Technische Universität München
Fakultät für Maschinenwesen
Lehrstuhl für Automatisierung und Informationssysteme
Prof. Dr.-Ing. Birgit Vogel-Heuser

Bachelor's Thesis

Concept and development of a human robot interface using parametrised Augmented Reality

Konzept und Entwicklung einer Mensch-Roboter Schnittstelle mit parametrisierbarer Augmented Reality (GERMAN)

Author:	Eric Vollenweider
Supervisor:	Prof. Dr.-Ing. Birgit Vogel-Heuser
Advisor:	Dr. Pantförder Dorothea
Issue date:	Februar 26, 2019
Submission date:	Juli 16, 2019

Abstract

Developing industrial Augmented Reality (AR) applications requires a certain set of skills which professionals of their field often do not have. This either results in fewer or less quick AR applications being developed, or Software Engineers being hired, which increases communication efforts, costs and possibly the time-to-market. The presented framework "PARRHI" (Parametrised Augmented Reality Robot-Human Interface) tries to solve this problem by enabling people who do not possess deep AR developing experience to develop and execute AR applications that are capable of communicating with industrial machines for their specific domains. This is done by creating an abstract layer above involved technologies and disclosing data to the developer through parameters. The framework was implemented in C# and Unity using a Fanuc CR-i7A Robot as a machine-demonstrator and a Microsoft HoloLens as an AR device. The framework's implementation has been evaluated against usability and intuitiveness by external participants posing as developers. The evaluation showed that the PARRI framework successfully relieves the developer from all AR related challenges and simplifies the development workflow, but has its limitations with extendibility. Solutions to these problems are proposed.

The parametrised development of industrial Augmented Reality applications as proposed in this thesis seems to be a promising approach but needs further research and work before it can be deployed to the industry.

Kurzzusammenfassung

Die Entwicklung von Augmented Reality (AR) Anwendungen erfordert ein bestimmtes Set an Fähigkeiten, welches Spezialisten in unterschiedlichen Industrien oft nicht besitzen. Dies resultiert darin, dass AR Anwendungen entweder gar nicht oder langsamer entwickelt werden, oder, dass Software Entwickler beauftragt werden, was Kommunikationsaufwände, Kosten und möglicherweise die Entwicklungszeit negativ beeinflusst. Das präsentierte Konzept "PARRHI" (Parametrisierte Augmented Reality Roboter Mensch Schnittstelle) versucht dieses Problem zu lösen, indem es Menschen, welche keine Erfahrungen in AR Entwicklung haben, ermöglicht industrielle AR Anwendungen zu entwickeln und auszuführen. Diese AR Anwendungen können mit industriellen Maschinen der jeweiligen spezifischen Industrie kommunizieren. Dies wird erreicht, indem ein abstraktes System über alle involvierten Technologien gespannt wird, welches Daten dem Entwickler durch Parameter zugänglich macht. Das System wurde in C# und Unity am Beispiel eines Fanuc CR-i7A Roboters und einer Microsoft HoloLens implementiert. Eine Evaluierung untersuchte die Anwenderfreundlichkeit und Intuitivität der Entwicklungsumgebung anhand externer Versuchspersonen, welche eine Demo-Anwendung im PARRHI System implementierten. Die Evaluierung zeigte, dass das PARRHI System den Entwicklern sämtliche AR verwandten Herausforderungen erfolgreich abnimmt und die Entwicklungsprozesse vereinfacht. Allerdings kamen Probleme mit der Erweiterbarkeit des Systems zum Vorschein. Lösungsansätze zu den genannten Problemen werden am Ende der Arbeit vorgeschlagen.

Die parametrisierte Entwicklung von industriellen Augmented Reality Anwendungen, wie sie in diese Arbeit vorgeschlagen wird, scheint eine vielversprechende Herangehensweise zu sein, benötigt allerdings noch intensivere Forschungsarbeit und Entwicklung, bevor das Konzept in der Industrie zur Anwendung kommen kann.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Reasons for- and Challenges with Human-Robot Collaboration

Today's world is becoming increasingly automated in most aspects of our lives. One of the most important advancements in the last 30 years were robotic systems. They allow reducing costs while simultaneously increasing output, quality and efficiency. With the rising need for agile production plants, non-collaborative classic robotic systems do not seem to fulfil the modern requirements of Industry 4.0 any more, since they lack the ability to cooperate with humans at the required level, which is essential to combine the agility and intelligence of humans with the speed, precision and power of robots. Current industrial robots' lack the ability of adapting quickly to changing environments. In contrast, the human's strength lies in being able to learn and adapt quickly and flexibly. This is one of many reasons why Human-Robot (HR) collaboration is an inevitable step towards the future of robotics.

Robotics needed to adapt to a new situation where humans and robots share a common perimeter. Thus compliant robots were developed. It is not enough to be compliant and to not hurt humans for machines to act naturally, though. As Gary Klein et al. stated, communicating intent is a key issue in effective collaboration within teams [1].

Human-Machine communication with robotic systems – a critical requirement of modern robotics - can be supported by Augmented Reality (AR). With rapid advancements in the field of robotics during the last few years [2], innovative ways to program, develop and operate these highly complex systems need to be researched and developed. The field of AR has been a highly active field for the last 20 years, but the combination of AR and robotics has remained mostly untouched. Lately though, it has become a new focus area in research. While it is known that AR can improve communication in specific use cases [3], actually developing them is complex due to multiple and highly diverse technical challenges needing to be resolved, e.g. 3D modelling, image tracking, performance and many more. Since most development environments require source code to be written for each use case, the reusability is quite low

which may result in a lot of duplication of efforts by software Engineers producing highly specialised source code. Thus, the time-to-market is long and investments are significant, if not prohibitive for many smaller use cases.

According to the U.S. Department of Labour, there is a massive hunt for talent in the software industry [4]. The department reported an expected growth of job employment in this industry of over 30% within the next 10 years. One possible reason for the lack of talent could be that (similarly to many other engineering fields) acquiring the necessary skills to develop software applications takes a long time and is viewed as a challenging career choice. The lack of coding talent is also experienced in the robotics industry, which requires its developers to possess an even wider combination of skills, including software development, mechanical engineering and the basics of electrical engineering. The scarcity of developers has to the potential to slow down innovation and to extend the time-to-market. This in turn leads to a market premium of about 20% for such developers [5].

Furthermore, managers responsible for developing AR apps are professionals in their own fields, but might lack the detailed knowledge of how to develop AR applications. In addition, the highly interdisciplinary nature of such systems further contributes to the challenge of finding enough talent.

In conclusion, AR applications can play a critical role in effective collaboration between humans and robots. Unfortunately, it is not easy to find enough qualified employees who have the domain-specific knowledge in developing AR applications. If non-programmers could set up, configure and develop Human-Robot interfaces involving AR and robot control all by themselves, i.e. without the need to write large amounts of source code and without being proficient in AR development, the cost efficiency and speed of development could be improved significantly.

However, the solution to the above challenge might lie in a strategy successfully used by many industries called *parametrised development* (see 2.3), which has many great advantages. For example, changes can be made more easily, since only the input parameters have to updated, or such input parameters can also be generated in specialised applications. Both can help mitigate the challenges involved in AR application development, making it easier for more and less specialised people to contribute.

This bachelor thesis aims to research the feasibility and practicability of such an abstract development environment which supports developing Augmented Reality interfaces for industrial applications without any specialised knowledge of AR. Chapter 2 describes the ”State of the Art” including current research and progress. Chapter 3 identifies gaps in current

approaches and proposes a new concept. Chapter 4 describes an actual implementation in a physical robot/AR environment, and Chapter 5 evaluates the implementation's performance. Finally, Chapter 6 analyses certain aspects of generalising the concept for a wider spectrum of devices.

2 State of the Art

This chapter highlights technologies and products which have contributed to this field of research. First, Augmented Reality will be explained and some important characteristics will be laid out. Then currently available AR-Standardisations will be described. Standards are important since they help developers speed up their processes using predefined solutions and approaches. Further, the combined field of AR and Robotics will be described in detail: a) how Augmented Reality technology is being applied, b) which projects succeeded in doing so, and c) which benefits were realised. Then, a description of parametrised thinking/coding and other related disciplines that already utilise such an approach draws out a set of important learnings as the basis for the concept presented in Chapter 3. And finally, a brief overview is presented how this thesis tries to combine the above technologies and principles into an innovative way of programming and maintaining AR applications.

2.1 Augmented Reality and its Standardisations

This bachelor's thesis revolves around AR and its use cases. It is important to clearly define some terms before using them throughout this document. AR technology must not be confused with Virtual Reality (VR). Whereas VR completely immerses the users, AR still allows the real world to be seen together with superimposed 3D objects projected into the user's view. T. Azuma gave a very general definition of AR-Systems very early on [6]. He wrote that in order for a system to be classified as an AR system it has to fulfil three characteristics: 1) The system has to combine real and virtual objects, 2) it has to be interactive in real time, and 3) it has to spatially map physical and virtual objects to each other. The term Augmented Reality is often used as a synonym for the also commonly used expression "Mixed Reality".

There are different underlying principles in AR technology. Two types of AR technologies are in use today: 1) see-through displays with strong user immersion, and 2) monitor-based

approaches like smartphones with cameras. A large variety of producers [7] release new AR-Devices in increasing frequency.

Standards often substantially contribute to the development and usage of technologies firstly, by unifying used technology stacks, thus potentially reducing complexity, and secondly by helping new members to navigate in a complex environment. There are not many international standards for AR due to the field's emerging nature. Nevertheless, C. Perey et al. [8] examined existing standards and some best-practice methods for AR and identified some gaps in the AR value-chain pointing out interoperability problems between different components. Another conclusion was that there are some existing standards from other domains that could be used for AR, but that there is not enough agreement on which ones. The researchers lay out numerous standards for low-level implementations such as geographic location tracking, image tracking, network data transmission, etc.

Figueroa et al. [9] researched a "Conceptual Model and Specification Language for Mixed Reality Interface Components". They intended to lay a foundation for other developers to standardize 3D interface assets for others to build upon. The team developed an XML-based data structure called "3DIC" that defines the look and some minor behavioural actions of UI control elements. Although their work features useful components, it does not yet include certain features that robot-human AR interfaces need such as communicating with external machines. In addition, "3DIC" uses pseudo code that cannot be fully processed automatically, since pseudo-code is not executable.

Concluding from this general overview on AR technology, there does not seem to be an existing standard for AR-HR-Interfaces utilizing parametrised Augmented Reality, that allows the development of complete HR-AR-applications that can be automatically processed, executed and used.

2.2 AR in Robotic Spplications

2.2.1 AR during Development and Testing of Robotic Systems

Wolfgang Hönig et al. examined three different use cases of Augmented Reality [10]. The research team primarily focused on the benefits offered by the co-existence of virtual and real objects during the development and testing phase of robotic systems. They documented three individual projects where the implementation of AR as a main feature resulted in lowered

safety risks, simplified debugging and the possibility of easily modifying the actual, physical setup.

Another important dimension of development and testing is of financial nature: upfront investments and operating costs. For example, robotic aerial swarms tend to be quite expensive due to the high cost of drones. Hönig et al. successfully scaled up the number of objects in their swarms without adding physical hardware by simulating additional drones in AR, thus, saving money and space [10]. However, it is stated that this approach might not be applicable to all experiments since simulations are never perfect replicas of actual systems. This small delta in physical behaviour might be enough to raise doubts regarding the correctness of the experiment results.

All projects by Hönig et al. focus on isolating certain aspects of the system to analyse and test them more flexibly, cheaply or with improved safety for all participants involved (humans and machines). Similarly, Chen et al. have created a software framework for simulating certain parts of robotic systems [11]. These researchers worked on methods to combine real world and simulated sensor data and navigate a real-world robot in the combined environment. Their approach was to intercept the raw sensor data originating from the real robot, and mixing it with the simulated data, before publishing it to the receivers.

For the to be built AR-Human Robot Interface, this approach could not only be used to combine the simulated virtual world and the real world data and then use it for the system's logic components but also to fully simulate the robot during the development of the system. This allows a decoupling of the development from the real world production hardware.

2.2.2 Operating Robotic Systems with AR Support

A well-known bottleneck in robotics is the controlling of and thus, the communication with robots [12]. In all previously cited cases, AR was not used to improve the interaction between humans and robots but to mitigate the current challenges in developing robotic systems. Early work by Milgram et al. [13] shows that even the most basic implementations of AR technology with the objective to improve the information exchange, enhance the bidirectional communication between the humand and the machine in multiple ways. The team proposed means to relieve human operators by releasing them from the direct control loop and using virtually placed objects as controlling input parameters. This replaces direct control with a more general command process.

As Gary Klein et al. stated, communicating intent is a key issue in effective collaboration within teams [1]. Whenever robots and humans collaborate in a confined space, it is critically important to know each other's plans or strategies in order to align and coordinate joint actions. For machines lacking anthropomorphic and zoomorphic features, such as aerial and industrial robots, it is unclear how the before-mentioned information can be communicated in natural ways between the humans and the machines.

