatible;
}
```

6.1.3 Subscription to the Kinect Streams

The Kinect's event model allows the acquisition of the data stream captured by the Kinect sensor by the means of event subscription. Whenever the application is subscribed to one or more event handler, the sensor will continuously send new frames until the application is stopped or unsubscribed. The *sensor* variable available in the **KinectIM** class contains an instance of the connected Kinect sensor, allowing it to be used by the whole class.

Subscription to the Audio Stream

Subscribing the application to the audio stream event results in starting the Kinect's *AudioSource*, and starting the *tAudioReading* thread responsible for recording the audio stream. This thread is independent from the application's main thread, and will therefore avoid freezing the GUI.

The *tAudioReading* thread records the audio captured during a presentation session and store it into a **WAV** audio file, for further use. Since the thread continuously reads and stores audio data in a buffer *audioBuffer*, the `AudioSourceReady()` method (see Listing 6.3) will be called, serving to send the buffer to all the plug-ins subscribed to the audio stream.

Listing 6.3: The method used to redirect audio data to the subscribed plug-ins.

```
private void AudioSourceReady()
{
 if (audioSubscribers != null)
 {
  string[] nameOfParameters = new string[1] { "audioBuffer" };
                object[] parameters = new object[1] { buffer };

                foreach (IPlugin plugin in audioSubscribers)
                {
   plugin.GetType().InvokeMember("getAudio", BindingFlags.InvokeMethod, null, plugin,
                              parameters, null, null, nameOfParameters);
                }
 }
}
```

For each subscribed audio plug-in available in the *audioSubscribers* list, the *AudioSourceReady* method uses Reflection to invoke the `getAudio()` method dynamically at runtime with the buffer containing the audio data stored in the *parameters* array.

Subscription to the Skeleton Stream

The skeleton stream provided by the Kinect allows to receive information from the recognised users, such as the position of the limbs, which is useful for tracking the face and the limbs of the speaker. Identical to the audio stream, the application has to subscribe to the skeleton events, providing

the skeleton frames obtained by the Kinect. Therefore, the Kinect's *SkeletonStream* must be enabled which will capture the skeleton frames by the means of the *SkeletonFrameReady* event handler.

Whenever a skeleton frame is ready to use, the *SkeletonFrameReady* event handler will redirect the skeleton frame *skeletonFrame* to all the subscribed plug-ins stored in the *skeletonSubscribers* list, by invoking the method `getSkeletonFrame()` by the means of the Reflection's `InvokeMember()` method (see Listing 6.4).

Listing 6.4: The method used to redirect the skeleton frames to the subscribed plug-ins.

```
private void Sensor_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
     using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
     {
         if (skeletonFrame != null)
         {
               //skeletonFrame.CopySkeletonDataTo(this.skeletonData);
               if (skeletonSubscribers != null)
               {
                    string[] nameOfParameters = new string[1] { "skeletonFrame" };
                    object[] parameters = new object[1] { skeletonFrame };

                    //invoke with reflection
                    foreach (IPlugin plugin in skeletonSubscribers)
                    {
                         plugin.GetType().
                           InvokeMember("getSkeletonFrame",
    BindingFlags.InvokeMethod, null, plugin,
    parameters, null, null, nameOfParameters);
}}}}}
```

Subscription to the AllFramesReady Stream

The *AllFramesReady* stream provides the application with a coloured image, a skeleton frame, and a depth image at the same time, allowing to perform face tracking to compute the sight line, or "eye contact", of the speaker. Since the skeleton stream is already enabled, the depth and colour streams should be enabled before subscribing to the *AllFramesReady* events.

Whenever the `Sensor_AllFramesReady()` event is fired by the Kinect sensor, the all frames ready argument *e* makes it possible to access the coloured, skeleton, and the depth frame captured by the sensor. This allows the event handler to redirect the three frames, stored in the array *parameters*, to the plug-ins subscribed in the *allFramesReadySubscribers* list using the Reflection's *InvokeMember* method (see Listing 6.5).

Listing 6.5: The method used to redirect all the frames to the subscribed plug-ins.

```
private void Sensor_AllFramesReady(object sender, AllFramesReadyEventArgs e)
{
     if (sensor == null)
          return;

     ColorImageFrame colorFrame = e.OpenColorImageFrame();
```

```
    SkeletonFrame skeletonFrame = e.OpenSkeletonFrame();
    DepthImageFrame depthFrame = e.OpenDepthImageFrame();

    if (allFramesReadySubscribers != null)
    {

        string[] nameOfParameters = new string[4] { "sensor",
            "skeletonFrame",
            "depthFrame",
            "colorFrame" };

        object[] parameters = new object[4] { sensor,
            skeletonFrame,
            depthFrame,
            colorFrame };

        //invoke with reflection
        foreach (IPlugin plugin in allFramesReadySubscribers)
        {
            plugin.GetType().InvokeMember("getAllFrames",
              BindingFlags.InvokeMethod, null, plugin,
                      parameters, null, null, nameOfParameters);
        }
}
```

Starting and Stopping the Kinect Sensor

Now that the plug-ins are subscribed to one of the Kinect's stream of data, the last thing that remains is to call one of the identified methods whenever a frame of data is ready. When *PRESMA* is configured and executed, the framework will start and stop the input modalities by the means of the `start()` and `stop()` methods forced by the **IInputModality** contract. The Kinect class will look for the connected Kinect sensor and start the audio, video, skeleton, and all frames ready streams. Ending a session with *PRESMA* will stop all the data streams and remove the reference to the Kinect sensor.

6.2 Gesture Detection Plug-in

The use of gestures has been identified as being one of the most important nonverbal communication skills, making presentations more dynamic while also strengthening the delivered message. Similar to the Presentation Trainer [2], the gesture plug-in does not detect specific gestures, but determines whether the speaker is using gestures or not. Taking advantage of the skeleton frame from the Kinect, providing the positions of the limbs, the gesture plug-in is able to determine whether gestures are used by simply computing the angle between the forearms and arms, and between the arms and shoulder blades [2]. Whenever the angle starts increasing or decreasing, making the angle exceed a certain amount of degrees, signifies that a gesture has been detected. The user of *PRESMA* is free to define the value of the angle, but the Presentation Trainer authors [2] recommend 5 degrees.

6.2.1 Importing the Gesture Plug-in

The **GesturePlugin** class is able to be loaded by the **PluginLoader** class, since it implements the **IPlugin** contract, making it a valid plug-in, and declares being an *Export* of type **IPlugin** (see Listing 6.6). The working of this plug-in is ensured by the contract, since it forces the plug-in to implement the methods listed in Code 5.6.

Listing 6.6: Declaring the gesture plug-in as a valid plug-in.

```
[Export(typeof(IPlugin))]
public class GesturePlugin : IPlugin
{
 //...
}
```

6.2.2 Computing the Gesture Angle

During the configuration of *PRESMA*, the implementation of the method `getSkeletonFrame()` ensures the gesture plug-in to be compatible with the Kinect sensor (see Listing 6.7). The *skeletonData* array is initialised, which will store the positions of the tracked speaker's limbs. The obtained *skeletonFrame* from the sensor is then forwarded to the `getTrackedUser()` method, responsible for copying the skeleton data from the frame to the *skeletonData* array, which will be used later on to compute the angle.

Listing 6.7: Acquiring the skeleton frame from the input modality.

```
public void getSkeletonFrame(SkeletonFrame skeletonFrame)
{
 skeletonData = new Skeleton[skeletonFrame.SkeletonArrayLength];
 getTrackedUser(skeletonFrame);
}
```

The gesture plug-in makes a distinction between two kind of gestures, the ones performed by the left side of the speaker's body, and the ones performed by the right side. Therefore, the positions of the elbow, wrists, and shoulders of the tracked speaker are stored in vectors with their correspondent 3-dimensional (x, y, z)-coordinates. Computing the angles α and β (see Figure 6.6) necessary for the detection of gestures requires two vectors, one obtained by subtracting the elbow and shoulder vectors, and the second one by subtracting the elbow and the wrist vectors. Based on the two obtained vectors, the `AngleBetweenTwoVectors()` method is able to compute the angle as follows:

1. Normalising both vectors with the following formula:

$$\widehat{u} = \frac{\overrightarrow{u}}{\| \overrightarrow{u} \|} \tag{6.1}$$

2. Multiply both normalised vectors using the dot product methodology as follows:
$$\overrightarrow{u} \cdot \overrightarrow{v} = u_x \cdot v_x + u_y \cdot v_y + u_z \cdot v_z \tag{6.2}$$

3. Computing the angle in degrees with the following formula, where d is the dot product obtained in the previous step:
$$angle = \frac{\arccos(d)}{\pi} 180 \tag{6.3}$$

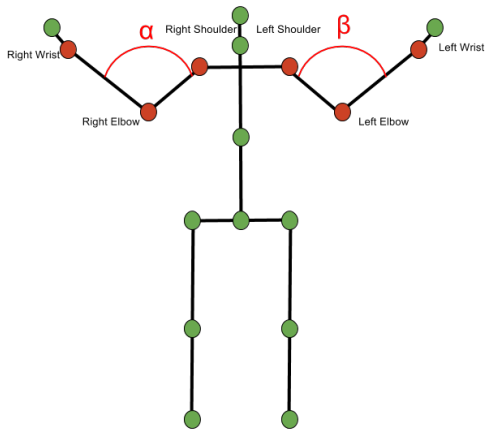


Figure 6.6: Angles α (right angle) and β (left angle) computed by the means of the wrists, elbows, and shoulders positions.

Finally, the `AngleBetweenTwoVectors()` method will return the angle of type *double*. Notice that the same principle is performed for both sides of the speaker's body.

6.2.3 Gesture Facts

As previously explained, the facts present in the rule engine's memory serve to store meaningful data obtained by the plug-ins. In the case of the gesture plug-in, the facts responsible for gesture detection should store the following variables:

- The *angle*'s amount (in degree) defining a valid gesture.

- The *time limit* (in seconds) in which gestures should occur. Whenever the speaker does not use gestures within the defined time limit, the rule engine should trigger a feedback message.

Notice that a third optional parameter *isActivated* defines whether the plug-in is activated or not, which could be necessary depending on the context of the presentation, for example, deactivating the eye contact plug-in depending on the culture of the audience.

When the gesture angle is computed (for both sides) by the `AngleBetweenTwoVectors()` method, the `processAngle()` method updates the angles in the subscribed facts (see Listing 6.8). To do so, the `gestureFact.feedbackFromPlugins()` method imposed by the **IFact** interface ensures the implementation of this method, making it callable by the gesture plug-in.

Listing 6.8: Updating the computed angles in the facts.