In order to solve this problem, Walker et al. [14] explored numerous methods to utilize AR to improve both the efficiency and the acceptance of robot-human collaborations via conveying the robot's intent. The group of researchers defined four methods of doing so, with varying importance being put on "information conveyed, information precision, generalizability and possibility for distraction" [14]. The conclusion was that spatial AR holograms are received much more intuitively than simple 2D projected interfaces.

For the framework to be built this implies that there have to be tools in place for the developer to explicitly communicate the robot's intent to the operator. There should also be ways to adapt the application's appearance to the domain in question (like industrial plants, laboratories or logistic centres), to maximise the acceptance of such interfaces.

2.3 Parametrised Development

Some disciplines heavily utilize parametrised development environments, which allows to define the relationship between a process' intent to the outcome via parameters and rules. Adjusting the parameters automatically adjusts the outcome, which results in a high process-maintainability and agility. For example the CAD software package *CADENCE* (used for designing integrated electrical circuits) offers parametrised cells to optimize the development process [15]. Users can place these cells and adjust certain parameters. For example, spiral inductors can be configured via a simple UI that sets the parameters in the background. Aspects like outer dimensions, metal width, number of layers, etc. can be configured. The software then calculates its properties and behaviour at runtime. Generally, there is no coding skill required.

In architecture, Parametric Design has become increasingly prevalent since the late 2000s. New capabilities in computer rendering and modelling opened up a whole new world for architects at the time. Using parametrised mathematical formulas with boundary constraints allow architects to generate building structures more easily and more efficiently [16]. Adaptions to a changing environment during the design process can be executed much more easily, since

a substantial portion of the work can be completed by algorithms and software applications. Additionally, the reusability of components strongly increases due to the formal, abstract way of representation. Architecture studios are also beginning to include VR and AR technologies actively in their design processes to further incorporate and better understand this new design method called *Parametric Design* [17]–[19].

2.4 Current Methods Programming Industrial Robots

To understand where and how the content of this research project fits into the world of robotics, this chapter describes the current workflows with robots and how they are configured and programmed. It is important to note that almost all industrial robots are programmed in three ways.

1.) Teach Pendant

2.) Simulation / Offline Programming

3.) Teaching by Demonstration

According to the British Automation & Robot Association, over 90% of all industrial robots are programmed using the Teach Pendant (TP) method [20]. Basically, these devices are touch-tablets fitted for industrial use with emergency stop buttons, more durable materials, etc. They allow the operator to jog (a term for steering the robot manually using the TP) the robot into certain positions and offer a number of other possibilities. For simpler tasks, some manufacturers (e.g. Fanuc) offer specific User Interfaces where the operator simply enters parameters and positions. More complex goals can also be achieved with the Teach Pendant by programming in a textual manner, using each manufacturer's own language. This type of programming requires training and practice to achieve good results.

Teach Pendants are great for trivial and simple tasks, not requiring a lot of collaboration between factory components. Reprogramming the robot using this method leads to a partial downtime since the actual real robot has to be used. It is important to note that parametrised programming is already an industry standard in industrial robotics offered by several manufacturers.

Offline programming is most often used for more complex tasks and production lines. While Offline Programming is very precise and powerful, it does require a substantial amount of training time for the operator. Teaching by Demonstration on the other hand, is the

exact opposite. Most people succeed quickly, but the complexity of achievable tasks and the corresponding precision is limited.

The AR Robot-Human Interface proposed in this thesis could be categorised as Offline Programming, also interacts in the Teach Pendant mode, since the operator should be able to take over control of the robot when necessary. It can therefore be categorised as a hybrid approach.

2.5 Combining Technologies to Mitigate AR Development Challenges

The above technologies and principles can be combined in an attempt to mitigate challenges during AR development.

Many industries are beginning to adopt AR in their daily processes. Its advantages and challenges are well known and being further developed. There are numerous examples of AR helping in different environments and tasks [21]–[23].

However, developing AR applications is generally-speaking still a very costly task. Applying the design principle of "Parametrisation" to AR might help lower the initial investment and the subsequent cost of changes and adaptions to altering environments. In addition, parametrisation offers the added benefit of enabling non-specialist developers to design AR use cases, thereby reducing the need for highly skilled and specialised programmers. Combining the benefits of both AR technology and parametrised architecture might therefore bear great potential.

In conclusion, there is prior work on most aspects of this bachelor's thesis' focus topic, which is a parametrised Augmented Reality Robot-Human Interface. Some of the discussed frameworks ('3DIC', [9]) allow the textual definition of AR interfaces, but lack the ability to control robotic hardware and contain components like pseudo code that does not support an automated environment. Other projects showed the feasibility of combining simulated and real-world data to control robotic systems, but required programming by professional software engineers with experience in AR and robotics. In conclusion, there is no published research describing systems allowing simple parametrised representations of AR interfaces able to communicate with robots allowing more complex, higher-level applications to be built.

3 Concept

This chapter proposes a novel concept for AR in Robotics based on parametrisation addressing the challenges laid out in the previous chapters. It starts by describing the goal and the general requirements for such a framework, particularly its design and setup. Then, a practical sample use case illustrates the goal and purpose of this research project. Chapter 3 then proceeds to proposing and detailing the *PARRHI* ("Parametrised Augmented Reality Robot-Human Interface") AR development framework.

3.1 Goal, Requirements and Use Cases

This thesis presents a novel approach to solve the challenges previously described in Section 1.1. It is the main objective of this thesis to demonstrate an AR development framework allowing to remove the necessity for advanced software engineering skills in developing reasonably complex AR Robot-Human Interfaces while simultaneously maintaining the quality of the outcome and increasing the degree of reusability. The intended benefits include lower investments and operating costs for AR use case design, shorter development cycles and corresponding time-to-market, and enlarging the talent pool required for AR use case development by reducing the need for highly specialised skill sets.

As a first step, a specific set of requirements has to be defined and formally documented.

3.1.1 Requirements

Before defining the framework's requirements, the typical end user persona needs to be described first. It is a generic AR developer, who develops AR use cases and applications for industrial systems using the PARRHI framework. For example, such personae could include application engineers, plant operators, or quality control employees among many other categories. Professionals in such or similar roles usually have a profound technical

background, but often do not have any specific experience in AR development or software development.

The PARRHI framework should allow these types of end users to:

- Create AR applications without any or much prior knowledge of AR development (image tracking, AR hardware, corresponding engines and frameworks)
- Leverage their domain-specific knowledge in the application's workflow, content and appearance.
- Superimpose the real world with three-dimensional holograms to describe, highlight or mark real and virtual objects.
- Reuse previous work easily since many applications share similar base workflows and only vary in details.
- Build AR applications for multiple platforms (desktop, mobile devices, tables, head-mounted AR devices)
- Use the tools necessary to create use cases of medium complexity such as tutorials, maintenance instructions or similar tasks.
- Control the robot, but also to let the operator take over the control at runtime.
- Simulate the real and the virtual worlds to allow development without the need to access the physical robots during the initial phases of development.

Derived from these user requirements, the PARRHI framework should provide the following capabilities and functionalities:

1.) To automatically handle most lower-level AR methods and functionalities in the background without any specific developer input. This includes basic functionalities such as the position recognition of the AR end user and the robot, the corresponding image and motion tracking, and the build-processes for different hardware and software platforms.

2.) To allow programming and parametrisation in a simple and intuitive way.

3.) To allow the reuse code from other projects.

4.) To allow the definition of 2D/3D objects with the possibility of attaching these objects to real-world objects such as the robot or the user.

5.) To deliver human-legible and intuitive feedback on the written AR PARRHI meta-code, comparable to, but not exactly equal to compiler error messages, since they often provide limited usefulness to non-experienced programmers.

6.) To support building use cases for multiple platforms such as handheld mobile devices and headmounted AR glasses.

7.) To allow creating AR use cases for medium complexity workflows (values in the range of 1-15 according to the McCabe Metric [24]). Event-based logic, conditions and actions should be offered.

8.) To allow the combination of simulated and real-world data to be used for the application's workflow and its logical operators.

9.) To allow the sensing and commanding of the robot's position both relatively and absolutely in the (robot) joint and task space.

10.) To allow the documentation of the application's workflow via comments

These basic requirements should be a guideline for further conceptualisation and therefore for the implementation. The next section will visualise one possible use case of the PARRHI System and why it might be interesting to utilise it here.

3.1.2 Use-Case example: Factory Maintenance

This section describes a single real-world sample use case where the PARRHI framework can be applied. But the reader should bear in mind that the PARRHI framework is a general development environment designed to handle a wide variety of classes of use cases and workflows. Further examples could be AR applications for educational/training purposes, factory workflow processes, investigation and analytic tasks, and process profiling.

The specific sample use case is described in the context of a manufacturing company producing its customised products in an agile factory. Plant operators should be supported by an AR app to ensure their safety and efficiency. The factory has highly automated workflows and production lines. Their assembly lines contain robots tasked to prepare the customised parts for the products produced, where each one of them has varying components built into them. The process of mounting and fitting these parts is not fully automated and thus still requires human collaborators.

Application engineers are tasked with developing AR applications that support the collaborative process between humans and machines during this assembly step to increase the safety and efficiency. These engineers have never built any AR applications before, but have substantial experience with robot assembly lines and corresponding surveillance applications. The engineers need this application to communicate the intent of robots to the surrounding

humans, to control the robot to perform its next movement, and also to ensure that the collaboration between all parties remains safe, so that neither the humans nor the robots get harmed or damaged. This should be achieved by visually augmenting real-world objects, but also by directly taking over the control of robots.

These engineers now use the PARRHI framework and development tool kit to set up AR applications for their respective missions. One, for example, constantly monitors the human's distance to the robot's moving parts and warns the employee visually if they are located within the danger zone. At the same time, it always visualises the robot's range of motion and a danger perimeter around its axes, so that the employee has more contextual information to work with. In the case of the human getting too close to the robot, the application immediately stops the robot and asks the user to move away into a safe space.

The engineers developing these applications can focus on their specific domain knowledge and fill the application with the relevant information without having to spend a substantial amount of time on system setups or framework installations. This use case is exemplary because it describes a realistic scenario in the real world. Very specific domain knowledge has to be packaged into AR applications, where the developers have no prior experience with AR application development and are under time pressure.

As mentioned at the top of this section, there are many classes of use cases for the PARRI framework. It is important to notice that for these use cases no AR specialists are needed since the system takes care of its components. Of course, for some use cases adaptions would have to be made. Certain modules would have to be expanded or adapted for PARRHI to be able to communicate with other machines (see technology transfer section 6).

In essence, the majority of the development effort can be invested in the application's workflow, which resolves the real-world challenges. The underlying low-level tasks can be ignored by the developer, since the PARRHI framework takes care of them automatically in the background.

3.2 PARRHI Framework

This chapter describes the proposed concept of the PARRHI framework. It starts with a general overview of the concept and how each requirement is addressed before briefly detailing each individual component. After that, the information flow in the system is visualised and commented.

Relieving the developer from AR efforts (Requirement #1) implies that the system needs to have all the underlying AR logic implemented on its own without the need for any developer input. Thus, the architecture has an input and an output module, which handle the AR aspects such as image tracking, gesture recognition and 3D hologram generation. The input module has to use the AR device's camera to perform the image and motion tracking, and the output module has to generate the corresponding holograms. The developer's input into the system has to be a document (Requirement #2), which allows the definition of parametrised objects (2D/3D) (Requirement #3) and logic components (Requirement #6). Since the developers need feedback on their work (Requirement #4), the input document has to be validated. Combining real and simulated world data (Requirement #7) requires the input module to intercept the data stream of the real world, and to feed the simulated data into the stream before passing it on. Finally, the input and output modules have to communicate with the robot (Requirement #8).

Fig. 3.1 conceptually depicts the PARRHI framework, which attempts to fulfil the given requirements and consists of three main parts. The left-hand side shows the developer, the grey box in the middle is the PARRHI system itself, and the right-hand side visualises the environment that surrounds the PARRHI system in both the virtual and the real space. Below, each component (from (0) to (8)) is systematically explained in detail, before describing the information flow between them.

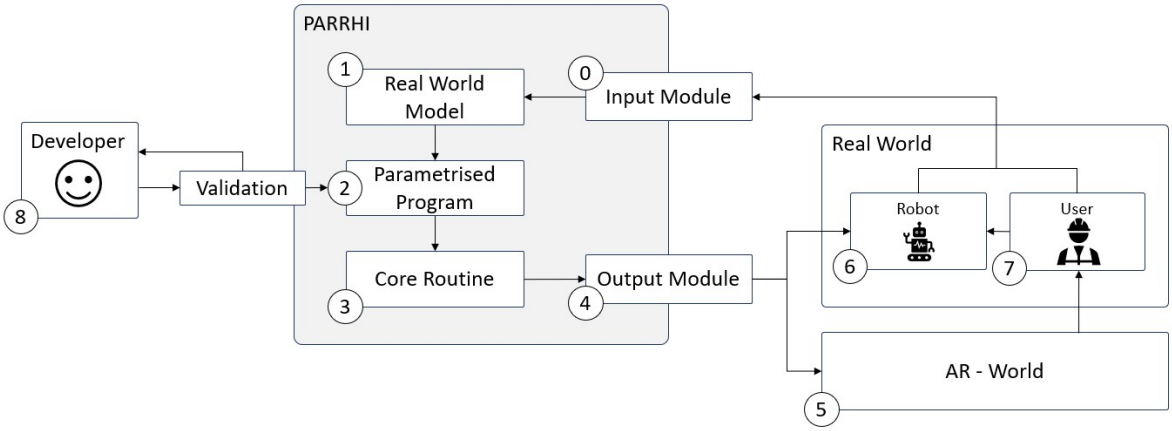

Figure 3.1: PARRHI general framework

0.) Input Module: The Input Module (0) is responsible for collecting the data needed from the real world. In this specific case, it receives the robot's (6) joint angles and the user's (7) position.

1.) Real World Model: Since the data from the Input Module (0) might not be in a usable format straight away, it has to be processed first. The Real World Model (1) has a

deeper understanding of the real world and helps extracting useful information from the data inflow. This model understanding still has to be implemented into the PARRHI system but could be enabled for parametrisation at a later stage. In this specific case, its outputs are the robot's (6) joint positions and the user's (7) location represented as parameters that are available to the Parametrised Program (2).

2.) Parametrised Program: This is a document that defines the AR application's behaviour and workflow. Its syntax is parametrised, meaning, that it makes use of placeholders (parameters) whose actual value is managed at runtime by PARRHI itself. There are two types of parameters: First, there are the ones that are provided by the Real World Model (1) (for example the robot's (6) joint positions), and second, any object defined in the Parametrised Program itself (2) acting as a parameter for other objects in the program. These objects can be 3D AR-Hologram definitions, logical instructions, or any variety of events and actions. For a more thorough explanation of the Parametrised Program see section 3.2.2.

3.) Core Routine: The Core Routine (3) is an interpreter for the Parametrised Program (2), and acts on its instructions. It generates the output of the application, which can either be commands to the robot (6) and thus the real world, or instructions for the AR-World (5).

4.) Output Module: This component manages the outgoing communication with PARRHI's environment.

5.) AR-World: The Augmented Reality World is the space, where the AR part of PARRHI's output is displayed. It can contain holograms like spheres and cylinders, but also written text for instructions. The AR-World (5) augments the Real World and is thereby seen by the User (7) through an AR Device.

6.) Robot: It is a Real World object that can be controlled by the User (7) themselves, or by the PARRHI system.

7.) User: This is the person that uses the finished application.

8.) Developer: This person develops the application's workflow and behaviour. The developer's output is the Parametrised Program (2) in the form of a document in PARRHI syntax, which is validated before being fed into the PARRHI system.

The following paragraphs track the information flow between these components. Starting with the Developer (8) crafting the Parametrised Document it describes the standard information cycle of the PARRHI framework.

After having defined the objectives of the AR-HR-Interface application, the Developer (8) crafts the Parametrised Program (2). At this point, the Developer inputs his or her domain-specific knowledge and expertise into the system so that other people like the User (7) can benefit from it. The document is validated before it is granted access into the PARRHI system. If the validation fails, the Developer (8) receives error messages accordingly and can iterate the document until its validation is successful and the AR app behaves as desired by the Developer (8).

The runtime loop starts with the Input Module (0). It not only collects the robot's (6) joint angles, location and gripper-state but also the User's (7) position. This input data is then fed into the Real World Model (1), which uses its Real World understanding to extract/calculate more useful information. In this specific case, it applies forward kinematics to the robot's (6) joint angles to calculate their 3D position and transforms the User's (7) location coordinates into PARRHI's internal coordinate system.

After transforming the input data into a usable format, the placeholders (parameters) in the Parametrised Program (2), are now filled with information by the PARRHI system. For example, the Parametrised Program (2) could use the robot's tool centre point (TCP) for a definition of some AR-holograms, without knowing where exactly the TCP is during development. At runtime, PARRHI inserts this information it received from the Real World Model (1) into the parameters.

The Core Routine (3) then interprets the now filled-in Program (2). It first updates its internal state with the new parameters, before updating all AR-Holograms. Finally, it evaluates all events and triggers their actions if needed. These actions might be commands for the AR-World (5) (for example to show and hide holograms or to change UI text) or commands for the Real World Robot (6) (e.g. to move or stop the robot). These commands are passed on to the Output Module (4), which is responsible for executing them. It has the required tools to communicate with the Robot (6) and with the AR - World (5).

At this point, the User (7) sees two items. They see the Real World, with the Robot (6) and its surrounding environment in front of the user. The Robot (6) might have been instructed to do something by the Output Module (4) already. Superimposed onto the Real World, they also see the Augmented Reality World (5), which contains all visual information that the Output Module (4) constructed. For example, there could be cylindrical translucent holograms marking a danger zone around the robot's (6) axes. The AR-World's (5) content influences the User (7) to do certain things. He or she could be instructed by holograms to move into a safe-zone. At the same time, the User (7) might control the Robot (6). The

Output Module (4) enjoys priority over the User's (7) input when commanding the Robot (6), ensuring its the safety if it needs to stop the robot.

As Fig. 3.1 depicts, the Input Module (0) then finally closes the feedback loop by receiving fresh data from the Real World. The information flow that was followed in the paragraphs above, changed the real world either by direct commands to the robot or indirectly by commanding the User (7) via the AR World (5). These changes will now be reflected in the input data and are thus available for the next cycle. The presented loop repeats itself, where every iteration only takes a fraction of a second until the PARRHI system is terminated for some reason.

The preceding paragraphs have given an overview of the PARRHI framework. The following chapters explain each part in an even higher degree of detail. The order follows the explanation of the information flow in the PARRHI concept.

3.2.1 Preparation of Parameters

One main advantage of the PARRHI framework is the disclosure of its knowledge of the outside world to its inner components via parameters. Two steps are necessary to achieve this. First, the Input Module (0) has to automatically retrieve the real world information. Since the latter is hard to comprehend for computer programs and might not be usable for the Developer's purpose, the Real World Model (1) processes the input data in some way. Since the preparation of these parameters is an essential part of this concept, the following section will explain their origin, and how they are being prepared for the Parametrised Program.

The complexity of collecting real-world data strongly depends on the use case. I decided to limit my scope to the collection of three pieces of information since it is not the main focus of this thesis. These three pieces are the user's position, the robot's joint angles, and its gripper state which are then fed into the Real-World Model and later offered as parameters to the Parametrised Program.

To get the user's position, the relative distance and orientation relative to the robot have to be retrieved first. For AR applications, it is important for the AR device to know its six-dimensional orientation (position and rotation). Otherwise, superimposed holograms would not make any sense to the viewer since they appear misplaced. As soon as misplacements of visual augmentations happen, they are more a distraction than an assistance. To get this relative position, the robot's position has to be received first. This could be done via image

tracking. From this vector (user to robot) PARRHI calculates the user's location in the robot's coordinate frame with its model of the real world.

Then the robot's joint positions are needed for the application program to offer the possibility to fully integrate the robot in the application's workflow. Since most robots do not offer their individual joint positions in 3D vectors, but only their joint angles, a corresponding robot model is needed to calculate each joint's position from their joint angles. This is called forward kinematics, a well-known challenge in robotics with known mathematical tools and ways to solve it (at least in the case of basic industrial robots). In principle, a robot's tool point can be calculated via every joint angle and some knowledge of the robot's configuration. The latter is specified by the types of joints (degrees of freedom (DOF)) and the distance between consecutive joints. A mathematical model then computes each joint's three-dimensional location, which is offered to the Application Program (2) via parameters. To get an example of how the joint's position might be used in a parametrised way see section 3.2.2.2.

3.2.2 Parametrised Program

The Parametrised Program is the document the Developer ((8) in Fig. 3.1) of the PARRHI system produces. It contains parametrised, hierarchically structured data that defines the behaviour, look and feel of the final AR-HR-application. This document is responsible for fulfilling Requirement #6 in section 3.1.1, which requires the PARRHI system to allow complex application workflows to be modelled.

The Parametrised Program contains instructions and definitions which from now on will all be called "objects". All these objects have a certain set of parameters that are needed to fully define their functionality, visual appearance and behaviour. These parameters can either be previously defined objects, data from the Real World Model, or constant values. By using other objects as parameters, the Developer can create an interconnected program with links to other components, where instructions might be able to manipulate other objects in the program at runtime. To reference these other objects, every single object has to have a unique name or ID.

The following chapters describe the set of tools that are available to the Developer and how they work and interconnect. Table 3.1 displays the different categories of objects available in PARRHI. Each section describes what exactly is parametrised in their definition and how they can be used as parameters to define other program parts.

Table 3.1: Parametrised Program structure

Name	Section	Explanation
Variables	3.2.2.1	Integer variables in a traditional sense
Points	3.2.2.2	Different kinds of 3D Point definitions (fixed, relative to the robot, relative to the user)
Holograms	3.2.2.3	3D virtual augmentations like spheres and cylinders
Events	3.2.2.4	Tools for logic operations to define workflows

3.2.2.1 Variables

Variables are storage locations for numbers and have a symbolic name. When using Variables as parameters, they can be the input and target for instructions and thus take part in the application's logic.

3.2.2.2 Points

Points essentially are three dimensional vectors (X, Y, Z), with their coordinates being defined using parameters. Depending on the type of point, different parameters for the definitions are used. There are three types of points.

1.) Fixed-Point

2.) Robot-Point

3.) Camera-Point

Points are probably the best example of parametrised information in the PARRHI system. At runtime the data from the Real World Model (see (1) in Fig. 3.1) is directly fed into the definition of all points that use the respective parameters. Thus, the system updates these objects repeatedly with Real World information fed through the Real World Model. Although points are defined using parameters in some way, points themselves are parameters to other objects in the Parametrised Program.

The **Fixed-Point** has static coordinates, and thus fixed values are used to define their coordinate parameters (see figure 3.2). It could be used to set up holograms that visualise

certain spacial, environmental constraints that do not move.
Definition Parameters: *name : string, X : float, Y : float, Z : float.*

Robot-Points are defined by two indices of the robot's joints and one scalar value (see figure 3.3). The application's developer does not have to understand the robot's kinematics and simply uses the joint-indices as parameters. At runtime, the *PARRHI* system retrieves the robot's joint position, uses the Real World Model to calculate each joint's position ($J_0 - J_5$) and then assigns this data to the Robot-Points. The final point's position P is calculated as follows (with s being the scalar value and J_n the position vector of Joint n):

$$\boldsymbol{P} = \boldsymbol{J_1} + (\boldsymbol{J_2} - \boldsymbol{J_1}) * s \tag{3.1}$$

As can be seen, the scalar value linearly interpolates between the two joint positions and thus allows for a wider variety of Robot-Points.
Definition Parameters: *name : string, J1 : int, J2 : int, Scale : float.*

Camera-Points are a way to involve the user's position in the application. Similarly to Robot-Points, the PARRHI system repeatedly retrieves the camera's coordinates via the Input Module, feeds it through the Real World Model to map it onto the internal coordinate systems and finally updates the Camera-Points with the new position data of the head mounted AR-Device. Since there is only one Camera in the PARRHI system, its only parameter is the point's *name : string.*

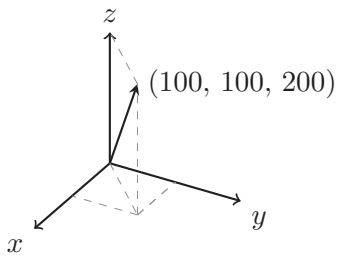

Figure 3.2: Fix-Point example

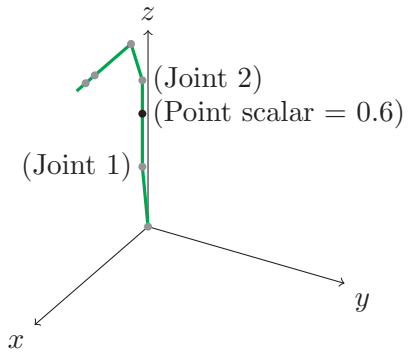

Figure 3.3: Robot-Point example

3.2.2.3 Holograms

The AR World can be filled with numerous holograms. In the Parametrised Program these holograms are defined using parameters. In this case, Points from section 3.2.2.2 are used to define the Hologram's location and orientation. Since Points are parametrised themselves and are updated by the data from the Real World model, holograms are indirectly updated regularly.

Holograms in the PARRHI system have a number of attributes (see table 3.2). Currently, *PARRHI* supports two types of holograms. There are spheres and cylinders, taking one or two points respectively and a radius as input parameters to define their size and position. Other attributes describe their visual appearance.

Table 3.2: Hologram attributes

Attribute	Possible values	Explanation
name	any text	Used to identify the object
visibility	visible, hidden	Sets the hologram's visibility
renderMode	normal, transparent	normal: Hologram is rendered as a solid object transparent: Hologram is rendered translucent
radius	number	The radius of the sphere or cylinder
point1	name of any defined point	Used for the holograms position definition
point2	name of any defined point	Used for the cylinder's position definition (not available for spheres)

A Sphere's centre is always equal to its point of definition (*point1*), whereas a cylinder always connects the two points of its definition (*point1* and *point2*). All types of points described in section 3.2.2.2 can be used as input parameters for the location. This is the first example of Parametrised Program's internal use of objects as parameters.