```
private void processAngle(double angleL, double angleR)
{
 object[] param = new object[2] { angleL, angleR };

 foreach (IFact fact in getFacts())
 {
  fact.feedbackFromPlugins(param);
 }

 updateFact();
}
```

The `updateFact()` method will ask the rule engine to modify the angles of the facts in the memory, and fire the rules, in order to trigger feedback messages if needed.

6.2.4 Gesture Rules

The gesture rules, used for both sides of the speaker's body, implemented in the rule engine are responsible for updating variables from the gesture fact and to trigger feedback messages to the speaker whenever needed. In this case, the rule engine will be provided with a continuous stream of angles computed by the gesture plug-in in order to detect valid increasing and decreasing gestures defined by the user. To perform this, the rules were divided in three parts:

- **Detector Rules** responsible for detecting valid gestures.
- **Update Rules** responsible for updating variable from the gesture fact. They also serve to keep a short history of angles in order to allow the Detector Rules to detect gestures.
- **Feedback Rule** responsible to alert the speaker with feedback messages.

Gesture Detector Rules

The rules **isIncreasingGesture** and **isDecreasingGesture** are responsible for detecting gestures whenever one of the following rules are satisfied:

$$\mid firstIncreasedAngle - currentAngle \mid \geq gestureAngle \tag{6.4}$$

where the *firstIncreasedAngle* was the first detected increasing angle, or by

$$\mid firstDecreasedAngle - currentAngle \mid \geq gestureAngle \tag{6.5}$$

where the *firstDecreasedAngle* was the first detected decreasing angle. Whenever the absolute value of *firstIncreasedAngle*, or *firstDecreasedAngle*, subtracted with the current computed angle is equal or greater than the *gestureAngle*, which is the angle defining a valid gesture by the user, the `isIncreasingGesture()` or `isDecreasingGesture()` method will be called by one of the two rules in order to modify the *datetime* of the last detected gesture, or in other words, tell to the rule engine that the speaker is using gestures and no feedback messages should be triggered. Notice that the same principle is used for both sides of the speaker's body.

Gesture Updater Rules

The angle computed by the gesture plug-in may follow the flow of an increasing or decreasing gesture. Since they are not yet recognised by the rule engine as being valid gestures defined by the user, they nevertheless still posses useful information that should be stored in order to allow the rule engine to "predict" whether the gesture will be an increasing and decreasing one. Detecting the first potential increasing or decreasing gesture, and updating the previous angle is done as follows:

The **IncreasingAngleRule** and **DecreasingAngleRule** rules define whether the gesture will be an increasing or decreasing one. Consider the *firstIncreasingAngle* to be the *currentAngle* if and only if the *previousAngle* is less or equal than the *currentAngle*, and the *firstIncreasingAngle* should be empty, stating that no *firstIncreasingAngle* has been detected before.

$$\begin{aligned} firstIncreaseAngle = currentAngle \Leftrightarrow \\ [(firstIncreaseAngle == null) \wedge (previousAngle \leq currentAngle)] \end{aligned} \tag{6.6}$$

When the **IncreasingAngleRule** "predicts" the current gesture to be an increasing angle, the `isPossibleIncreasingGesture()` method available in

the **GestureFact** class will be called by the rule in order to update the first increasing angle *firstIncrAngle*, which will be useful for determining future gestures by the gesture detectors. The same principle is applied for detecting a decreased gesture, with the exception that the *previousAngle* should be greater or equal to the *currentAngle*.

Gesture Feedback Rule

The **FeedbackGestureRule** is responsible for triggering feedback messages to the speaker whenever the use of gestures was not sufficient enough. When no valid gestures have been detected by the rule engine during a certain amount of time defined by the user, a feedback message will be fired. Therefore the following rule should be satisfied:

$$[t(lastDetectedGesture) < t(currentComputedAngle - timeLimit)] \Rightarrow fireFeedback \tag{6.7}$$

When the time t of the *lastDetectedGesture* is lower than the time of the *currentComputedAngle* subtracted by the feedback time limit *timeLimit* defined by the user, a feedback message will be triggered to the speaker, in order to adapt their behaviour by using more gestures during the speech.

6.3 Sightline Plug-in

Eye contact during presentation has proven to increase the credibility of the delivered message and to give a certain degree of importance to the audience. Even though it is not appreciated in certain cultures, it still remains an important nonverbal communication skill. Unfortunately, performing eye-tracking for determining the sightline of the speaker is practically not possible with the Microsoft Kinect, since the sensor is designed to use between one to four meters, making it unsuitable for acquiring detailed information about the eyes. Nevertheless, performing face-tracking is possible with the **Face Tracking SDK** in combination with the Kinect SDK, allowing to track the speaker's head position and to deduce facial expressions [129]. By including the face-tracking SDK in the project, *PRESMA* is able to capture the *head pose angles* (see Figure 6.7) from the tracked speaker, which are:

- The speaker's head **Pitch** determines whether the speaker is looking downwards or upwards. The values, expressed in degrees, range from -

90 to +90, where 0 is neutral. Nevertheless, the tracking work whenever the head's pitch is less than 20 degrees.

- The speaker's head **Yaw** angle, also expressed in degrees, allows to determine whether the speaker's head is turned towards the left or right shoulder. The values range from -90 to +90, where 0 is neutral. Similar to the pitch angle, tracking the speaker's head yaw works best when it is less than 45 degrees.

- **Roll**: The speaker's head **Roll** determine whether the speaker's head is horizontal parallel with one of its two shoulders with values ranging between -90 and +90, where 0 is neutral.

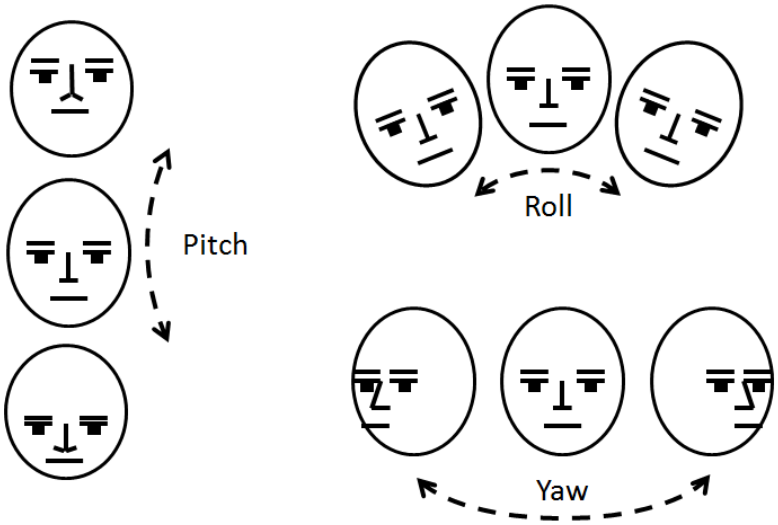


Figure 6.7: The speaker's head pose angles (yaw, pitch, and roll) [129].

By allowing the pitch and yaw angles to be configured manually by the sightline plug-in, the speaker is able to adapt its visual axis, as presented in Figure 6.8, depending on the presentation's context. Moreover, this method allows also to adapt the speaker's vision depending on obstacles present in the presentation room, as shown in Figure 6.9. The idea behind the sight plug-in is to allow the speaker to graphically define the range of the yaw and pitch angles. Whenever the speaker does not look to the audience during a defined time, by having a yaw and pitch angle exceeding the pre-defined range, the plug-in should trigger a feedback message by the intermediary of the rule engine. Therefore, the rule engine will be provided with a stream of yaw and pitch angles.

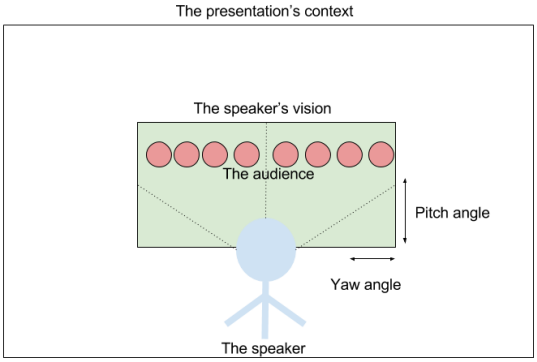


Figure 6.8: Adapting the angles in function of the presentation's context.

6.3.1 Importing the Sightline Plug-in

The **SightPlugin** class is able to be loaded by the **PluginLoader** class, since it implements the **IPlugin** contract, making it a valid plugin, and declares being an *Export* of type **IPlugin** (see Listing 6.9). The working of this plug-in is ensured by the contract, since it forces the plug-in to implement the methods listed in Code 5.6.

Listing 6.9: The sight plug-in as a valid plug-in.

```
[Export(typeof(IPlugin))]
public class SightPlugin : IPlugin
{
 //...
}
```

6.3.2 Computing the Head Pose Angles

In order to compute the angles needed to configure the speaker's sight, the **SightPlugin** class uses the **Microsoft.Kinect.Toolkit.FaceTracking** project in order to perform the face-tracking. This report does not explain in detail how the **FaceTracking** project works, which is well explained in [129], but only how to acquire the head pose angles.

In order to make the **FaceTracking** work, the **SightPlugin** class needs the skeleton frame, depth image frame, and the colour image frame from the Kinect sensor. Since the **SightPlugin** class implements the `getAllFrames()`

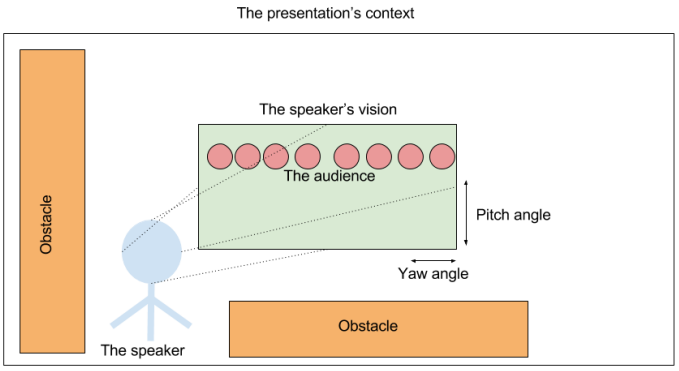


Figure 6.9: Adapting the angles in function of the obstacles.

method, it succeeds the Kinect's compatibility test, making it subscribed to the *AllFramesReady* event. The reference to the Kinect sensor is also passed to the **FaceTracking** project for configuration purposes. Whenever the frames are received from the Kinect sensor, the `getTrackedUser()` method tests whether all the frames contain effectively data, since they are all important for the face-tracking, before copying them locally (see Listing 6.10).

Listing 6.10: Acquiring all the frames from the input modality.

```
public void getAllFrames(KinectSensor sensor, SkeletonFrame skeletonFrame, DepthImageFrame
     depthFrame, ColorImageFrame colorFrame)
{
 his.sensor = sensor;
 getTrackedUser(skeletonFrame, depthFrame, colorFrame);
}
```

Succeeding all the tests in the `getTrackedUser()` method makes it possible to look for the tracked users in the *skeletonData* array with a *SkeletonTrackingState.Tracked* as tracking state. Such kind of skeletons provide detailed information about the position of twenty user's body joints. A user with a *SkeletonTrackingState.PositionOnly* as tracking state has information about the user's position, but not about the body joints. The *trackedSkeletons* dictionary serves to keep a record of any skeletons, whether tracked or not. Since each user has its own *skeletonFaceTracker* object, used to perform face-tracking, it is important to use the right object, in other words, the speaker's *skeletonFaceTracker* object containing precious data for the **FaceTracking** project. The Kinect identifies every tracked user with an

ID, accessible by the means of the *TrackingID* member. Therefore, the speaker's *skeletonFaceTracker* object becomes accessible, allowing to call the `skeletonFaceTracker.OnFrameReady()` method in order to update each frame obtained by the Kinect.

When the speaker is identified, and the frames updated, the *OnFrameReady* method from the **SkeletonFaceTracker** class is able to acquire the yaw and pitch angles, by accessing the positions of the limbs. Therefore, a user with a tracked *TrackingState* is necessary. Succeeding the test checking whether the speaker's tracking state is indeed *Tracked*, make it possible to try initialising the *faceTracker* object, which will track faces.

The last step consists of providing the `Track()` method from the face-tracking project with the freshly updated frames received from the Kinect sensor and with the skeleton data of the speaker. This will return a frame allowing to determine whether the tracking has been successful or not, and to acquire the yaw and pitch angles from the speaker's face. The stored angles in the *faceOrientations* array will be used to update the facts for the rule engine by the means of the `updateFact()` method (see Listing 6.11).

Listing 6.11: Updating the orientations in the fact.