Holograms have an attribute called *renderMode*, which if set to "transparent", renders the hologram in a half transparent way allowing holograms to be used for boundary or zone visualizations. Furthermore, the visibility of holograms can be changed by setting the *active* parameter, which is done by actions as described in section 3.2.2.4.

3.2.2.4 Events

All previous elements (variables, points and holograms) exist to define the scene and to set up assets that can be utilised by events, which now actually describe the application's workflow. There are two subtypes in this category. There are event-triggers and event-actions (or short *triggers* and *actions*). Triggers take on boolean expression, which is checked periodically. As soon as the boolean expression evaluates to *true*, the attached actions will be executed and the trigger will be disabled, avoiding multiple executions. One could say that if triggers are *PARRHI's* sensors, then actions are its actuators.

Although there are different types of triggers, all their definitions have some attributes in common, which are listed in the table 3.3. Of course, specific trigger types have additional parameters to define their functionalities.

Table 3.3: Trigger attributes

Attribute	Possible values	Explanation
name	any text	Used to identify the object
canTrigger	true, false	Enables/Disables the trigger
actions	multiple names of actions	Defines the payload of the trigger

To reach a reasonable capability, three different and easy-to-understand triggers are available to the application's developer. In table 3.4 is a complete list of all defined triggers.

Distance triggers are mainly defined using two Points and a distance value. They trigger as soon as the Euclidean distance of the two points is below the distance parameter. They can be used for two main purposes. First, the application can use the robot's movement as an input using a *Robot-Point*. An action could be triggered as soon as the robot's TCP moves near a desired target point by using a *Fixed-Point*. Second, the user's movement can

Table 3.4: Event Triggers

Name	Input Parameters	Trigger expression
Distance trigger	Two Points $\boldsymbol{P_1}, \boldsymbol{P_2}$, distance d	$\lvert\boldsymbol{P_2} - \boldsymbol{P_1}\rvert \leq d$
Variable trigger	Variable v, trigger value v_{tr}	$v = v_{tr}$
Time trigger	trigger time t_{tr}, time since enabling $t_{enabled}$	$t_{tr} \geq t_{enabled}$

be monitored by using a *Camera-Point* as an input parameter. The application can thus ask the user to move to a specific location (e.g. a safe-zone).
Definition Parameters: *name:string, canTrigger:bool, point1:Point, point2:Point, distance:float, actions:[Action]*

Variable triggers are defined using a variable parameter, and one *trigger value* parameter. They trigger, as soon as the given variable equals the *trigger variable*. A counter for certain events could be implemented, and trigger an action when a threshold value is reached. It can also be used for workflows that need states or steps.
Definition Parameters: *name:string, canTrigger:bool, varName:Variable, triggerValue:int, actions:[Action]*

Finally the **Time trigger** allows the application to involve timers. These triggers are defined using a *timeSinceActivation* parameter and trigger, as soon as the time passed since setting *canTrigger* to true is equal or greater than the *timeSinceActivation* value. If *canTrigger* is true from the beginning onwards, the timer starts immediately. For example, the user could be given a maximum time for a task, or holograms can be hidden after a few seconds.
Definition Parameters: *name:string, canTrigger:bool, timeSinceActivation:int, actions:[Action]*

Whenever a trigger's boolean expression evaluates to true, its actions are invoked and the trigger gets disabled. There are six different types of actions - each serving a specific purpose. As with triggers, actions have a set of input parameters they need to fulfil their task and to be completely defined. The table 3.5 gives an overview over all actions that *PARRHI* currently supports.

Increment-counter actions increment the value of the Variable parameter by 1. If a developer wanted to count the number of times a user jogged the robot into a specific

Table 3.5: Event Actions

Action Name	Input Parameter	Explanation
Increment Counter	Variable v	Increments the value of v by 1
Set Hologram State	Hologram-names, State to set	Enables/disables all specified holograms
Set Trigger State	Trigger-names, State to set	Enables/disables all specified triggers
Change UI Text	Text to set	Sets the UI Text
Move Robot	Point P	Moves the robot to P
Set robot-hand State	State to set (open/close)	Opens or closes the robot's gripper

region, an Increment-counter action could be used as a payload of a Distance trigger. After a threshold value is reached, a Variable trigger could change the UI text and display a hint.
Definition Parameters: *name:string, intVar:Variable*

The **Set-Hologram-State action** allows the visibility of holograms to be changed at runtime. For example, if a hologram represented a region for a tutorial step, it could be hidden after the user's task was completed. The new scene could then be set up by displaying new holograms that guide the user's way. Another possible scenario would be displaying a warning boundary, if the user moves into a forbidden zone.
Definition Parameters: *name:string, onHolograms:[Hologram], offHologram=[Hologram]*

When using PARRHI applications the user is presented a GUI that shows text. The **Change-UI-Text action** allows changing this displayed text. There are numerous obvious scenarios where this is useful. Whenever it is of value to inform the user about something that cannot be achieved by holograms, this is a simple way to do so.
Definition Parameters: *name:string, text:string*

To create meaningful and longer applications, enabling and disabling triggers is an essential tool. This is what the **Set-Trigger-State action** is for. Triggers can only invoke their payload actions, if their attribute *canTrigger* is true. Triggers can either be defined as disabled from the beginning or automatically get disabled by triggering. The Set-Trigger-State action enables or disables the trigger, which is referenced by the *triggerName* parameter. There is a speciality in the case of *Time triggers*. Their inner timer starts ticking, whenever they get enabled. This allows for timers to be used in the middle of applications, relative to other events.
Definition Parameters: *name:string, triggerName:Trigger, canTrigger:bool*

The last two actions are designed to take active control over the robot. The **Move-Robot action** moves the robot to the specified coordinates. It is important to note that this action has two different modes defined by a `mode` parameter. If *mode* is *'t'*, then the coordinates will be interpreted as absolute euclidean task space coordinates, and if *mode* equals *'j'* then PARRHI assumes that the coordinates are defined in joint-space and will act accordingly. Instead of defining the target parameter by a six-dimensional vector, a *Point* reference can also be passed into this parameter.
Definition Parameters: *name:string, target:Vector6, mode:char*

The **Set-Robot-Hand-State action** behaves as one would expect and opens or closes the robot's gripper according to its parameters *state*.
Definition Parameters: *name:string, state:bool*

3.2.2.5 Program example

To further explain the parametrised program, I want to give a short example of how such a small program might be set up. The following program will wait for the user to move the robot's tool centre point (*TCP_Point*, line 3) to the *Target_Point* (line 2) at some static coordinates and visualise both points with Sphere holograms (lines 6,7). Upon reaching the target with a certain tolerance (line 10), it will display a success message to the user (line 13).

```
1  <!-- Define Points -->
2  <PointFix name="Target_Point" X="200" Y="200" Z="200" />
3  <PointRobot name="TCP_Point" J1="6" J2="6" scale="0"/>
4
5  <!-- Define Holgorams -->
6  <Sphere name="Target_Area" radius="15" point="Target_Point"/>
7  <Sphere name="TCP_Area" radius="15" point="TCP_Point"/>
8
9  <!-- Define Trigger -->
10 <DistanceTrigger name="d_trigger" point1="TCP_Point" point2="Target_Point"
       actions="SetUI_Success" distance="15"/>
11
12 <!-- Define Action -->
13 <ChangeUITextAction name="SetUI_Success" text="You have reached your target
       successfully!"/>
```

3.2.3 Core Routine

The Core Routine's responsibility is to interpret and execute the Parametrised Program. Its input is the Parametrised Program with all data from the Real World Model already set, and it's outputs are commands to the Real or AR World. At the very beginning of each cycle in the Core Routine, all Parameters are resolved (see (0) in Fig. 3.4). This means that the interpreter finds all parameters used in the Parametrised Program, and inserts all values from the referenced objects. For example, if a hologram is defined using a Point object, the interpreter looks for the point and fetches its momentary coordinates to fully define the hologram. Similarly, all other parameters are found and processed.

After that, the Core Routine updates its inner state (1). This means that it updates all Holograms and Triggers with newly received data. Holograms are moved to their new position and Triggers receive their new input values depending on the Trigger type. Distance Triggers, for example, receive the new coordinates they monitor.

With all values updated, the core routine can now check all *Triggers* whether or not their boolean expression evaluates to *true* (2). Whenever a *Trigger* does evaluate to true, it invokes all *actions* of that particular trigger(3). The executing *actions* result in a group of commands, that will be sent to the Output Module, which then handles them going forward.

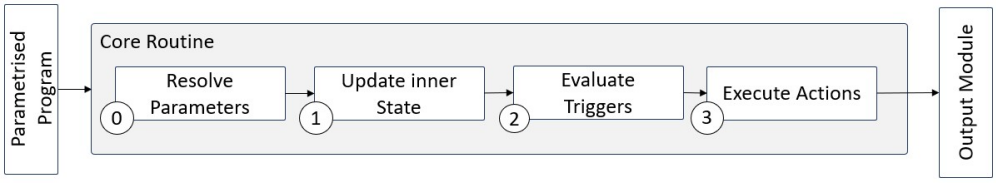

Figure 3.4: Core Routine workflow

3.2.4 I/O Modules

The last step in the PARRHI runtime cycle is to send all generated commands to their receivers. As described, there are two types of commands. Firstly, there are the ones that are directed to the AR-World, and secondly, there are commands that a directed towards the Real World. I will now briefly explain the difference in their nature, and why AR-commands are much easier to process conceptually than Real World commands.

The AR-World is *only* software that runs on the same device, which hosts the complete PARRHI system. That means, that sending commands to the AR-World is as simple as actually calling some methods in the back-end. Real World commands, on the other hand, are much more complex because they have to be sent to other hardware devices, possibly containing different operating systems (OS) that run on different programming languages. This is why some network protocols have to be defined to communicate with these Real World objects.

In this case, the Robot (see (6) in fig. 3.1) is the only Real World object, that does not run on the same OS as the AR-Device. This means, that an OS neutral protocol has to be defined for this type of communication. Additionally, operating systems on robots tend to be sealed towards the outer world securing them against unwanted access but, at the same time, are hard to interconnect to other systems. To get more information about the implementation regarding this component see section 4.5.

4 Implementation

This chapter documents the implementation of the PARRHI framework and begins by briefly describing the hardware and software main-components before going into detail in certain areas of interest.

4.1 PARRHI Hardware

For the Augmented Reality purposes, the first edition of Microsoft's Augmented Reality glass called HoloLens [25] was used. The HoloLens has all necessary capabilities like cameras for image and environment tracking, simple hand-gesture recognition, a reasonably good field of view and was programmable with the 3D game engine Unity. Thankfully, the TUM chair of "Automation and Information Systems (AIS)" supplied me with this product.

The second main hardware component in the PARRHI system is the Robot. The robot manufacturing company Fanuc [26] was gracious enough to gift the TUM chair "Automation and Information Systems (AIS)" a CR-7iA/L collaborative, 6 DOF, industrial robot [27] and an R-30iB controller [28]. The R-30iB controller can either be programmed in Fanuc's TP-Language or Fanuc's KAREL language. KAREL offers many possibilities with network sockets, reading and writing data from and to the robot's controller, whereas TP programmes are fit for moving and steering the robot.

The last component is a gripper provided by SCHUNK. The company gave the chair AIS collaborative Co-act EGP-C [29], that seamlessly integrated onto the Fanuc robot with a hardware interface for exactly that purpose.

At this point, I would like to thank Fanuc, Schunk and the AIS massively for their supplies. Without these components this thesis' implementation would have never been possible.

4.2 PARRHI Software

The PARRHI framework's software implementation is split into three components: The PARRHI library, the Robot library and the Unity engine. The PARRHI library implements the PARRHI framework's logic (see Fig. 3.1), while the Robot library has the tools necessary to communicate with the Robot in the Real World. The third component *Unity* hosts the PARRHI and Robot libraries.

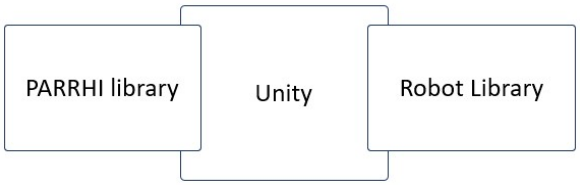

Figure 4.1: PARRHI software implementation

4.3 PARRHI Library

The PARRHI system was programmed in Microsoft's .NET framework version 4.6.1 [30] using *C#* [31]. This decision was made because the Unity game engine is scripted in the language *C#*. The PARRHI library itself is a *Class Library* project [32] and has three main tasks.

1.) Import the Parametrised Program and perform some operations on it. The Parametrised Program is a XML [33] based document. After importing, it is validated using an XSD [34] document. Finally, the PARRHI library has to extract useable *C#* objects from the deserialised XML document and set up references to resolve parameters.

2.) The Real World Model and the logic behind it is a subcomponent of this library. In my specific case, this means, that the PARRHI library contains the Robot's forward kinematics model and the required information to convert the HoloLens' internal coordinates, into the robot's coordinate system.

3.) The Core Routine is located in the PARRHI library, meaning, that on every iteration the PARRHI system calls into the PARRHI library to perform its task.

This library's main interface to the hosting environment is a container object, that contains all Parametrised Program object references like Holograms, Points, etc., but also manages

the importing of the Parametrised Program document itself. Fig. 4.2 is a class diagram of the `Container` object. As can be seen, it holds references to all objects of all types in the Parametrised Program and provides some update methods, that will be called from the library's host.

It is important to note, that all objects inherit from the `PProgramObject` class, which gives them the id property that allows the object to be identified. This is needed to resolve parameters in the Parametrised Program. As soon as an object is instantiated, it calls back to a list of id's which checks all ids for duplicates. In the case of duplicate ids, an error is thrown and displayed to the developer. Whenever an object is used as a parameter in the Parametrised Program, this id is used to find the reference to the target object.

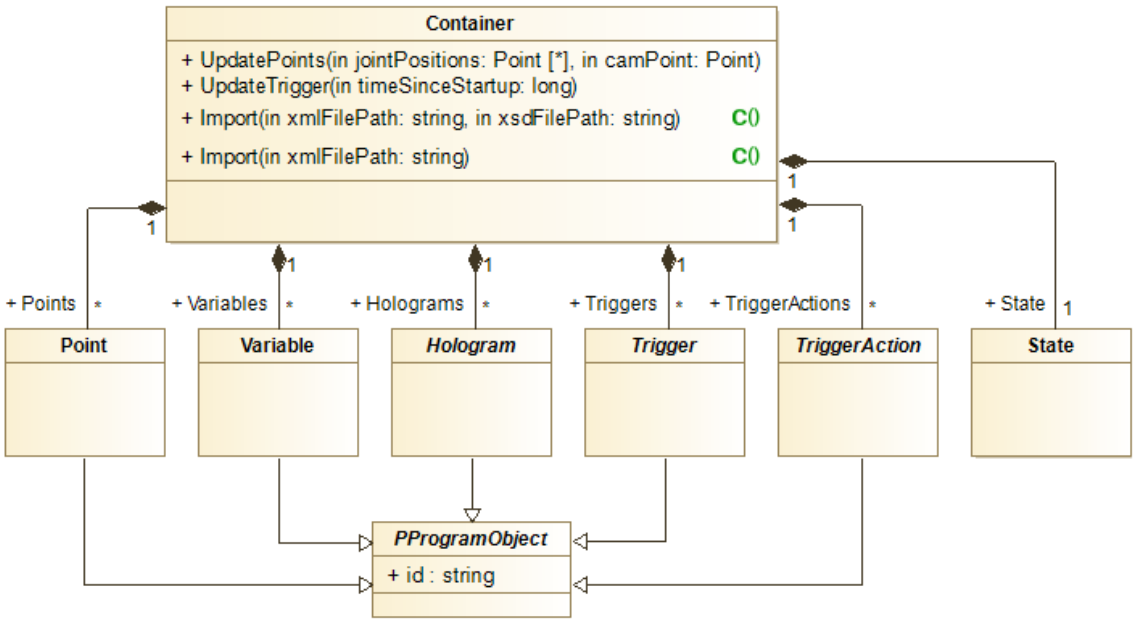

Figure 4.2: UML class diagram of the Container class

All compositions besides the `State` class simply represent the Parametrised Program objects presented in section 3.2.2. The `State` itself is an object, that holds the Real Worlds state data at all times, so that the PARRHI library has access to it. Fig. 4.3 shows that the State object is composed of two other objects - each serving a specific purpose. The `Robot` class handles all operations that have anything to do with the robot. For example the Forward Kinematics. It also contains two delegates. Instead of letting the Robot class have an instance of the robot library (section 4.5), I chose to let this class have two delegates. The PARRHI library invokes these two delegates, which will be configured from the hosting environment. This creates some independence from the Robot Library, and may allow the PARRHI library to be used with absolutely any robot as long as there is a .NET framework controller for it.

This is not the only advantage of abstracting the controlling of the robot. During the development of the PARRHI system, I wanted to simulate as much as possible since I did not always have access to the robot. So I created an interface between the PARRHI system and the real world, that was able to simulate all values instead of retrieving them from the real world. During simulations, this delegate can be configured to send the commands into the simulation and not to the real robot.

The `World` class has attributes that represent the state of the real world and discloses them to the PARRHI library. The `SetUIText` delegate similarly to the `MoveDelta` method from the `Robot` class, is a function pointer that can be configured from the library's host. This was done for the same reasons as with the `Robot` class - to create independence from the host.

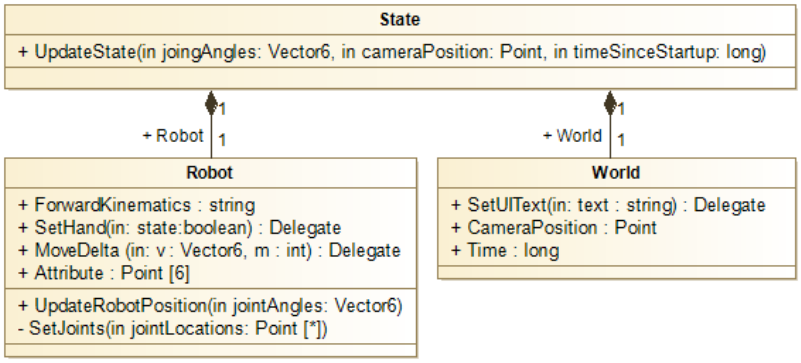

Figure 4.3: UML class diagram of the State class

These two classes are the main interface to the library's host. They contain multiple subclasses which will be explained in the following section.

4.3.1 Parametrised Program objects

The Parametrised Program contains five different objects: Variables, Points, Holograms, Triggers and Actions. During the implementation, abstract classes of each object type and then the concrete implementations for each sub-type were created. The PARRHI interpreter only utilises the public methods that are disclosed in the abstract classes. This approach was chosen because this way it is not important how the abstract class was instantiated, but only what values they hold.

This approach also allows the PARRHI system, to use any type of abstract object as a parameter for the definition of other objects. The abstract classes provide all data needed

for the object's definition where it is referenced from. It is not important which concrete implementation the given instance is. This section will explain how these objects were implemented and how the inheritance tree is set up.

The Variable class is very straight forward and only contains an integer value. There is also only one type of it, which makes it even simpler.

The `Point` class, on the other hand, is a bit more interesting. The abstract `Point` class contains the three-dimensional position vector and some read-only properties, but the inheriting classes define how the position is set. For example the `PointRobot` class takes two joint indexes for its definition as it was described in section 3.2.2.2. In each cycle, the public method `UpdatePoint` is invoked, which allows the `PointRobot` to calculate its new position. The `PointCamera` and `PointFix` do not add or overwrite any methods, since the `Point` class contains all needed functionalities. For clarity, some less important attributes and methods were omitted in these class diagrams.

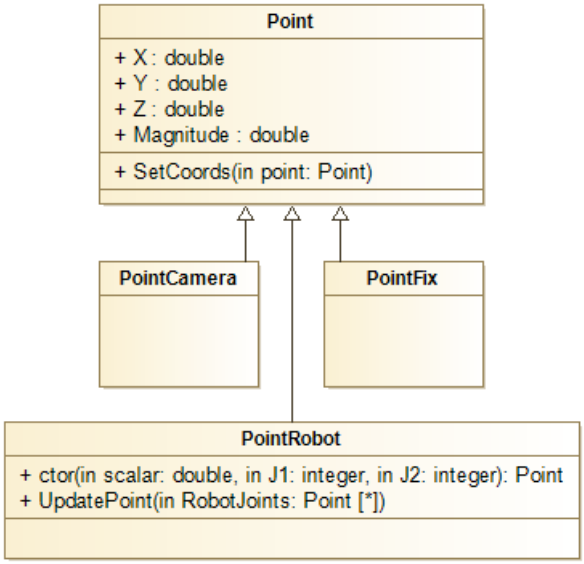

Figure 4.4: UML class diagram of the Point class

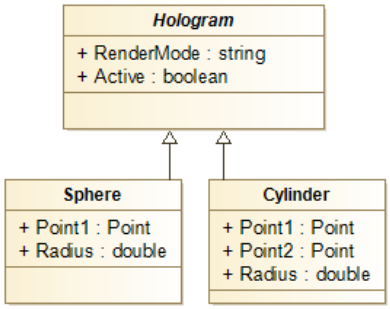

Figure 4.5: UML class diagram of the Hologram class

The same principle was applied to the `Hologram` class (see Fig. 4.5). The two inheriting classes `Sphere` and `Cylinder` provide their specific constructors and hold their values. Since the `Point` instances in the two different Holograms are references to the real `Point` object, and the way references work, they do not have to be updated each cycle specifically. Only the hosting environment (in this specific case the Unity Game Engine) has to update the 3D objects it generates.

The `Trigger` class (Fig. 4.6) is structured in a similar way. The abstract class `Trigger` has a virtual method called `CheckTrigger`, which evaluates the trigger expression. Since every inheriting class triggers on different data, this method cannot be implemented in the parent class. As it was with the `Point` class, the different inheriting classes have different constructors and store different data to evaluate their triggers. The `DistanceTrigger` for example takes two references to `Point` objects. It is important to note, that the Point object can be any object, that inherits from the `Point` class. This means, that the `DistanceTrigger` can be constructed with `PointCamera`, `PointRobot` or `PointFix`. Each inheriting class provides the `CheckTrigger` implementation, which corresponds to the defined boolean expression in section 3.2.2.4.

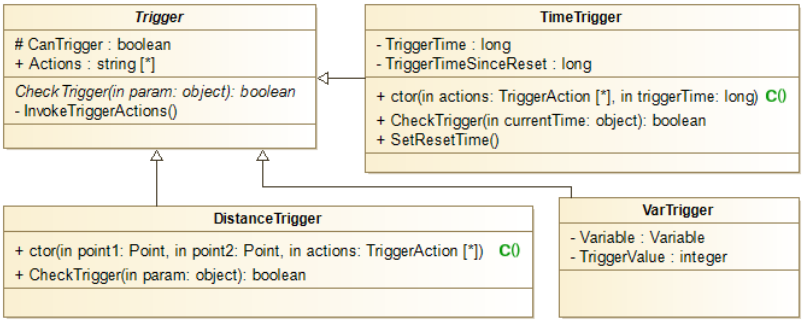

Figure 4.6: UML class diagram of the Trigger class

The last Parametrised Program object is the `Action` class. In this case, the abstract parent class is not as powerful as in the previous classes. The reason simply is, that all action types are fundamentally different and do not have anything in common, besides one method called `Action`. Figure 4.7 depicts the corresponding class structure. Each inheriting class has to implement the virtual `Action()` method itself since the parent object does not have the necessary data.

There are some interesting aspects to this implementation. As described, the `World` object holds a function pointer (delegate) to a method, which prints a given string to the corresponding AR output. The `ChangeUITextAction` takes a reference to the `World::SetUIText(...)` delegate. This way, the hosting environment has to configure the UI delegate only once and it will be applied everywhere in the PARRHI library.

Similarly to the `ChangeUITextAction`, the `MoveRobotAction` holds a reference to the `Robot::MoveDelta(...)` delegate. In this specific case, the class has two different private attributes, which both save the position or references to the point which will return a position that should be sent to the robot. This is the case because there are two different modes this Action can be used. In one case, the input can be a reference to a `Point` object and in

the other case, a six-dimensional `Vector6` object. Remember, that the used Fanuc Robot is six-dimensional. Since `Point` objects are only three dimensional, the robot will then approach the coordinates of the `Point` object from above, with the gripper facing downwards. The predefined orientation provides the last three needed coordinates, which completes the command and makes it executable. If a `Vector6` object is given, then the command can be sent without adjustments.