```
if (frame.TrackSuccessful)
{
 //Yaw & Pitch

 if (faceOrientations == null)
  faceOrientations = new float[2] { frame.Rotation.X, frame.Rotation.Y };

 updateFact();

}
```

6.3.3 The Sightline Fact

The **SightFact** is responsible for storing the stream of pitch and yaw angles acquired by the **SightPlugin**, making it possible for the rule engine to fire rules. Whenever the sight plug-in is configured by the speaker, the constructor of the **SightFact** class is called in order to initialise the following variables:

- The *minYaw* and *maxYaw* defining the minimum and maximum of the yaw's range.
- The *minPitch* and *maxPitch* defining the minimum and maximum of the pitch's range.
- The *feedbackTime* variable holds the time limit (expressed in seconds) in which the speaker is authorised to not look to the audience. When

the speaker's pitch or yaw exceed the defined ranges during the time limit, the rule engine will trigger a feedback.

- The boolean *isActivated* serves to activate or deactivate the plug-in during a session.

Since the sight fact is forced, by the **IPlugin** contract, to implement the `feedbackFromPlugin()` method, this one will be used by the `updateFact()` method from the **SightPlugin** class whenever the angles need to be updated.

6.3.4 The Sight Rules

The sight rules implemented in the rule engine are responsible for updating variables from the sight fact and to trigger feedback messages to the speaker whenever needed. Therefore, a continuous stream of yaw and pitch angles will be provided to the rule engine. To perform this, the rules were divided in two parts:

- **Update Rules** responsible for updating variable from the sight fact.
- **Feedback Rule** responsible to alert the speaker with feedback messages.

The Updater Rules

The **SightPitchUpdate** and **SightPitchUpdate** rules responsible for updating the actual computed yaw and pitch angles work similarly. Therefore, only one of them will be explained, which is the **SightPitchUpdate** rule. When the sight plug-in updates the yaw and pitch angles in the sight fact, the rule engine needs to be informed by calling its `updateFact()` method, in order to perform the changes in the engine's memory. Whenever this is done, the rule engine will try to fire rules satisfying the data contained in the fact. The **SightPitchUpdate** rule is responsible for updating the *datetime* of the last valid received pitch, in other words, whenever the *actualPitch* lays between the range of pitch defined by the user, the *lastDetectedPitch*'s time will be updated.

$$[(minPitch \leq actualPitch) \wedge (actualPitch \leq maxPitch)] \Rightarrow update(lastDetectedPitch) \tag{6.8}$$

The rule will call the `pitchDetected()` method from the fact in order to update the last detected pitch variable, which will avoid the rule engine to trigger a feedback message to the speaker.

The Feedback Rule

The *SightFeedbackRule* rule is really simple, since it uses only the *datetime* of the last detected pitch and yaw angles considered to be valid ones. What this rule basically does is to check whether the last detect pitch, or yaw, has exceeded a certain time duration defined by the user. If this becomes the case, a feedback message will be triggered, informing to speaker to pay attention for the audience. The rule works as follows:

Whenever the time t of the *currentPitch* subtracted with the defined *feedbackTime* duration, defined by the user, is greater than the *lastDetectedPitch*, means that the speaker has not been looking for the audience during the defined limit *feedbackTime*. The rule will alert the speaker by calling the **SightFact**'s method `fireFeedback()`.

$$[t(lastDetectedPitch) < (t(currentPitch) - feedbackTime)] \Rightarrow fireFeedback \tag{6.9}$$

Notice that an *Or* statement is to be found in the *SightFeedbackRule*, meaning that the same concept is also applicable for the yaw angle.

6.3.5 Limitations

Testing the sight plug-in has shown its limitations, since the **FaceTracker** third-party project is not always able to track the speaker's head. Moreover, when the yaw, or pitch angles exceed a certain amount of degrees, the Kinect sensor is no longer able the track the face correctly, since the orientation of the face hides certain important facial elements used to compute the angles.

Therefore, a second approach is implemented, which uses the distance from the Kinect to the shoulders to determine whether the speaker's body is oriented towards the audience. As shown in Figure 6.10, when the difference between d_1, being the distance between the speaker's right shoulder and the Kinect, and d_2, being the distance between the speaker's left shoulder and the Kinect, exceeds a certain defined threshold, defined by the user, means that the speaker's body is not oriented towards the audience. The rule engine will trigger a feedback message to the speaker, stating to look at the audience. Note that this method is not always accurate, since the body of the speaker could be oriented towards the audience while the speaker is not looking at the audience.

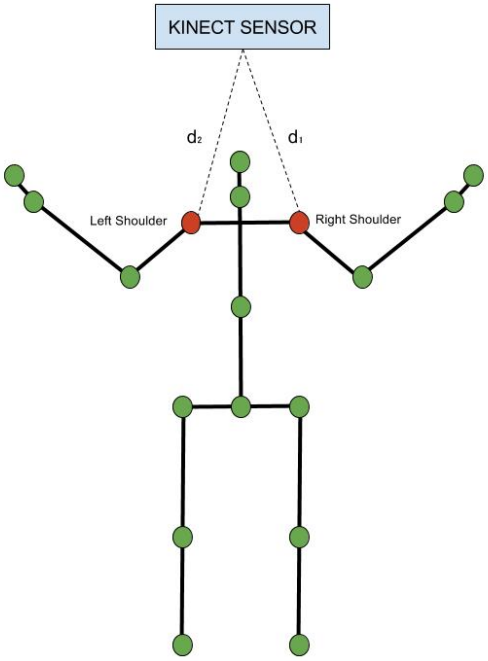


Figure 6.10: A second approach for measuring eye-contact.

6.4 Speech Analysis

The **SpeechRatePlugin** and **LoudnessPlugin** are both plug-ins used to perform speech analysis on the sound captured by the Kinect sensor. One is responsible for the computing the speech rate of the speaker, while the other one the loudness, or voice volume. PRAAT [106] is a light-weight and flexible tool offering all the needed procedures to measure different properties of sound. Similar to RHEMA [3], *PRESMA* uses PRAAT to measure the loudness and compute the speech rate of the speaker. Since both plug-ins share some similarities, common subjects will be explained together, knowing that they are implemented in different classes.

6.4.1 Importing the Speech Rate and Loudness Plug-ins

Both **SpeechRatePlugin** and **LoudnessPlugin** classes are able to be loaded by the **PluginLoader** class, since they both implement the **IPlugin** contract, making them valid plug-ins, and declare being *Export*s of type **IPlugin** (see Listing 6.12 and 6.13). The working of these plug-ins are ensured by the contract, since they are forced to implement the methods listed in Code 5.6.

Listing 6.12: The loudness plug-in as a valid plug-in.

```
[Export(typeof(IPlugin))]
public class LoudnessPlugin : IPlugin
{
 //...
}
```

Listing 6.13: The speech rate plug-in as a valid plug-in.

```
[Export(typeof(IPlugin))]
public class SpeechRatePlugin : IPlugin
{
 //...
}
```

6.4.2 Acquiring Audio Data

Both plug-ins are considered to be compatible with the Kinect sensor, since they both implement the `getAudio()` method, needed to subscribe to the stream of audio data. When the Kinect has read audio data enough to fill the *audioBuffer*, both **SpeechRatePlugin** and **LoudnessPlugin** will benefit from it, making them able to perform some audio analysis. Since PRAAT is an external application, it is not possible to redirect the audio stream to it in real-time. The only possible way is to save the audio data into separate samples, identical to how RHEMA [3] proceeds, in order to allow PRAAT to analyse the audio samples. In order to do so, the user of *PRESMA* is engaged to define the time of the audio samples, which will serve to:

1. Fill the audio sample files with a limit, expressed in seconds, defined by the user.

2. Define the reactiveness of the feedback messages. In other words, if the samples are five seconds long, it will take at least five seconds before a possible feedback message will be triggered by the rule engine.

Whenever the samples are ready, the second step consists of analysing them, by calling PRAAT in a different thread. Avoiding the main application thread to perform this, will make *PRESMA* more reactive, especially the user interface. The *sampleAnalyzer* thread will use either the `computeLoudness()`, or the `computeSpeechRate()` method (depending on the plug-in), to start the PRAAT process with the following parameters:

- The **SpeechRatePlugin** calls PRAAT with the script used to compute the *pitch*, and the sample file with its name ending by *"_speechRate_wav.WAV"*.

- The **LoudnessPlugin** calls PRAAT with the script used to compute the *intensity*, and the sample file with its name ending by *"_loudness_wav.WAV"*.

The *sampleAnalyzer* thread will wait until PRAAT has finished to analyse the samples, before giving the authorisation to the plug-ins to open the files containing the analysis.

6.4.3 Computing the Speech Rate

PRESMA uses the same script, provided by RHEMA [3], to get a rough estimation of the speech rate. Therefore, the *SpeechRate* plug-in calls PRAAT with the *pitch* script in order to use its Pitch Detection Algorithm on the audio sample. The algorithm provides a file containing all the pitch values, making it possible to "draw" the pitch contour (as shown in Figure 6.11), while being able to discern the voiced from the unvoiced areas (as shown in Figure 6.12).

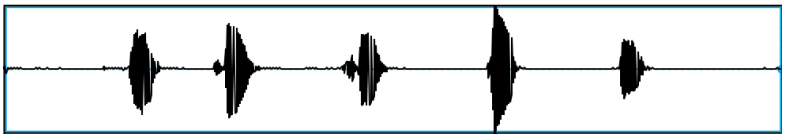


Figure 6.11: The pitch contour of a sample in PRAAT [106].

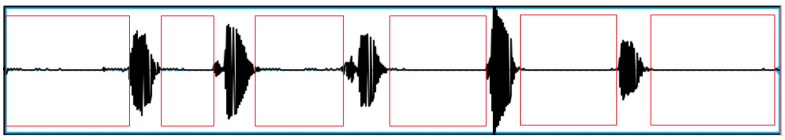


Figure 6.12: The pitch contour of a sample in PRAAT [106] with the unvoiced areas (in red rectangles).

The `analyzePitch()` method provides an estimation of the speech rate by counting the number of times the values go from an voiced to an unvoiced area, in other words, counting the number of discontinuities. The *speechRate* variable, containing the speech rate of the current analysed sample, is then sent to the **SpeechRateFact**, by the intermediary of the implemented `feedbackFromPlugins()` method. Calling the `re.updateFact()` method from the rule engine will perform the changes in the engine's memory, used to fire the satisfied rules (see Listing 6.14).

Listing 6.14: Update the speech rate in the facts.