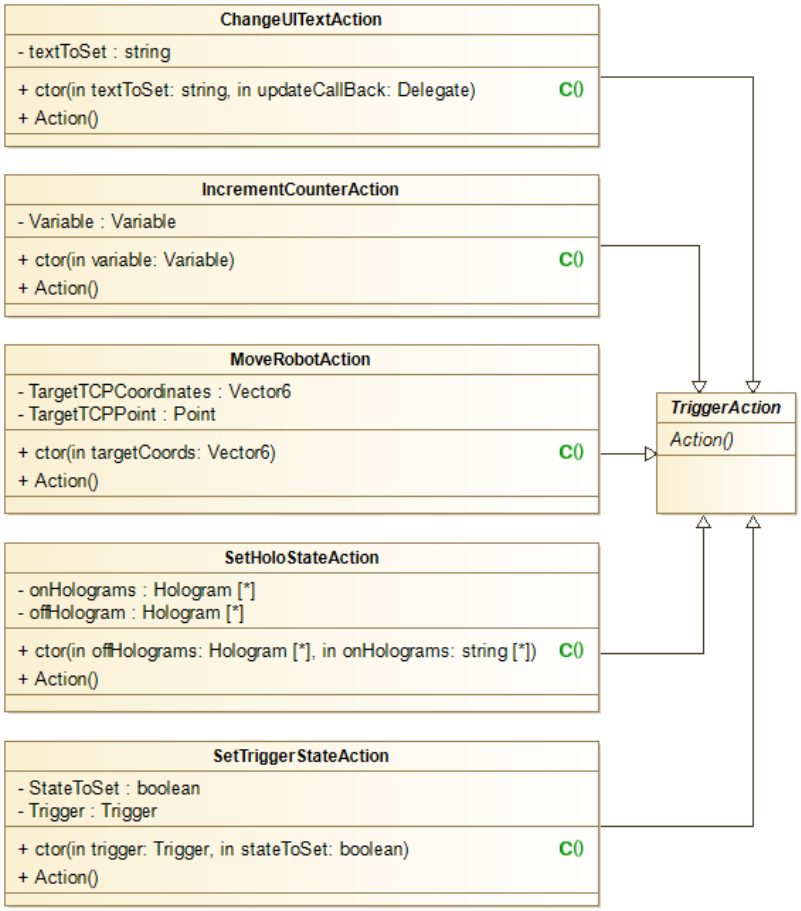

Figure 4.7: UML class diagram of the Action class

4.3.2 Importing and interpretation of the Parametrised Program

Importing and validating the XML document is easy since the used .NET framework natively supports that. There is an API within the .NET framework, which takes an XML and XSD file, and validates the XML file. A provided deserialiser then returns an object that represents the XML data.

The implementation of the PARRHI Core Routine is also straight forward. The container object first updates all `PointRobot` objects. It then loops through all its `Trigger` objects and checks their boolean expressions. If the trigger's `canTrigger` value and the mentioned expression both equal `true`, then the Core Routine invokes all actions from the trigger in question. If an action targets another object from the Parametrised Program (such as `SetTriggerStateActions` do with triggers) the action can directly be performed. `MoveRobotActions`, on the other hand, have to be sent to the Robot Library which will be explained in section 4.5.

4.3.3 CR-i7A Forward Kinematics

In robotics, a robot can be described in two different spaces. On one hand, there is the joint space, meaning that all vectors and matrices depend on the joint positions, velocities or accelerations. Since the robot has six joints, the joint space is six-dimensional. In the joint space, the position vector is often represented as $\vec{q}$.

On the other hand, there is the task-space. In this space, all vectors and matrices depend on the Cartesian position and orientation and their first two derivations of the robot's TCP $(\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}})$. If the joint space has six independent DOF, then generally speaking the task space also has six DOF and is represented as follows, with x, y, z being the 3D coordinates, and α, β, γ being the orientation of the tip.

$$\vec{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \end{pmatrix}, \vec{x} = \begin{pmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{pmatrix} \; with \; \vec{x}, \vec{q} \in \mathbb{R}^6$$

The forward kinematics now describes the process of transitioning form $\vec{q}$ to $\vec{x}$. This is done as follows: $\vec{x} = f(\vec{q})$ $with$ $f : \mathbb{R}^6 \to \mathbb{R}^6$. Where the mapping f depends on the robot's specific configuration and geometric properties. For reference, if f is a mapping from $\mathbb{R}^6 \to \mathbb{R}^6$ then f^{-1} (if it exists!) with $\vec{q} = f^{-1}(\vec{x})$ is called inverse kinematics.

To calculate f, I first defined local Cartesian coordinate systems after each joint. In Fig. 4.8 the black vector packs of three represent the coordinate systems $K_i = (x_i, y_i, z_i)$ and $\vec{l_i}$

represent the axes defined in the coordinate system K_i. In the following paragraph, the notation of $\vec{l}_{a,b}$ will be used to name the a^{th} vector called $\vec{l}$ expressed in the coordinate system K_b.

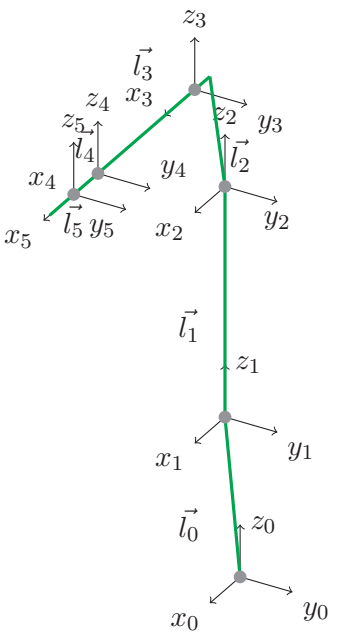

Figure 4.8: Robot forward kinematics coordinate systems

The matrix (ϕ_{n+1}) allows the conversion from vectors expressed in the coordinate system $n+1$ to the expression in system n, with $\vec{l}_{n+1,n} = \phi_{n+1}(q_{n+1}) * \vec{l}_{n+1,n+1}$. After transforming the vector $\vec{l}_{n+1}$ one can add the vector $\vec{l}_{n,n}$ because they are now both defined in the same coordinate system. Meaning, that the position of the tip of the vector $\vec{l}_{n+1}$ relative to the origin of the coordinate system n is, $\vec{x}_{n+1,n} = \phi_{n+1}(q_{n+1}) * \vec{l}_{n+1,n+1} + \vec{l}_{n,n}$.
Repeating this process from $n = 5$ to $n = 0$, results in the robot's TCP position relative to the coordinate system $\vec{O}$, which is what we wanted. One has to remember, that only the matrices ϕ_i are variables, since they depend on q_i. The vectors $\vec{l}_i$ are constants. This means, that for each iteration one only has to calculate all matrices ϕ_i and can then calculate the positions (x_i, y_i, z_i) of each coordinate system relative to the base system $\vec{O}$.

Since the *C#* implementation of this algorithm is only about 20 lines of code (excluding the definitions of the matrices ϕ_i and the vectors $\vec{l}_i$), takes only about 4ns to run, and the results are all joint positions in Cartesian coordinates relative to a base system, I am quite happy with the outcome. If one wanted to add multiple robots into this mathematical framework, it could be done quite easily by adding one additional world-coordinate system, where all robot base systems are embedded with a constant translation.

Adjusting this model to other types of robots would be easy. If the adjusted joints are all 1 DOF and rotational, then only the rotation matrices ϕ_i have to be adjusted, and $\vec{l}_i$ still stay constant. If one included translational, linear joints, the vectors $\vec{l}_i$ would depend on some variables, but the mathematical process would stay the same.

Implementation wise one interesting fact is, that Unity (the game engine which hosts the PARRHI library) uses a left-hand coordinate system, whereas my robot forward kinematics was done in a right-handed system. This means, that the last step in the forward kinematics is, to transform all produced vectors into Unity's coordinate system, which is rather easy. Only the z and y coordinates of each joint position needed to be swapped.

For further reading into this topic see [35].

4.4 Unity

Unity is the host of the PARRHI and Robot libraries. Originally, Unity itself is a game engine that helps developers to develop, publish, maintain and market their games [36]. Within recent years the company behind Unity, called Unity Technologies, made efforts to become a de facto standard for AR and VR development.

There are a lot of third party products for Unity that helps the developer with image tracking and general AR interaction. The next paragraphs explain what libraries, and tools were used to handle hand gesture recognition and image tracking.

4.4.1 Image Tracking

For image tacking, I used the Vuforia Engine [37]. This engine had a rather simple setup in Unity and worked relatively seamlessly. Compared to other engines and libraries the Vuforia Engine first of all worked, has good documentation and a user-friendly configuration. This library is free of charge for non-commercial use.

The PARRHI implementation makes use of two image markers. One of them is attached directly at the robot's base and is used to synchronise the Real World with the majority of the AR World. Since the centre of origin of the robot was not at its base, but about 30 cm above the first joint, there is a small translation between the image markers position and the AR World's coordinate system. The second image marker is used to display the User's UI. The User is given a small sheet of paper with the said image on it, and the PARRHI system projects the User's tasks, some configuration and debug options onto it.

4.4.2 AR-Toolkit

Interacting with Augmented Reality objects is an essential part of AR environments. Fortunately, Microsoft's open-sourced Mixed Reality Toolkit [38] makes this an easy task. The library directly integrates into the Unity environment and allows easy hand gesture recognition (pinching) for clicking and pointing at objects in the augmented space. Since both the

HoloLens and the Mixed Reality Toolkit are developed by Microsoft, the library utilises all features and allows for easy integration.

4.5 Robot Library

Next to the PARRHI library and Unity, the Robot library is the big third component in the PARRHI implementation. The whole Robot library was built together with Florian Leitner, who is currently writing his bachelor's thesis about a virtual programming environment of industrial robots. Since his thesis is being written at this very moment, I cannot refer to any source of his. The following chapter will explain how we accessed the controller's data, transferred the data to the HoloLens and generally communicated with the robot.

The Robot Library is responsible for the communication between any system that is capable of executing .NET framework applications, and the robot's controller R-30iB (see section 4.2). It is important to know, that Fanuc's controllers are not easily controllable by non-Fanuc components. Our goal was, to wirelessly control the robot from external devices that run .NET framework applications. Since there were no "plug and play" solutions available, we came up with the system, that will be explained in the rest of this chapter. As it has been done in the concept chapter, I will first shortly explain each component before elaborating on the information flow between them. Finally, I will give some details about the implemented, custom protocol we use. Fig. 4.9 depicts how the system is setup.

4.5.1 Robot Communication Architecture

The architecture can be split up into nine individual components, ranging from the actual commanding application to the robot itself. This architecture is tightly linked with the concept of the PARRHI system, thus many components can be recognised.

1.) Application Logic: An application that wants to communicate with the robot. In my case the PARRHI system.

2.) Robot Library: A custom library written in *C#* that offers some wrappers around our custom protocol, which simply is a syntax we defined to transfer commands and data over the Network Socket. This protocol follows certain rules so that both the KAREL Host and the *C#* Library can parse them to extract the important information in a standardised way.

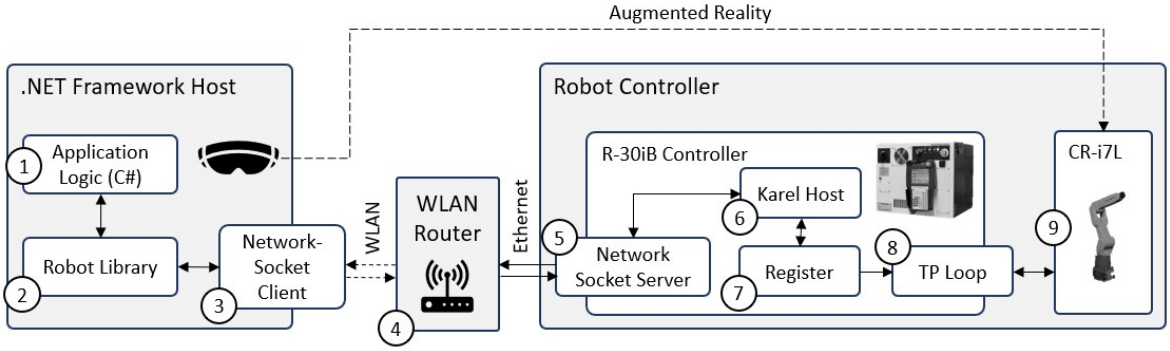

Figure 4.9: Robot library setup

3.) Network Socket Client: Standard Network Socket, where the PARRHI library represents the client.

4.) WLAN Router: Standard WLAN Router, which is connected to the controller via a standard patch cord.

5.) Network Socket Server: Standard Network Socket, where the R30iB controller represents the server.

6.) KAREL Host: A programme written in KAREL [39] that receives commands from the network socket, parses them according to the defined protocol and returns the requested data or executes the requested commands.

7.) Register: A storage that is used to command the Teach Pendant (TP) Loop (8), which also has access to this storage location. Some registers are used to set different modes like coordinate systems, absolute versus relative and so on, and six registers are used to transfer vectors, that the TP program should process according to the defined mode.

8.) TP Loop: A program written in TP that scans the Register (7) every cycle, and controls the robot accordingly.

As can be seen, that Application Logic (1) is the PARRHI itself. The network socket client(3) is a part of the concrete implementation of PARRHI's In- and Output modules and the complete Robot Controller is the "real world" excluding the end user.

Since researching the difference between KAREL and TP programmes on the internet is rather hard because one simply does not find a lot of information about it, I would like to summarise

our current knowledge about that. KAREL is a programming language, that is similar to Fortran. It was developed by Fanuc itself and is used internally for a lot of components in the controller. Nowadays, KAREL is most often used for managing tasks like communication, synchronisation, data handling and so on. Teach pendant programmes, on the other hand, are specialised on actually commanding the robot's movement and behaviour. TP supports numerous methods to control a robot, that the KAREL language simply does not offer. This is why we chose to split up the controller-side software into these two components.

The information flow is as follows: First, the application (1), which wants to send a command to the robot, has to call into the Robot Library (2). Besides managing the network socket connection (3), this library offers 12 commands to the application. A command could be the wish to fetch or set the robot's joint position. The Robot Library (2) then sends the command to the network socket (3), which is connected to the controllers server-endpoint. It is important that the communication between the HoloLens (or a Laptop for instance) is wireless to stay mobile. The controller is connected to the WLAN Router (4) via a cable using the Ethernet protocol. There, the network socket server (5) receives the command and passes it on to the KAREL Host (6). The latter then processes the command, which is not trivial and described in the following two sections.

4.5.2 Robot Communication KAREL Host

At the point of reception, the command is only a string following a very specific syntax that we defined. The KAREL Host (6) then parses the command and acts on its instructions. If the command requests data to be sent, the KAREL Host (6) collects the needed data, constructs the message and sends it back via the network. If the command is targeted at the robot itself, the KAREL Host (6) fills the Register (7) with all necessary data that the TP Loop (8) needs to execute the movement command. This data is a six-dimensional vector (coordinates in task or joint space), the mode (whether the vector should be interpreted absolutely or relatively to the current position), the coordinate system which should be used to interpret the six-dimensional data and some internal flags.

The KAREL code consists of four main parts. There is the network communication, the command parsing, the data collection/register writing and lastly the response sending. The KAREL language makes it possible to parse strings and write the logic components that are needed for the communication system. We tried to achieve as much as possible within the KAREL Host, and limit the parallel running TP Program to its speciality, which is controlling the robot physically.

4.5.3 Robot Communication TP Program

The TP Loop (8) itself is written in Fanuc's TP-Language. It is one large loop that reads the Register (7) and acts on its instructions. Some problems had to be overcome though. For example, switching between the relative and absolute movement mode has one very critical aspect: During absolute movements, the values in the Register (7) might be relatively high, since the task space coordinates are defined in millimetres. Values here are in a range from -700 to +700 mm. When interpreting these coordinates absolutely, that is not a problem since they are in the robot's range of motion. But after switching to the relative mode, these values are added to the current robot position, resulting in a huge step, which most probably is not within the range of motion anymore.

Another example here is the switching of coordinate systems. When the current data in the register was inserted to be interpreted as joint coordinates, the individual values are in the range from about -360° to +360° (varying for each joint). This range includes values close to zero. A limit error would occur if these were to be interpreted as task space coordinates suddenly, since Cartesian positions close to zero are not possible in the task space. Cases like these have to be thought of and handled.

Fig. 4.10 shows the TP Loop's Activity Diagram. Of course, this is a rather simplified version since almost every action depends on two register values, which represent the coordinate system and the relative/absolute mode.

Resetting the register's state at the very beginning is crucial since the data registers are persistent stores, which means, that their value is saved even when the controller is shut down. The initialisation defines clear behaviour at each launch of the program.

The *Launch ARobot_Server* action starts the KAREL Host, which communicates with any potential client via the network socket. After starting the KAREL Host, the program has to check if the relative/absolute mode has changed. If so, it handles this change to avoid unexpected behaviour due to old data in the registers. This is done via edge detection, which needs an additional variable to store the previous mode.

The action *Store Current Pos to PR* fetches the current coordinate vector in the corresponding system (joint space or task space), which is then fed into the next action, which calculates the new target position vector. This action differentiates between the relative and absolute modes.

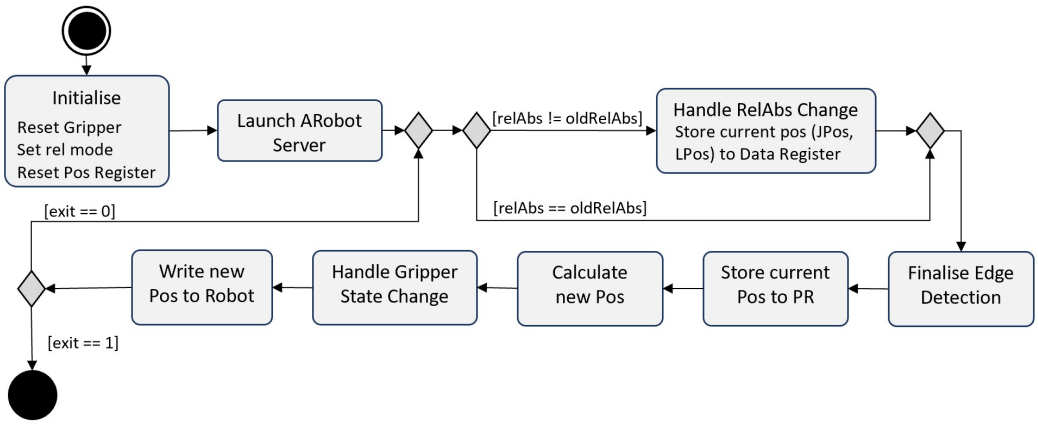

Figure 4.10: TP loop robot control

After the hand gripper state change is set, the new target position is written to the robot - again within the corresponding coordinate system. At the very end, there is an *exit* flag, which can be set to *'1'* by the KAREL Host. This quits the KAREL and TP Programm and thus allows the client (.NET library) to send an exit command to the robot. This allows controller to safely shutdown the network socket and the TP Loop.

4.5.4 Robot Motion Group Control Management

One important aspect to mention is the motion control system, which all FANUC robots are subject to. Within the FANUC environment, the robot can be separated into motion groups, which are groups of moveable elements within a FANUC robot. One motion group may, for example, contain the first three joints, and another motion group could own the second three joints. Or, one group can also contain all joints. At runtime, a motion group can be physically controlled by only one instance at a time. TP programmes, KAREL programmes and also the Teach Pendant are viewed as instances here. The reason for this is, to avoid cases, where undefined behaviour through double commanding might occur.

Since applications, which were developed in the PARRHI system, should both allow the user to jog the robot and then take over control programmatically through actions, the right to move the robot has to be managed between the two involved instances: The Teach Pendant and the TP Loop program.

The solution to this problem was, to implement a motion group control ownership management system. By default, the TP Loop would release the motion group control and not be

able to move the robot, unless it needs to execute a corresponding command. This means, that unless the TP Loop actively controls the robot, the Teach Pendant has the ownership over the motion group so that the user can jog the robot.

Implementation wise, this was achieved by a ”pause” flag in the data register. As soon as the KAREL program sets this pause flag to ”1”, the TP loop starts a second TP program called *”Waiter”*, which does not have any motion control rights. After launching the waiter program, the TP loop aborts its execution, which releases the motion control. This waiter program simply waits until the pause flag gets reset to ”0” by the KAREL program, and then restarts the TP loop. While the waiter program waits, the motion control rights are owned by the Teach Pendant, and thereby the user can take control of the robot. As soon as the PARRHI system has to control the robot, it activates the TP loop by setting the pause flag to ”0”, executes its movement command and then releases the motion control by setting the pause flag to ”1”.

4.5.5 Robot Communication Outcome

The presented architecture allows us to execute any pre-defined function in a fraction of a second. The whole process is so fast, that the PARRHI system fetches the robot’s joint position in each iteration.

To finish this chapter off, I would like to shortly explain what the system is capable of doing. The Robot Library can get the robot’s joint angles, TCP position and six-dimensional force/torque sensor data. It can command the robot to drive to an absolute or relative task or joint space position and set a speed vector for the TCP and joint vector. Finally, there are commands which release and claim the motion control. Generally, this system could be extended quite easily, by defining new commands in the robot library and the KAREL host.

4.6 Parametrised Program Implementation and Validation

One core component of the PARRHI implementation is the Parametrised Program. The document needs a well-defined structure since the content has to be imported (de-serialised). This automatic process only works seamlessly, if certain rules are kept perfectly. This is also why, it makes sense to validate the document before de-serialising it since the PARRHI system has some presupposed assumptions about the document.

XML [33] seems to fulfil these requirements. XML documents are strictly hierarchically and have the big benefit of being human and machine-readable. In PARRHI's case, this is exactly what the document in question should be. Written by a human with possibly limited software engineering skills, but interpreted by a machine.

4.6.1 Parametrised Program XML Structure

The content of the Parametrised Program was thoroughly discussed in section 3.2.2. In this section, I would like to present the actual implementation and how a Parametrised Program written in XML would look like.

XML documents must have one root element. In PARRHI's case this is the "<PProgram/>" tag. Similarly to the conceptual train of thought, the actual implementation contains four children.

```
1  <?xml version="1.0"?>
2  <PProgram xmlns="PARRHI">
3   <Variables>...</Variables>
4   <Points>...</Points>
5   <Holograms>...</Holograms>
6   <Events>
7    <Trigger>...</Trigger>
8    <Actions>...</Actions>
9   </Events>
10 </PProgram>
```

Each child element of PProgram can now take their object definitions directly. Only the "Events" element categorises its content in two sub-elements "Triggers" and "Actions". It is important to know, that every object defined in the Parametrised Program has to have a unique name attribute to identify it in other parts of the application.

The following block of code exemplary defines each object once. Please note, that this program is not written to do something meaningful, it only presents all ways to define objects in the Parametrised Program. Of course, the developer could use every object multiple times and also create multiple instances of the same object types, as long as their name is unique.

```
1  <?xml version="1.0"?>
2  <PProgram xmlns="PARRHI">
3   <Variables>
4    <Int name="Variable0">0</Int>
5   </Variables>
6   <Points>
```

```
7    <PointCamera name="CamPoint"/>
8    <PointFix name="FixPoint0" X="200" Y="-300" Z="300" />
9    <PointRobot name="RobotPoint0" J1="1" J2="2" Scale="0.6" />
10   </Points>
11   <Holograms>
12    <Cylinder name="Cyl0" point1="CamPoint" point2="FixPoint0" radius="10"
          visibility="hidden"/>
13    <Sphere name="Sphere0" point="RobotPoint0" radius="25" renderMode="
          transparent" visibility="visible"/>
14   </Holograms>
15   <Events>
16    <Trigger>
17     <DistanceTrigger name="DistanceTrigger0" canTrigger='true' point1="CamPoint"
           point2="RobotPoint0" distance="15.5" actions="HoloStateAction0"/>
18     <VarTrigger name="VariableTrigger0" canTrigger='true' varName="Variable0"
          triggerValue="2" actions="IncrVarAction0"/>
19     <TimeTrigger name="TimeTrigger0" canTrigger='true' timeSinceActivation="120"
           actions="TriggerStateAction ChangeUITextAction0"/>
20    </Trigger>
21    <Actions>
22     <IncrementCounterAction name="IncrVarAction0" intVar="Variable0"/>
23     <SetHologramStateAction name="HoloStateAction0" onHolograms="Sphere1 Sphere2
           Zyl2" offHolograms="Zyl1"/>
24     <SetTriggerStateAction name="TriggerStateAction0" triggerName="Trigger1"
          canTrigger='true'/>
25     <ChangeUITextAction name="ChangeUITextAction0" text="Tutorial step nr. 4"/>
26     <MoveRobotAction name="MoveRobotAction0" target="100 100 100 45 15 0/>
27     <SetRobotHandStateAction name="SetRobotHandAction0" state="open" />
28    </Actions>
29   </Events>
30  </PProgram>
```

It does not matter in which sequence all the elements occur. Only the hierarchically structure has to be kept exactly as the example document above. The PARRHI library will at some point deserialise the Parametrised Program using a standard .NET Framework XML library. A validation should be performed to find flaws beforehand.

4.6.2 XSD Generation and Validation

The hierarchy of XML documents can be specified very detailed in XML-scheme documents, which are also called XSD files [34]. These documents define which element can or must occur at witch location in the XML document, which attributes these elements can or must have

and so on. The XSD file can even specify minimum and maximum values, data types and much more.

Hence, an XSD document is an essential part of the importing / validation process. Since writing these documents can be work-intensive, Microsoft has developed ways to generate XML schemes from a set of XML documents. Especially, when the underlying XML document may change repeatedly, it can be very exhausting to update the XSD file every time. For that I wrote a small *C#* application, that takes multiple .xml documents as an input, and outputs one XSD document that defines rules, which fit all input XML files.

Another application developed by Microsoft called ”xsd.exe” [40] allows to convert XSD documents into *C#* class definitions. During the development of this bachelor's thesis, I automated the process of generating the XSD file and from that the *C#* class definition, using the programming language Perl [41].

This means, that whenever an attribute had to be added to some XML element in the Parametrised Program, this attribute had to be added in one of the example input XML documents and the execution of the Perl script would then output new XSD documents and a *C#* class to deserialise the Parametrised Program.

One huge advantage of XSD documents and their validator is, that the latter can output exact error messages with what went wrong in the input XML document. This includes the type of error, line- and column numbers and suggestions on how to fix it. For the PARRHI system this means, that very exact feedback can be printed to the user if any errors occur. This makes it easier for non-software engineers to accomplish their tasks.

5 Evaluation

After the PARRHI system's development, it is now time to ask, whether or not the end goal was achieved and the requirements are met. Therefore, two evaluations were performed. First, traditional AR development methods and the PARRHI system were compared, and then other people tried to succeed in a specific task that was given to them.