```
private void updateFact(int speechRate)
{
 object[] parameters = new object[1] { speechRate };

 foreach (IFact fact in getFacts())
 {
  fact.feedbackFromPlugins(parameters);
 }

 foreach (IFact fact in getFacts())
 {
  re.updateFact(fact);
 }
}
```

The Speech Rate Fact

The constructor of the **SpeechRateFact** has as parameters the minimum speech rate and maximum speech rate, defined by the user, in which the estimated speech rate should lay. The *isActivated* boolean serves to activate or not the plug-in. Whenever the `feedbackFromPlugins()` is called by the plug-in with the actual estimated speech rate, the *speechRate* variable will be used as a placeholder for the rules, responsible for the triggering of feedback messages to the speaker.

The Speech Rate Rule

Triggering feedback messages to the speaker is really simple, since the only thing that the **SpeechRateRule** has to do is to check whether the estimated speech rate in the fact lays between the defined range by the user. Therefore the following rule is used:

Whenever the actual estimated speech rate *actualSpeechRate* is lower than the minimum speech rate *minSpeechRate*, or, greater than the maximum speech rate *maxSpeechRate*, the rule engine will trigger a feedback message to the speaker (*fireFeedback*).

$$(actualSpeechRate < minSpeechRate) \Rightarrow fireFeedback \tag{6.10}$$

$$(actualSpeechRate > maxSpeechRate) \Rightarrow fireFeedback \tag{6.11}$$

6.4.4 Computing the Loudness

Computing the loudness of an audio sample is similar as computing the speech rate. Calling PRAAT with the *intensity* script provided by RHEMA [3],

will calculate the loudness, in decibel, of each sound frame from a specific given audio sample. Afterwards PRAAT provides a file containing the loudness of each frame, making it possible to calculate the average loudness, used to trigger feedback messages when needed.

The `analyzeLoudness()` method will be called whenever PRAAT has finished processing the audio sample, which will open the file containing the analysis to calculate the average loudness. Afterwards, the *averageLoudness* will be sent to the fact, which will be updated in the rule engine's memory.

6.4.5 The Loudness Fact

The constructor of the **SpeechRateFact** has as parameters the minimum and maximum loudness, defined by the user, in which the calculated average loudness should lay. The *isActivated* boolean serves to activate or not the plug-in. Whenever the `feedbackFromPlugins()` is called by the plug-in with the actual average loudness, the *avgLoudness* variable will be used as a placeholder for the rule, responsible for the triggering of feedback messages to the speaker.

6.4.6 The Loudness Rule

The **LoudnessRule** rule works similarly to the **SpeechRateRule**, since it only checks whether the average loudness lays between the range defined by the user. If it is not the case, the rule will trigger a feedback message to the speaker. The following rule determines whether the speaker should speak louder or not:

Whenever the actual average loudness is lower than the minimum loudness *minLoudness*, or, greater than the maximum loudness *maxLoudness*, the rule will trigger a feedback message to the speaker (fireFeedback).

$$(actualAvgLoudness < minLoudness) \Rightarrow fireFeedback \tag{6.12}$$

$$(actualAvgLoudness > maxLoudness) \Rightarrow fireFeedback \tag{6.13}$$

6.5 The WPF Window Output Modality

Output modalities are used to present feedback messages to the speaker in a particular form (e.g. haptic feedback, visual feedback, and so forth). In the case of the *PRESMA*, a simple Windows Presentation Foundation (WPF) window is used, showing informative feedback messages. In order to avoid cognitive overload, the window is designed to present messages up to two words. Nevertheless, the system is not limited to GUI windows, but supports different kinds of output modalities thanks to its extensibility. Therefore, each valid output modality should implement the **IOutputModality** contract and declare the *Export* attribute (see Listing 6.15).

Listing 6.15: The WPF window as a valid output modality.

```
[Export(typeof(IOutputModality))]
public partial class frmComputerDisplayModality : Window, IOutputModality
{
 //...
}
```

The `getFeedbackMessage()` method allows the window output modality to acquire feedback messages from the application controller, which are stored in a *queue*. A thread, responsible for continuously reading the queue, is used to present one feedback message at the time. Since, the possibility that multiple feedback messages are triggered consecutively remains, the thread is designed to wait a certain amount of time, defined by the user, before displaying the next feedback message, in order to avoid cognitive overload and to put the speaker on the defence.

7 Use Case

The previous chapters explained the used technologies making the implementation of *PRESMA* possible, going from the architecture to the inclusion of input modalities, output modalities and plug-ins. Since the extensible parts are all designed in a way allowing the speaker to customise a presentation session based on his needs, this chapter walks through a scenario explaining how the GUI should be used to configure the tool. The GUI has been kept simple, since the main goal of this thesis was to build an extensible system capable of providing feedback messages to the speaker.

7.1 The Main Window

The main window of *PRESMA*'s GUI serves mainly for configuration purposes. As stated before, it is important that the speaker should be able to configure the system depending on the context and needs. As presented in Figure 7.1, the main window consists of three important functionalities, which should be triggered in the following chronological order:

1. The **Configure Plug-ins** phase consists of subscribing the loaded plug-ins to the available input modalities by the means of the *Linker* button.

2. The **Configure Facts** phase allows to load different rule sets and to subscribe one or multiple facts (detection logics) to a specific plug-in by the means of the *Facts Linker* button.

3. The **Configure Output Modalities** phase consists of configuring the output modalities used to present the feedback messages to the speaker by the means of the *Configurator* button.

When all the previous steps have been achieved, the system will be able to run properly, by the means of the **start** button. On the other hand, the **stop** button serves to stop a session correctly.

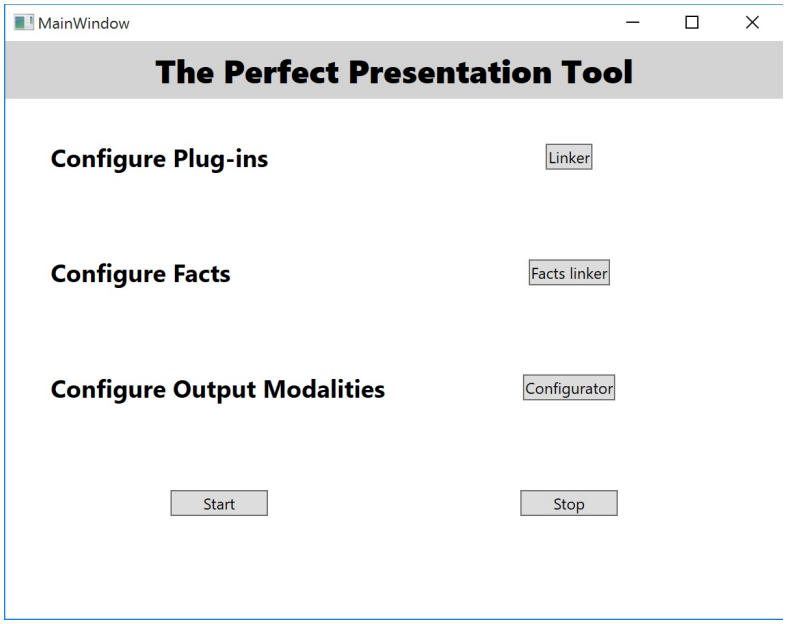


Figure 7.1: The main window of *PRESMA*'s GUI.

7.2 Configuring the Plug-ins

The first configuration step in *PRESMA* consists of redirecting the streams of data generated by the input modalities to the plug-ins. Since in this scenario only the *gesture*, *sightline*, *loudness*, *speech rate* and *eye-contact* plug-ins are available, it is important that the system should be configured as presented in Figure 7.2, in order to allow them to detect, or measure, nonverbal cues.

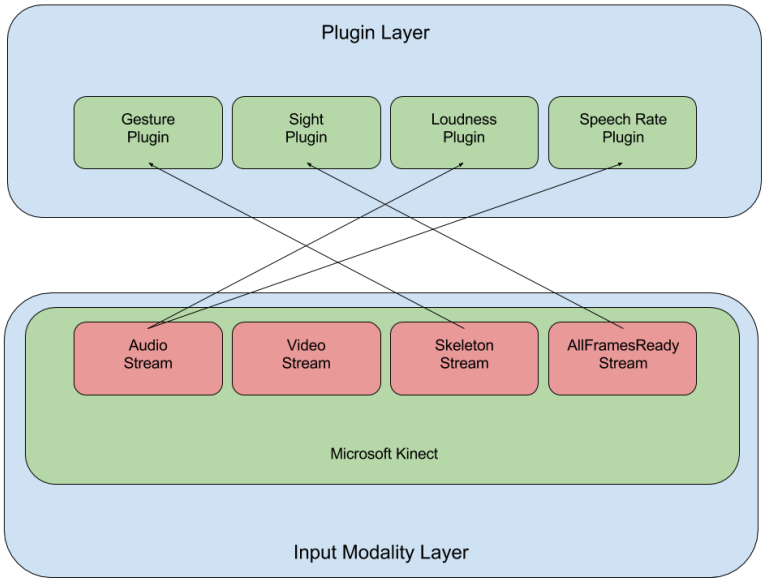


Figure 7.2: Redirection of the streams of data from the Microsoft Kinect to the plug-ins.

7.2.1 Subscribing Plug-ins to Input Modalities

Whenever the *Linker* button is clicked by the user, a new dynamically created window is automatically opened, allowing the user to redirect the streams of data generated by the available input modalities to the loaded plug-ins. As presented in Figure 7.3, the window is divided in two columns, the left side represents the loaded plug-ins, while the right side represents the included input modalities.

The *comboboxes* next to each plug-in are populated with the available input modalities. Selecting one of the list, as presented in Figure 7.4, will associate the plug-in to it only if both are compatible. Since in this scenario the Microsoft Kinect is the only available input modality, each combobox will only consist of this one.

Finally, clicking on the *Link* button (see Figure 7.3) will finish the subscription process.

7.3 Configuring the Facts

The main goals of a fact is to store the data obtained by the plug-ins and fire feedback messages when certain rules are satisfied. Since the platform supports rule sets to be imported, as presented in Figure 7.5, the user is

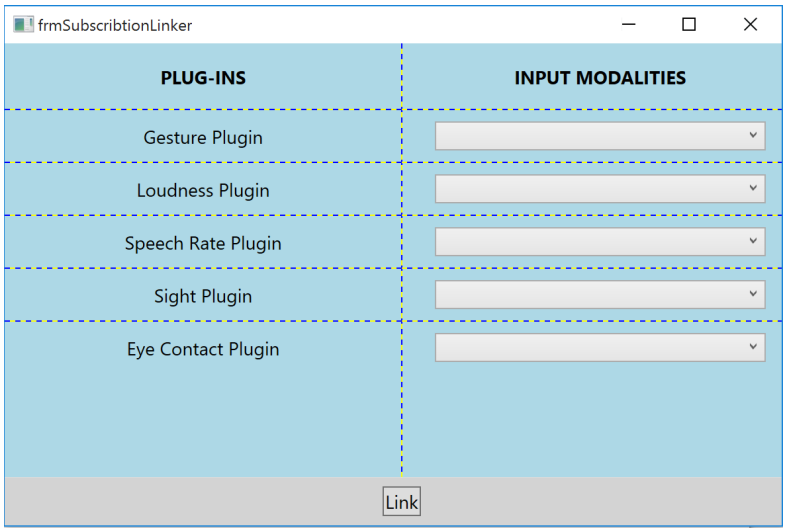


Figure 7.3: The plug-in subscription linker window.

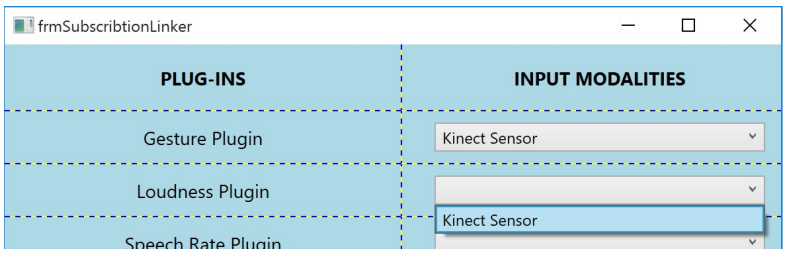


Figure 7.4: Subscribing a plug-in to a specific input modality.

responsible for selecting the set of rules that will be used during a presentation session, for example, a set of rules destined for business purposes, educational purposes, and so forth. The type of the selected ruleset will be used to look for all the facts and rules having the same type as parent (super class). Afterwards, as shown in Figure 7.6, the facts should be added to the corresponding plug-in, in order to benefit from the stream of data. Clicking on the *Link* button will finish the configuration process and allow the user to configure the thresholds of each plug-in, which will be stored in the facts for the rules.

7.3.1 Configurator Pages

The plug-in contract **IPlugin** forces the plug-ins to implement the method `getConfiguratorPage()`, which is a WPF page used to configure the plug-in thresholds through a GUI. After the subscription process, *PRESMA* will

Figure 7.5: Selecting rulesets for a presentation session.

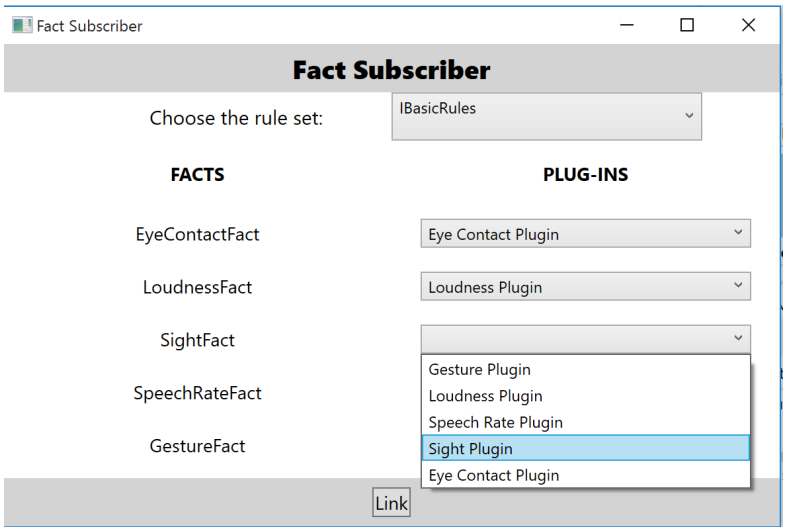


Figure 7.6: Configuring the facts from a specific ruleset.

get all the pages from the loaded plug-ins and show them to the user. The entered values are vital, since they serve to parametrise the facts used by the rule engine in order to trigger feedback messages to the speaker.

Gesture Plug-in Configurator

The gesture plug-in determines whether the speaker is indeed using gestures during the presentation or not. Therefore, as presented in Figure 7.7, the user should enter two important values, namely:

- The `Angle` value, expressed in degree, defines the value of a valid gesture. In this scenario, a valid gesture is a gesture with an increasing or decreasing angle of minimum 15 degrees.

- The `Time Feedback` value, expressed in seconds, is the time limit in which the speaker is able to not use gestures. In other words, whenever

the time of the last detected gesture exceeds 10 seconds, the rule engine will trigger a feedback message stating to use gestures.

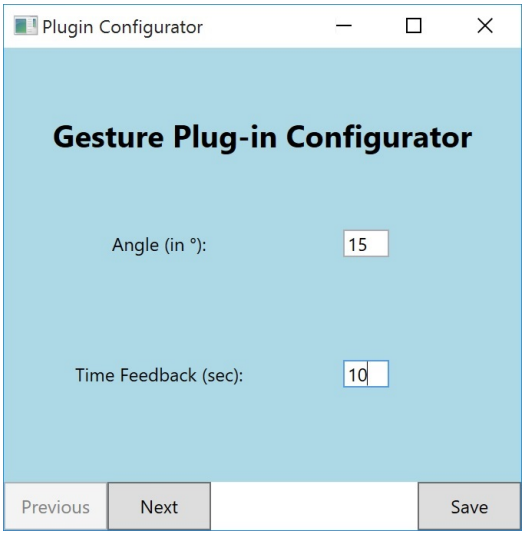


Figure 7.7: The gesture plug-in configurator.

Sightline Plug-in Configurator

The sightline plug-in serves to determine whether the speaker is indeed paying enough attention to the audience or not by the means of the speaker's face orientation. By defining the pitch and yaw angles, the system is able to determine where the speaker is looking at. As presented in Figure 7.8, the user is able to configure the plug-in as follows:

- The `Time Feedback` value, expressed in seconds, is the time limit in which the speaker is allowed to not look at the audience. In other words, whenever the speaker does not look to the audience during 10 seconds (in this scenario), the rule engine will trigger a feedback message.

- The `Min.` and `Max. Pitch`, and the `Min.` and `Max. Yaw`, all expressed in degrees, serves to define the area in which the speaker's visual axis, or sightline, should be situated[1].

[1]For a better understanding of these values, the face-tracking tutorial by Microsoft can serve as a basis: `https://msdn.microsoft.com/en-us/library/jj130970.aspx`

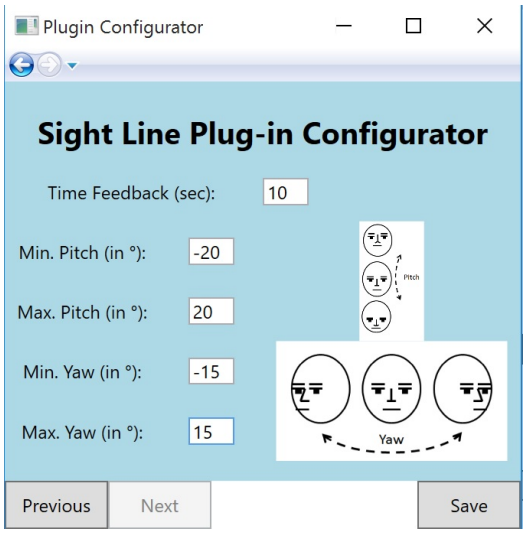


Figure 7.8: The sightline plug-in configurator.

Loudness Plug-in Configurator

The loudness plug-in serves to determine whether the speaker's message can be understood clearly by the audience by speaking loud enough. The plug-in will compute the average loudness, which will allow the rule engine to trigger feedback messages whenever needed. Therefore, as presented in Figure 7.9, the user needs to fill the following thresholds:

- The `Sample Time` value, expressed in seconds, defines the length of the audio samples used to compute the average loudness of the speaker. In this case, the average loudness will be computed every 5 seconds by the system. Entering a low value is recommended, since it will make the system more reactive.

- The `Min.` and `Max.` `Loudness Filters`, expressed in decibel (dB), are thresholds used to filter out the noise present in the environment. In this scenario, the system will not take in consideration loudnesses lower than 30 dB and greater than 80 dB.

- The `Min.` and `Max.` `Loudness`, expressed in decibel (dB), defines what a valid loudness should look like. The system will trigger a feedback message whenever the computed average loudness does not lay between both values.

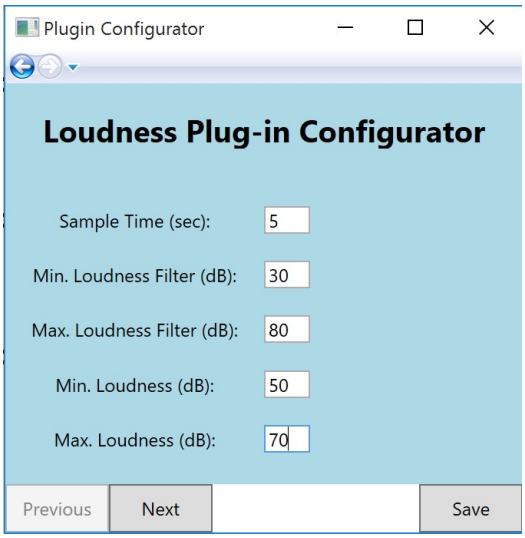


Figure 7.9: The loudness plug-in configurator.

Speech Rate Plug-in Configurator

The speaker's speech rate is computed by the speech rate plug-in. Being one of the most important nonverbal communication aspect, it is important that the speaker does not speak too fast or two slow. As presented in Figure 7.10, the user can configure the plug-in as follows:

- The `Sample Time` value, expressed in seconds, defines the length of the audio samples used to compute the speech rate of the speaker. In this case, the speech rate will be computed every 5 seconds by the system. Entering a low value will make the system more responsive regarding the speaker's speech rate.
- The `Min.` and `Max. Speech Rate`, expressed in number of words per time unit, where the time unit is in this case the sample time, define the boundaries of the speech rate. In this scenario, whenever the computed speech rate is lower than 5 words or higher than 15 words, the rule engine will trigger a feedback message. Notice that the methodology used to compute the speech rate is not accurate, therefore it is important to widen the wished boundaries with a few words.

Eye-Contact Plug-in Configurator

The eye-contact plug-in is a second approach used to determine whether the speaker is looking at the audience or not. As presented in Figure 7.11, the

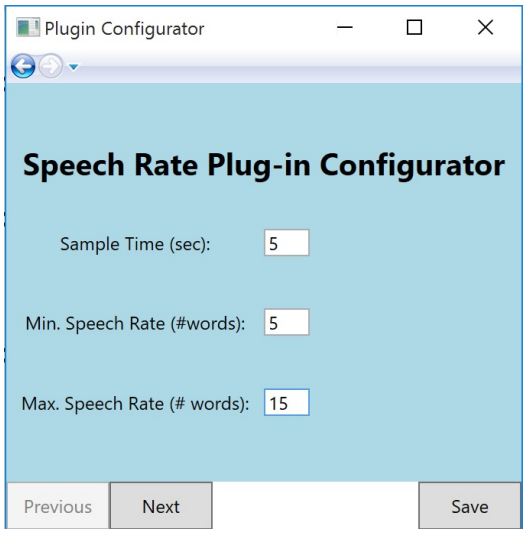


Figure 7.10: The speech rate plug-in configurator.

user can configure the plug-in as follows:

- The `Time Feedback` value, expressed in seconds, is the time limit in which the speaker is allowed to not look at the audience. In other words, whenever the speaker does not look to the audience during a certain defined time limit, the rule engine will trigger a feedback message.

- The `Distance Difference` value, expressed in centimetres, is used to determine whether the speaker's body is oriented towards the audience or not. Whenever the difference between the Kinect and both shoulders exceeds this value, means that the speaker is not paying attention to the audience.

Saving the Plug-in Configurations

Whenever all the plug-ins have been configured, clicking on the *Save* button available in the configurator window browser (see Figure 7.10) will call the `configurePlugin()` method available in each plug-in with their corresponding configuration page, which will allow them to acquire the values defined by the user and to configure the facts.

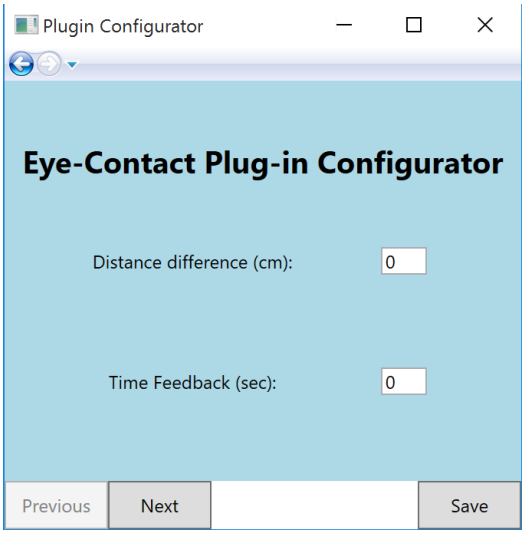


Figure 7.11: The eye-contact plug-in configurator.

7.4 Configuring the Output Modalities

Configuring the output modalities is practically the same as the methodology used for the plug-in configurators. As presented in Figure 7.1, the **Configure Output Modalities** will automatically open a window allowing the user to browse over all the configurator pages of the loaded output modalities in order to configure them.

7.4.1 The WPF Window as Output Modality

The only output modality used to demonstrate the working and effectiveness of *PRESMA* consists of a simple WPF window, which will be used to display informative feedback messages to the speaker. Before being able to use it, the window should first be configured. As presented in Figure 7.12, the configurator expects the following values to be defined by the user:

- The `Display Time`, expressed in seconds, defines the time in which a feedback message will be displayed in the WPF window. In this scenario, each triggered feedback message will be displayed for 5 seconds.
- The `Interval`, expressed in seconds, defines the time interval between feedback messages. In other words, the window will wait for 6 seconds before displaying the next feedback message. This feature is important, since it allows the system to not constantly distract, or cause cognitive overload, by displaying all the feedback messages in a row.

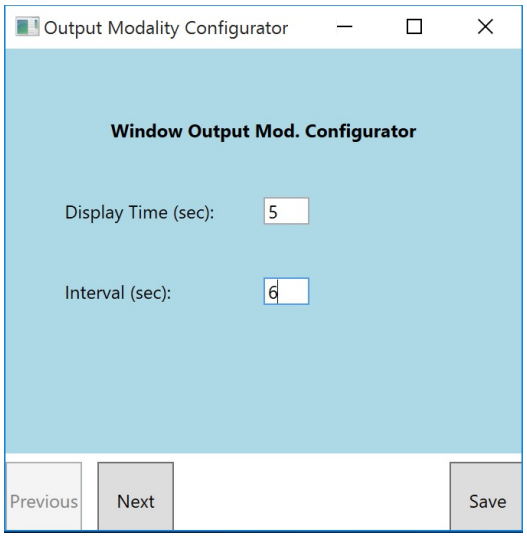


Figure 7.12: The WPF window output modality configurator.

The WPF window has two states, one with and one without feedback messages. The first state, as presented in Figure 7.13, is the one where no feedback message needs to be displayed. Two horizontal green bars are shown, stating that the speaker's behaviour is conform to the defined values during the configuration steps. On the other hand, as presented in Figure 7.14, a simple feedback message with maximum two words in combination with horizontal red bars are shown to the speaker. The reason behind displaying only two words is that it will not cause cognitive overload and allow the speaker to quickly adapt their behaviour by simply reading two words. The red horizontal bars show that a nonverbal communication aspect has not been respected.

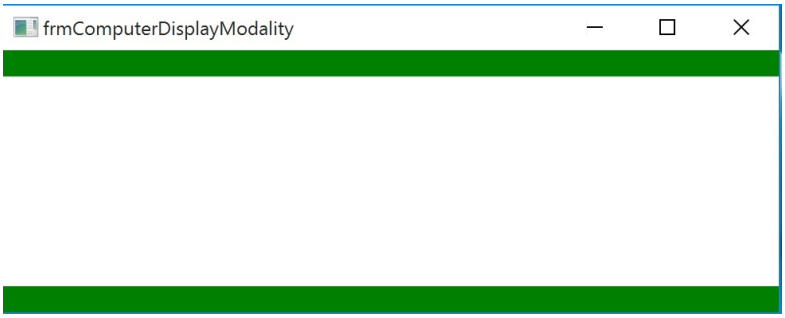


Figure 7.13: The normal state of the output modality.

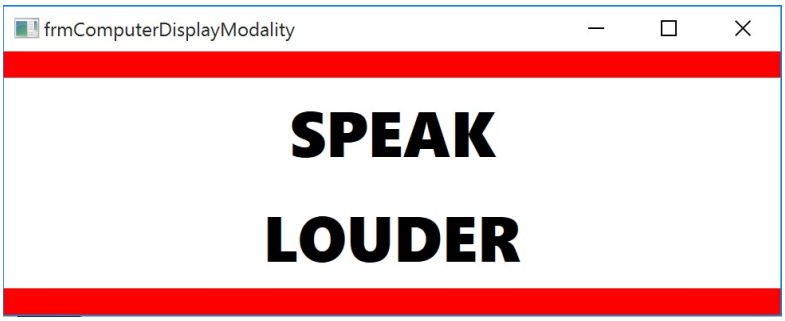


Figure 7.14: The feedback state of the output modality.

8

Conclusions and Future Work

In order to build *PRESMA*, an extensible multimodal system capable of providing informative feedback to the speaker in real-time during presentations, the first step was to investigate the communication process, allowing the identification of important elements present and needed while performing presentations. This led to the conclusion that there exist no "fixed" rules defining a good presentation, since not only verbal communication aspects should be taken in consideration, but also the speaker's nonverbal behaviours should be adequate to the presentation context. Presentation rules might be different depending on the culture of the audience, for example, the use of eye contact ratio with Asian audiences will differ from Western cultures. Moreover, the speaker's approach will be different for a business presentation than for an educational one. Therefore, it becomes important to identify the characteristics of the audience in advance.

Then, an investigation, analysis and comparison of existing multimodal feedback systems was conducted. The result of the comparison, based on important criteria, led to the conclusion that a need for extensibility was evident. Moreover, most of the analysed systems did not allow the inclusion of input and output modalities, or even identified a fixed set of communication aspects in advanced, which is contradictory with the conclusion obtained in the study of the communication process.

Afterwards, the findings from the conducted literature study enabled the implementation of *PRESMA*. Using the advantages and inconveniences of existing multimodal feedback systems as a strong basis, the design and implementation of an elaborated architecture allowing the system to be extensible by the means of plug-ins was possible. Each part of the system was designed in way to allow the speaker to adapt the system based on the needs and context of a presentation session.

Finally, based on important nonverbal communication aspects, a set of plug-ins were developed, serving to prove the working and effectiveness of the system. The imported plug-ins were able to measure the speaker's performances and to provide informative feedback, allowing the speaker to adapt his behaviour.

8.1 Contributions

The first contribution of this thesis is an overall study of Human-Machine Interaction, especially about interaction techniques and the impact of user interfaces on people's daily life. Based on their limitations, the field of multimodal systems has emerged, attracting the attention of a number of researchers. The literature provided about Multimodal User Interfaces (MUI) and Multimodal Interaction helps to understand multimodal systems and how using them for extracting meaningful information about users is feasible.

The second contribution is the study of the communication process. Since public speaking is a form of communication, identifying important communication aspects and communication elements was possible. These findings could serve as a basis for the preparation and rehearsal of presentations.

Finally, the last contribution is an extensible multimodal system capable of measuring the speaker's nonverbal cues to provide feedback messages, called PRESMA. Due to its elaborated architecture, users are free to extend the system's functionalities by means of plug-ins, bringing the system to another level. Compared to existing multimodal feedback systems [1, 2, 3, 4, 5, 6, 7, 8], *PRESMA* is the only one allowing the inclusion of presentation aspects, making it possible to customise the system according to the context and needs of a presentation session. Moreover, users are able to define their own thresholds used to trigger feedback messages, which were predefined in most of the discussed multimodal feedback systems [1, 3, 4, 5, 6, 7, 8]. Compared to [2, 4, 7, 8], *PRESMA* is able to assist the speaker during both real-time presentations and training situations due to its ability to include output modalities from different kinds, used to present the system in an

adequate form. In terms of input modalities, *PRESMA* allows the user to specify which ones to use during a presentation session. One can use the in-build components of a laptop for business purposes, while using more advanced ones for other purposes, which is not the case for [1, 3, 4, 5, 6, 8]. The user is provided with a less constraint feeling and is able to adapt the system according to their "ideal" perception of presentations, context and needs.

8.2 Future Work

PRESMA is mainly designed to tackle the gaps of existing multimodal feedback systems. However, the system's GUI is not as elaborated and friendly as it should be. Due to the limited time, the GUI focusses only on the configuration of the system, it should therefore be improved to allow the users to save their configurations for further use and to allow the testing of included plug-ins in order to be able to derive the ideal thresholds for a specific presentation. Also more logic should be implemented to automatically configure the system by the means of compatibility testing.

Another important aspect is to allow the users to visualise their presentation sessions. Currently, the system is able to save a session's speech and video data obtained from the Kinect from, which could be used in combination with annotations to show the speaker's presentation issues. Instead of showing statistics, the speakers would be able to visualise their performances and improve their behaviours for future presentations.

Up to now, the implemented plug-ins focusses only on nonverbal cues. Nevertheless, it is possible to introduce Microsoft PowerPoint into the system to detect whether keywords of a specific slide have been said by the speaker by simultaneously analysing audio data.

Finally, the system needs a user evaluation to assess the quality and efficiency for future refinements or improvements.

Bibliography

[1] K. Kurihara, M. Goto, J. Ogata, Y. Matsusaka, and T. Igarashi, "Presentation Sensei: A Presentation Training System Using Speech and Image Processing," in *Proceedings of the 9th international conference on Multimodal interfaces*, New York, USA, November 2007, pp. 358–365.

[2] J. Schneider, D. Börner, P. Van Rosmalen, and M. Specht, "Presentation Trainer: a Toolkit for Learning Non-Verbal Public Speaking Skills," in *European Conference on Technology Enhanced Learning*, Graz, Austria, September 2014, pp. 522–525.

[3] M. I. Tanveer, E. Lin, and M. E. Hoque, "Rhema: A Real-Time In-Situ Intelligent Interface to Help People with Public Speaking," in *Proceedings of the 20th International Conference on Intelligent User Interfaces*, Atlanta, USA, March 2015, pp. 286–295.

[4] S. Kopf, D. Schön, B. Guthier, R. Rietsche, and W. Effelsberg, "A Realtime Feedback System for Presentation Skills," in *EdMedia: World Conference on Educational Media and Technology*, Montréal, Canada, June 2015, pp. 1686–1693.

[5] A.-T. Nguyen, W. Chen, and M. Rauterberg, "Feedback System for Presenters Detects Nonverbal Expressions," *SPIE Newsroom*, 2012.

[6] ——, "Online Feedback System for Public Speakers," in *IEEE Symposium on E-Learning, E-Management and E-Services (IS3e)*, Kuala Lumpur, Malaysia, October 2012, pp. 1–5.

[7] T. Gan, Y. Wong, B. Mandal, V. Chandrasekhar, and M. S. Kankanhalli, "Multi-sensor Self-Quantification of Presentations," in *Proceedings of the 23rd ACM international conference on Multimedia*, Brisbane, Australia, October 2015, pp. 601–610.

[8] T. Wörtwein, M. Chollet, B. Schauerte, L.-P. Morency, R. Stiefelhagen, and S. Scherer, "Multimodal Public Speaking Performance Assessment," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, Seattle, USA, November 2015, pp. 43–50.

[9] P. A. DeCaro, "Origins of Public Speaking," in *The Public Speaking Project*, 2011, pp. 311–324.

[10] A. Nikitina, *Successful Public Speaking.* Bookboon, 2012.

[11] J. A. DeVito, *The Essential Elements of Public Speaking.* Allyn and Bacon, May 2002.

[12] J. Schneider, D. Börner, P. Van Rosmalen, and M. Specht, "Stand Tall and Raise your Voice! A Study on the Presentation Trainer," in *Design for Teaching and Learning in a Networked World.* Springer, 2015, pp. 311–324.

[13] J. Hattie and H. Timperley, "The Power of Feedback," *Review of educational research*, vol. 77, no. 1, pp. 81–112, March 2007.

[14] J. Schneider, D. Börner, P. Van Rosmalen, and M. Specht, "Augmenting the Senses: A Review on Sensor-Based Learning Support," *Sensors*, vol. 15, no. 2, pp. 4097–4133, February 2015.

[15] B. Dumas, D. Lalanne, and S. Oviatt, "Multimodal Interfaces: A Survey of Principles, Models and Frameworks," in *Human Machine Interaction.* Berlin, Heidelberg: Springer-Verlag, 2009, pp. 3–26.

[16] P. Booth, *An Introduction to Human-Computer Interaction (Psychology Revivals).* Psychology Press, September 2014.

[17] F. Karray, M. Alemzadeh, J. A. Saleh, and M. N. Arab, "Human-Computer Interaction: Overview on State of the Art," *International Journal on Smart Sensing and Intelligent Systems*, vol. 1, no. 1, pp. 137–159, March 2008.

[18] G. J. Kim, *Human-Computer Interaction: Fundamentals and Practice.* CRC Press, February 2015.

[19] P. M. Encyclopedia, "Definition of User Interface," accessed: 2016-05-08. [Online]. Available: http://www.pcmag.com/encyclopedia/term/53558/user-interface

[20] S. Oviatt and P. Cohen, "Perceptual User Interfaces: Multimodal Interfaces That Process What Comes Naturally," *Communications of the ACM*, vol. 43, no. 3, pp. 45–53, March 2000.

[21] A. Soriano, "Revolutionary User Interfaces," accessed: 2016-05-08. [Online]. Available: https://timeline.knightlab.com/examples/user-interface/

[22] D. Engelbart, "Revolutionary User Interfaces," accessed: 2016-05-08. [Online]. Available: http://www.foresightinhindsight.com/article/show/1164

[23] M. Turk and G. Robertson, "Perceptual User Interfaces (Introduction)," *Communication of the ACM*, vol. 43, no. 3, pp. 32–34, March 2000.

[24] E. S. Raymond and R. W. Landley, "Chapter 2. History: A Brief History of User Interfaces," accessed: 2016-05-08. [Online]. Available: http://www.catb.org/esr/writings/taouu/html/ch02.html

[25] A. Van Dam, "Post-WIMP User Interfaces," *Communications of the ACM*, vol. 40, no. 2, pp. 63–67, February 1997.

[26] V. Bush, "As We May Think," *SIGPC Notes*, vol. 1, no. 4, pp. 36–44, April 1979.

[27] T. Burghart, "Post-WIMP Interfaces," *Beyond the Desktop*, pp. 78–86, April 2013.

[28] J. Nielsen, "Noncommand User Interfaces," *Communications of the ACM*, vol. 36, no. 4, pp. 83–99, April 1993.

[29] R. J. K. Jacob, "A visual language for non-WIMP user interfaces," in *Visual Languages, 1996. Proceedings., IEEE Symposium on*, Medford, USA, September 1996, pp. 231–238.

[30] M. Green and R. Jacob, "SIGGRAPH 1990 Workshop Report: Software Architectures and Metaphors for non-WIMP User Interfaces," *SIGGRAPH Comput. Graph.*, vol. 25, no. 3, pp. 229–235, July 1991.

[31] R. J. Jacob, A. Girouard, L. M. Hirshfield, M. S. Horn, O. Shaer, E. T. Solovey, and J. Zigelbaum, "Reality-based Interaction: A Framework for post-WIMP Interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, USA, April 2008, pp. 201–210.

[32] H. Ishii, "Tangible User Interfaces," in *Human-computer Interaction: Design Issues, Solutions, and Applications*, D. Lalanne and J. Kohlas, Eds. New York, USA: CRC Press, 2007, pp. 469–487.

[33] M. Weiser, "Ubiquitous computing," *IEEE Computer*, vol. 26, no. 10, pp. 71–72, October 1993.

[34] D. Kirk, A. Sellen, S. Taylor, N. Villar, and S. Izadi, "Putting the Physical into the Digital: Issues in Designing Hybrid Interactive Surfaces," in *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*, Swinton, UK, September 2009, pp. 35–44.

[35] H. Ishii, D. Lakatos, L. Bonanni, and J.-B. Labrune, "Radical Atoms: Beyond Tangible Bits, Toward Transformable Materials," *Interactions*, vol. 19, no. 1, pp. 38–51, January 2012.

[36] D. Herzner, "Organic User Interfaces," *Beyond the Desktop*, pp. 9–14, April 2013.

[37] R. Vertegaal and I. Poupyrev, "Organic User Interfaces," *Communications of the ACM*, vol. 51, no. 6, pp. 26–30, June 2008.

[38] K. Lyytinen and Y. Yoo, "Ubiquitous Computing," *Communications of the ACM*, vol. 45, no. 12, pp. 63–96, December 2002.

[39] D. Mestre, P. Fuchs, A. Berthoz, and J. L. Vercher, "Immersion and Presence," *Le traité de la Réalité Virtuelle 3ème édition.*, vol. 1, pp. 309–338, March 2006.

[40] R. T. Azuma, "A Survey of Augmented Reality," *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 355–385, August 1997.

[41] N.-N. Zhou and Y.-L. Deng, "Virtual Reality: A State-of-the-Art Survey," *International Journal of Automation and Computing*, vol. 6, no. 4, pp. 319–325, November 2009.

[42] F. P. Brooks, "Grasping Reality Through Illusion–Interactive Graphics Serving Science," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, USA, June 1988, pp. 1–11.

[43] ——, "The Computer Scientist As Toolsmith II," *Communications of the ACM*, vol. 39, no. 3, pp. 61–68, March 1996.

[44] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Visual Displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, December 1994.

[45] H. Tamura, H. Yamamoto, and A. Katayama, "Mixed Reality: Future Dreams Seen at the Border Between Real and Virtual Worlds," *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 64–70, August 2001.

[46] M. Billinghurst and H. Kato, "Collaborative Mixed Reality," in *Proceedings of the International Symposium on Mixed Reality*, Yokohama, Japan, March 1999, pp. 261–284.

[47] M. Bajura, H. Fuchs, and R. Ohbuchi, "Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery Within the Patient," in *ACM SIGGRAPH Computer Graphics*, New York, USA, July 1992, pp. 203–210.

[48] P. Sajda, K.-R. Müller, and K. V. Shenoy, "Brain-Computer Interfaces," *IEEE Signal Processing Magazine*, vol. 25, no. 1, pp. 16–17, January 2008.

[49] M. D. Linderman, G. Santhanam, C. T. Kemere, V. Gilja, S. O'Driscoll, M. Y. Byron, A. Afshar, S. I. Ryu, K. V. Shenoy, and T. H. Meng, "Signal Processing Challenges for Neural Prostheses," *IEEE Signal Processing Magazine*, vol. 25, no. 1, pp. 18–28, January 2008.

[50] M. Turk, "Review Article: Multimodal Interaction: A Review," *Pattern Recognition Letters*, vol. 36, pp. 189–195, January 2014.

[51] H. Bunt, R.-J. Beun, and T. Borghuis, *Multimodal Human-Computer Communication: Systems, Techniques, and Experiments*. Springer Science & Business Media, April 1998.

[52] S. Oviatt, "Advances in Robust Multimodal Interface Design," *IEEE Computer Graphics and Applications*, vol. 23, no. 5, pp. 62–68, September 2003.

[53] J. A. Jacko, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*. CRC press, May 2012.

[54] R. A. Bolt, "&Ldquo;Put-that-there&Rdquo;: Voice and Gesture at the Graphics Interface," in *Proceedings of the 7th Annual Conference*

on Computer Graphics and Interactive Techniques, New York, NY, USA, July 1980, pp. 262–270.

[55] A. Farner, "Report on Speech Gesture Recognition Systems," 2009, accessed: 2016-08-18.

[56] A. Popescu-Belis, H. Bourlard, and S. Renals, *Machine Learning for Multimodal Interaction IV*. Springer-Verlag, September 2008.

[57] P. Cohen, D. McGee, S. Oviatt, L. Wu, J. Clow, R. King, S. Julier, and L. Rosenblum, "Multimodal Interaction for 2D and 3D Environments," *IEEE Computer Graphics and Applications*, vol. 19, no. 4, pp. 10–13, July 1999.

[58] E.-L. Sallnäs, K. Rassmus-Gröhn, and C. Sjöström, "Supporting Presence in Collaborative Environments by Haptic Force Feedback," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 4, pp. 461–476, December 2000.

[59] E. Tse, S. Greenberg, and C. Shen, "Multi User Multimodal Tabletop Interaction over Existing Single User Applications," in *Adjunct Proc ACM CSCW*, Banff, Canada, November 2006, pp. 111–112.

[60] N. Negroponte, "An Iconoclastic View Beyond the Desktop Metaphor," *International Journal of Human-Computer Interaction*, vol. 1, no. 1, pp. 109–113, September 1989.

[61] N. Sebe, "Multimodal Interfaces: Challenges and Perspectives," *Journal of Ambient Intelligence and Smart Environments*, vol. 1, no. 1, pp. 23–30, January 2009.

[62] S. Oviatt, "Ten Myths of Multimodal Interaction," *Communications of the ACM*, vol. 42, no. 11, pp. 74–81, November 1999.

[63] A. K. Bhowmik, *Interactive Displays: Natural Human-Interface Technologies*. John Wiley & Sons, January 2014.

[64] M. Tavanti, "Multimodal Interfaces: A Brief Literature," *Human-Computer Interaction*, vol. 12, pp. 93–129, April 2007.

[65] L. M. Reeves, J. Lai, J. A. Larson, S. Oviatt, T. S. Balaji, S. Buisine, P. Collings, P. Cohen, B. Kraal, J.-C. Martin, M. McTear, T. Raman, K. M. Stanney, H. Su, and Q. Y. Wang, "Guidelines for Multimodal User Interface Design," *Communications of the ACM*, vol. 47, no. 1, pp. 57–59, January 2004.

[66] L. Schomaker, J. Nijtmans, A. Camurri, F. Lavagetto, P. Morasso, and C. Benoît, "A Taxonomy of Multimodal Interaction in the Human Information Processing System," 1995, accessed: 2016-08-18. [Online]. Available: http://www.ai.rug.nl/~lambert/projects/miami/reports/taxrep-300dpi.pdf

[67] D. A. Norman, *The Design of Everyday Things: Revised and Expanded Edition*. Basic books, November 2013.

[68] P. K. Atrey, M. A. Hossain, A. El Saddik, and M. S. Kankanhalli, "Multimodal Fusion for Multimedia Analysis: A Survey," *Multimedia systems*, vol. 16, no. 6, pp. 345–379, November 2010.

[69] L. Nigay and J. Coutaz, "A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion," in *Proceedings of the INTERACT 1993 and CHI 1993 conference on Human factors in computing systems*, New York, USA, April 1993, pp. 172–178.

[70] J. A. ter Heerdt, "Design Spaces, Mechanisms and Architectures for Multimodal Interfaces," *1st Twente Student Conference on IT*, June 2004.

[71] D. Lalanne, L. Nigay, p. Palanque, P. Robinson, J. Vanderdonckt, and J.-F. Ladry, "Fusion Engines for Multimodal Input: A Survey," in *Proceedings of the 2009 international conference on Multimodal interfaces*, Cambridge, USA, November 2009, pp. 153–160.

[72] R. Sharma, V. I. Pavlovic, and T. S. Huang, "Toward Multimodal Human-Computer Interface," *Proceedings of the IEEE*, vol. 86, no. 5, pp. 853–869, May 1998.

[73] M. E. Foster, "State-of-the-Art Review: Multimodal Fission," *COMIC project Deliverable*, vol. 6, no. 09, September 2002.

[74] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. M. Young, "Four Easy Pieces for Assessing the Usability of Multimodal Interaction: the CARE Properties," in *INTERACT: IFIP International Conference on Human Computer Interaction*, Lillehammer, Norway, June 1995, pp. 115–120.

[75] L. Nigay and J. Coutaz, "Multifeature Systems: The CARE Properties and Their Impact on Software Design," in *Multimedia Interfaces: Research and Applications*, July 1997.

[76] D. Tzovaras, *Multimodal User Interfaces: From Signals to Interaction*. Springer Science & Business Media, February 2008.

[77] F. C. Lunenburg, "Communication: The Process, Barriers, and Improving Effectiveness," *Schooling*, vol. 1, no. 1, pp. 1–11, 2010.

[78] SkillsYouNeed, "What is Communication?" 2016, accessed: 2016-05-08. [Online]. Available: http://www.skillsyouneed.com/general/what-is-communication.html

[79] J.-P. Brun and C. Cooper, *Missing Pieces: 7 Ways to Improve Employee Well-Being and Organizational Effectiveness*. Springer, January 2009.

[80] D. C. Summers, *Quality Management: Creating and Sustaining Organizational Effectiveness*. Pearson Prentice Hall, January 2005.

[81] M. Yate, *Hiring the Best: A Manager's Guide to Effective Interviewing and Recruiting*. F+ W Media, Inc., August 2005.

[82] E. M. Eisenberg, H. L. Goodall Jr., and A. Trethewey, *Organizational Communication: Balancing Creativity and Constraint*. Macmillan Higher Education, November 2013.

[83] C. Learning, "Understanding the Basics of Verbal Communication," 2009, accessed2016-02-01. [Online]. Available: http://assets.cengage.com/pdf/4004_1111063796_Understanding\%20the\%20Basics\%20of\%20Verbal\%20Communication.pdf

[84] S. Adrignola, "Survey of Communication Study/Chapter 2 - Verbal Communication," 2009, accessed: 2016-08-18. [Online]. Available: http://www.saylor.org/site/wp-content/uploads/2012/05/COMM001_Wikibooks_-Survey-of-Communication-Study_Chapter-2_5.11.2012.pdf

[85] S. R. Van Hook, "Verbal and Nonverbal Communication," 2009, accessed: 2016-08-18. [Online]. Available: http://www.saylor.org/site/wp-content/uploads/2013/03/CUST105-1.4-FINAL.pdf

[86] SkillsYouNeed, "Non-Verbal Communication," 2016, 2016-07-06. [Online]. Available: http://www.skillsyouneed.com/ips/nonverbal-communication.html

[87] N. Ambady and R. Rosenthal, "Nonverbal Communication," *Encyclopedia of mental health*, vol. 2, pp. 775–782, 1998.

[88] M. Argyle, *Bodily Communication.* Routledge, April 2013.

[89] T. Dixon and M. O'Hara, "Communication Skills," 2013. [Online]. Available: http://cw.routledge.com/textbooks/9780415537902/data/learning/11_Communication\%20Skills.pdf

[90] J. B. Stiff, J. L. Hale, R. Garlick, and R. G. Rogan, "Effect of Cue Incongruence and Social Normative Influences on Individual Judgments of Honesty and Deceit," *Southern Journal of Communication*, vol. 55, no. 2, pp. 206–229, March 1990.

[91] H. Ellgring, *Non-Verbal Communication in Depression.* Cambridge University Press, November 2007.

[92] SkillsYouNeed, "Body Language, Posture and Proximity," 2016, accessed: 2016-07-06. [Online]. Available: http://www.skillsyouneed.com/ips/body-language.html

[93] M. L. Knapp, J. A. Hall, and T. G. Horgan, *Nonverbal Communication in Human Interaction.* Cengage Learning, January 2013.

[94] W. Bank, "Non-Verbal Communication," 2010. [Online]. Available: http://siteresources.worldbank.org/EXTGOVACC/Resources/NonverbalCommweb.pdf

[95] E. Goldman, *As Others See Us: Body Movement and the Art of Successful Communication.* Routlegde, 2004.

[96] SkillsYouNeed, "Effective Speaking," 2016, 2016-07-03. [Online]. Available: http://www.skillsyouneed.com/general/what-is-communication.html

[97] L. Wong, *Essential study skills.* Cengage Learning, January 2014.

[98] A. Mehrabian and M. Wiener, "Decoding of Inconsistent Communications," *Journal of personality and social psychology*, vol. 6, no. 1, p. 109, May 1967.

[99] A. Mehrabian and S. R. Ferris, "Inference of Attitudes from Nonverbal Communication in two Channels," *Journal of consulting psychology*, vol. 31, no. 3, p. 248, June 1967.

[100] A. Mehrabian, ""Silent Messages" – A Wealth of Information About Nonverbal Communication (Body Language)," *Personality & Emotion Tests & Software: Psychological Books & Articles of Popular Interest. Los Angeles, CA: self-published*, vol. 7, no. 31, p. 2011, 2009.

[101] A. Georgievska, "Comunication, the Importance of Feedback and a Study Research on the Rating of the Two Courses "Advanced Fife Support" and "Emotional Management in the Areas of Emergency"," December 2011.

[102] J. Ford, J. Knight, and E. McDonald-Littleton, "Learning Skills: A comprehensive Orientation and Study Skills Course Designed for Tennessee Families First Adult Education Classes," *Knoxville: University of Tennessee, Knoxville Center for Literacy Studies*, vol. 16, p. 2009, November 2001.

[103] Google, "Glass," accessed: 2016-02-09. [Online]. Available: https://developers.google.com/glass/

[104] M. Rouse, "Google Glass," accessed: 2016-02-09. [Online]. Available: http://internetofthingsagenda.techtarget.com/definition/Google-Glass

[105] I. Bartenieff and D. Lewis, *Body Movement: Coping with the Environment*. Routlegde, January 1980.

[106] P. Boersma and D. Weenink, "Praat, a System for Doing Phonetics by Computer," *Glot international*, vol. 5, no. 9/10, pp. 341–345, October 2002.

[107] L. Batrinca, G. Stratou, A. Shapiro, L.-P. Morency, and S. Scherer, "Cicero - Towards a Multimodal Virtual Audience Platform for Public Speaking Training," in *International Workshop on Intelligent Virtual Agents*, August, Edinburgh, UK 2013, pp. 116–128.

[108] G. Degottex, J. Kane, T. Drugman, T. Raitio, and S. Scherer, "COVAREP – A Collaborative Voice Analysis Repository for Speech Technologies," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, May 2014, pp. 960–964.

[109] Imotions, "Emotient Module: Facial Expression Emotion Analysis," 2016, accessed: 2016-08-01. [Online]. Available: https://imotions.com/attention-tool-facet-module-facial-action-coding-system-facs/

[110] S. Lao and M. Kawade, "Vision-Based Face Understanding Technologies and their Applications," in *Advances in Biometric Person Authentication*. Springer Berlin Heidelberg, 2004, pp. 339–348.

[111] Microsoft, "Managed Extensibility Framework (MEF)," 2016, accessed: 2016-07-16. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd460648(v=vs.110).aspx

[112] ——, "Reflection in the .NET Framework," 2016, accessed: 2016-07-17. [Online]. Available: https://msdn.microsoft.com/en-us/library/f7ykdhsy.aspx

[113] Redhat, "Why use a Rule Engine?" 2016, accessed: 2016-07-17. [Online]. Available: https://access.redhat.com/documentation/en-US/JBoss_Enterprise_SOA_Platform/4.3/html/JBoss_Rules_Reference_Guide/the_rule_engine-why_use_a_rule_engine.html

[114] C. Open-Source, "Drools.NET," 2014, accessed: 2016-07-17. [Online]. Available: http://www.csharpopensource.com/droolsdotnet/

[115] S. Nikolayev, "NRules," 2015, accessed: 2016-03-15. [Online]. Available: https://github.com/NRules/NRules

[116] B. Schneier, "The Rete Matching Algorithm," 2002, accessed: 2016-03-15. [Online]. Available: http://www.drdobbs.com/architecture-and-design/the-rete-matching-algorithm/184405218

[117] Microsoft, "Introduction to the C# Language and the .NET Framework," 2016, accessed: 2016-07-17. [Online]. Available: https://msdn.microsoft.com/en-US/library/z1zx9t92.aspx?f=255&MSPPError=-2147217396

[118] ——, "Windows Presentation Foundation," 2016, accessed: 2016-07-17. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd460648(v=vs.110).aspx

[119] ——, "XAML Overview (WPF)," 2016, accessed: 2016-07-17. [Online]. Available: https://msdn.microsoft.com/en-US/library/ms752059(v=vs.110).aspx

[120] ——, "The MVVM Pattern," 2016, accessed: 2016-07-17. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx

[121] ——, "Object (C# Reference)," 2016, accessed: 2016-07-22. [Online]. Available: https://msdn.microsoft.com/en-us/library/9kkx3h3c.aspx

[122] W. K. Pediaopolis, "The 5 Senses," 2016, accessed: 2016-02-01. [Online]. Available: http://udel.edu/~bcarey/ART307/project1_4b/

[123] Oxford, "Sensor," 2016, accessed: 2016-02-01. [Online]. Available: http://www.oxforddictionaries.com/definition/english/sensor

[124] Microsoft, "Kinect for Windows Sensor Components and Specifications," 2016, accessed: 2016-02-01. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx

[125] ——, "Meet Kinect for Windows," 2016, accessed: 2016-02-01. [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect

[126] ——, "Kinect for Windows Architecture," 2016, accessed: 2016-07-16. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131023.aspx

[127] ——, "Coordinate Spaces," 2016, accessed: 2016-07-16. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh973078.aspx

[128] ——, "Tracking Users with Kinect Skeletal Tracking," 2016, accessed: 2016-07-16. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131025.aspx

[129] ——, "Face Tracking," 2016, accessed: 2016-07-23. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj130970.aspx