5.1 Evaluation 1: Comparing PARRHI with traditional methods

First, a Use-Case was defined. The use case required an AR application to be developed. This AR application was then developed in two different ways. Once using Unity and traditional *C#* development, and afterwards using the PARRHI system. For future reference, the Unity-approach will be called "*manual attempt*" and the PARRHI approach will be called the "*PARRHI attempt*". After both developments are finished, the Pros and Cons of each approach were summarised and compared.

It is important to mention, that I know the PARRHI syntax very well and do not need any time to get used to it. But, the same is true for the manual attempt with Unity. This evaluation can more be seen as a test rather than an actual evaluation due to its statistical insignificance.

5.1.1 Evaluation 1: Use Case Definition

The following use case was the base this evaluation. As a context, the same factory as explained in section 3.1.2 was used.

The task, which should be supported by an Augmented Reality Robot-Human Interface is the following. First, the employee using the AR application approaches the robot work cell

and should then be commanded to walk over to a specific point, that is in a safe distance from the robot. After that, the person has to move the robot's TCP close to their position, so that they can touch the robot's tip. The user will be asked to remove the item in the robot's gripper and jog the robot back into its starting position. Having reached this position, the user will be told to move away into a marked area. The robot will then drive into his "zero" position, where all joint angles are 0 degrees.

This task involves multiple components in the factory, including the robot itself, the user, jogging the robot and walking around. To keep the scope of this evaluation at a reasonable level, the Robot-Library was reused for the *Manual attempt*.

5.1.2 Evaluation Manual Attempt

The manual attempt is defined by programming the task described above manually. The same programming language and game engine as during the development of the PARRHI system was used, which means that the complete project setup including libraries and tools could be copied. I reused the Robot forward kinematics model and the robot library that handles the communication between any .NET Framework application and the robot controller.

First, the image tracking system had to be set up. There are two image markers. One for the robot and thus the AR world itself, and one for the user's instructions UI. Although I have already done AR setups in Unity, this process took quite some work again, since there is always something, that would not compile or work. After this setup was done, a lot of work went into setting up a framework, which allowed me to implement the actual workflow task. I periodically cycled through a method, which kept track of the current state the application was in and also checked state transition conditions in every cycle. Since this application's workflow contained many states, where either the user or the robot had to move somewhere, I had to implement a three-dimensional `IdAboutEqual()` method, which took two vectors and a tolerance value and checked whether or not the two points were close enough. Similarly to that, there were many specific smaller tasks that had to be solved to succeed with the task.

Since the solution to this task involved five points that were targeted in some way, there were many visibility changes for the holograms marking these locations. The small framework I have built for this Use-Case worked with the objects' names. Unity has something referred to as "GameObjects" which can be geometric 3D objects. For this task, I defined, that all the 3D objects, which marked locations for a certain step, have a name beginning with *"State_i"*, where i represents the step number. The framework would then always keep 3D objects, which had the current state-number in their name visible and hide the others.

In summary, the manual attempt took about 450 lines of source code to be written, and some configurations in the Unity engine to be made.

The developer, unfortunately, must be an experienced programmer, know Unity and *C#* development well, and also know how to handle image tracking and gesture recognition libraries to develop AR applications using Unity. Although the most complicated tasks like image tracking are done by some third party libraries, the developer has to know how to use them effectively. Also the Unity Engine is not perfectly intuitive.

The upside is, that any non-trivial task can be achieved. If one step in the application involves complex logic, the developer can simply add the feature, because they are developing everything in source code anyway. The downside of this is that also the most trivial processes have to be implemented every time. Furthermore, developing the application in Unity itself means, that the product has to be compiled, built and then transferred to the HoloLens every single time something changes. This process takes about 3-5 minutes and is very error prone.

5.1.3 Evaluation PARRHI Attempt

The PARRHI attempt means, that the same use case as above was be implemented using the PARRHI system. This attempt was based on the default template of the Parametrised Program, which contains the basic XML structure and defines the needed objects to animate the robot.

The following listing shows the mentioned XML template, which was used for this attempt. As can be seen, the template already predefines all of the robot's joint positions and draws the robot with spheres and cylinders.

```
1  <PProgram>
2   <Points>
3    <!--Robot points to show the actual robot-->
4    <PointRobot name="Joint1" J1="0" J2="1" Scale="0" />
5    <PointRobot name="Joint2" J1="1" J2="2" Scale="0" />
6    <PointRobot name="Joint3" J1="2" J2="3" Scale="0" />
7    <PointRobot name="Joint4" J1="3" J2="4" Scale="0" />
8    <PointRobot name="Joint5" J1="4" J2="5" Scale="0" />
9    <PointRobot name="RobotTip" J1="5" J2="5" Scale="0" />
10   <PointCamera name="Camera"/>
11  </Points>
12   <Holograms>
13   <!-- Robot Start -->
```

```
14    <Sphere name="Joint1Sphere" point="Joint1" radius="90"/>
15    <Sphere name="Joint2Sphere" point="Joint2" radius="80"/>
16    <Sphere name="Joint3Sphere" point="Joint3" radius="70"/>
17    <Sphere name="Joint4Sphere" point="Joint4" radius="40"/>
18    <Sphere name="Joint5Sphere" point="Joint5" radius="20"/>
19    <Sphere name="Joint6Sphere" point="RobotTip" radius="20"/>
20    <Zylinder name="Axe1" point1="Joint1" point2="Joint2" radius="90"/>
21    <Zylinder name="Axe2" point1="Joint2" point2="Joint3" radius="70"/>
22    <Zylinder name="Axe3" point1="Joint3" point2="Joint4" radius="70"/>
23    <Zylinder name="Axe4" point1="Joint4" point2="Joint5" radius="40"/>
24    <Zylinder name="Axe5" point1="Joint5" point2="RobotTip" radius="20"/>
25   </Holograms>
26   <Events>
27    <Triggers>
28    </Triggers>
29    <Actions>
30    </Actions>
31   </Events>
32  </PProgram>
```

Since the PARRHI system handles the AR specific challenges, I directly started with implementing the workflow. I first defined the four positions together with four corresponding holograms, which would be needed for the task. After that, I chronologically worked through the task and defined each states trigger and its actions. The finished Parametrised Program contains five point definitions, four holograms, eight triggers for the individual steps and 21 actions that are executed by the triggers. In total there are 38 lines of parametrised objects.

5.1.4 Conclusion Evaluation 1

Comparing the two methods described above, I come to the conclusion, that using the PARRHI system definitely lowers the entry barriers for AR development but at the same time might limit the developer's possibilities.

For trivial tasks, the PARRHI system greatly simplifies the development process of these applications. A potential developer most importantly does not have to be a software engineer to do work with the PARRHI system. The developer can invest more time into the application's workflow using their domain-specific knowledge since all AR and many other aspects are handled by the PARRHI system.

But PARRHI's strengths are its biggest weaknesses. As soon as a developer needs a feature that is not supported by the PARRHI system, other solutions have to be found. There is

currently no way of extending the trigger/action system with custom objects. This limitation only applies to specific features that are not supported by actions or triggers. The PARRHI system is indeed capable of creating high complexity workflows. Parallelisation and loops, for example, are not a problem.

A solution to the mentioned weaknesses discovered in this evaluation will be proposed in section 7.2.

5.2 Evaluation 2: New Developers

As briefly described in this chapter's introduction, the second evaluation tested the system's usability by asking participants to develop an AR Application using the PARRHI system.

5.2.1 Evaluation 2: Use Case Definition

The task for the experimentees was to write a Parametrised Program for the PARRHI system, which implements the following AR application.

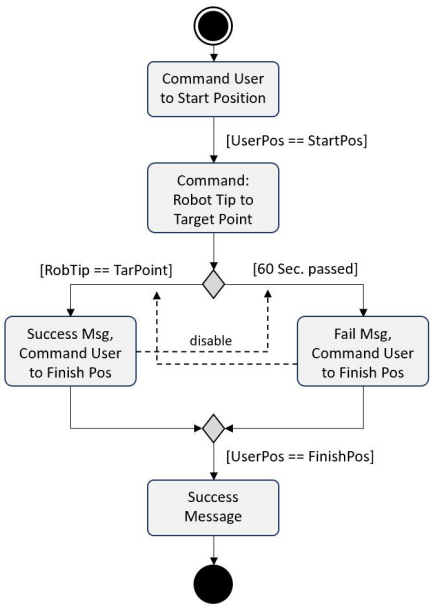

Figure 5.1: Evaluation 2 use case-workflow

1.) The User should approach the robot safely by walking to the starting position.

2.) Then, the robot should be jogged into a given position in under 1 minute.

3.) The application should then display a success or failure message according to step 2.

4.) In both cases, the user should be commanded to move away into the finishing position, such that the robot can not inflict any damage.

The coordinates of the Start, Finish and Robot-Target Points will be given to the experimentees. Fig. 5.1 depicts one solution to the given task as a workflow diagram.

Each experimentee was given an explanation of the PARRHI system and syntax and was provided a cheat sheet, which briefly summarised all commands and object definitions in the PARRHI system. The basis of their work was the same Parametrised Program template, as it was used for the PARRHI attempt in evaluation 1. The participants were asked to develop an AR application according to the Use Case definition above. After they have completed their task, they were asked three questions, the answers of which are summarised below. Their work was timed and evaluated whether or not they successfully achieved the goal. I was available for questions during their attempts, but did not answer any questions regarding the core concept of the PARRHI system.

5.2.2 Summary Evaluation 2

The second evaluation has been completed by six people, five of which are mechanical engineering and one a software engineering bachelor student all aged from 20 to 25. It is important to note that no students knew anything about the PARRHI system and the parametrised program syntax beforehand, and it was their first attempt of developing with it. All students had some minor experiences with different programming languages, and about half of them are very experienced in different programming related technology stacks. None of them had any experience in Augmented Reality development.

Five out of the six students reached their goal, with the average time being 42 minutes, excluding the student who did not succeed, who took about 55 minutes while omitting the time limit during step 2. While two students did not ask any questions and worked completely unassisted, some others did not comprehend the new system as fast and needed some guidance along the way. I tried not to answer questions regarding the core concept of the Parametrised Program. Most questions answered regarded the use case's workflow.

All but one student took a chronological approach to the task. The only software engineer approaches the challenge completely different. He first counted the number of elements in each category he might need and wrote them down. In the end, he connected all elements by filling out the corresponding parameters. Most realised rather quickly, that they had to define the triggers deactivated and activate them when appropriate. In some but not all cases, the students lost the overview towards the end, since the given names of the parametrised program objects were not optimal. This resulted in confusion regarding the names of objects and occasionally misplacing some parameters.

Another common mistake was, that XML elements were placed within the wrong tags, which, of course, resulted in the PARRI system not accepting the Parametrised Program. The

corresponding feedback message from the PARRHI system quickly helped them to solve the issue and reorganise their Parametrised Program.

After the students debugged their Parametrised Program within the simulated environment, I uploaded their project onto the used Microsoft HoloLens and let them test their application with the real robot.

All students were asked the same questions after they finished their assignment. The following listing displays the question and summarises the students' answers.

1.) *"How much experience do you have with AR development and programming in general?"*
All students had some basic experience with C programming. Some were very experienced in some other programming languages, but none of them ever developed an Augmented Reality application nor operated a robot. Only one of them used a head-mounted Augmented Reality device so far.

2.) *"What do you think is the biggest advantage of the PARRHI system?"*
All experimentees said, that they would never have been able to fulfil the given task without the PARRHI system, since they either do not know *C#*, Unity or both. The software engineer, who had experience in Unity Game development answered, that it would have taken him much more time to achieve a similar result, even with his knowledge about Unity. Furthermore, they all liked the fact, that they saw immediate success with their first attempt of developing the AR application.

3.) *What do you think is the biggest disadvantage of the PARRHI system?*
Generally all students reported, that they either lost the overview (or were very close to do so) towards the end. The reason for that was the strict separation between triggers and their actions in the parametrised program.
Since their use case was crafted in a way, that it was possible to succeed with the PARRHI system, none of them saw a weakness in the PARRHI system's capabilities. During the discussion, they all agreed, that it is currently impossible to solve very specific problems, which require a different kind of trigger or action, since the developer cannot define any custom object types.
A solution to both mentioned problems will be proposed in section 7.2.

5.3 Solution to Problems found in Evaluation

The evaluation found two major flaws in the current concept of the PARRHI system. Firstly, the expermimentees lost the overview of their own parametrised program. Secondly, the possible use cases are limited by PARRHI's predefined capabilities.

5.3.1 Solution to Problem 1: Overview and Program Complexity

This problem might be solvable in two distinct but not exclusive ways.

The discussion with the experimentees from evaluation 2 brought up, that a simple re-organisation of the parametrised program's structure could help to increase the overview. Namely, the separation of triggers and their corresponding actions in combination with the name referencing confused the developers, when their program grew in length.

By allowing actions and triggers to be defined underneath the same XML-parent element, the logical workflow could be represented in the program much better. Furthermore, adding the possibility to define "inline actions" would help to reduce the number of names (and with that the confusion) within a program.

Most of the time, one action was exclusively triggered by one trigger. This one to one relationship could be referenced implicitly by defining the action as an XML-child of the trigger. As shown in the example below. A trigger would then invoke all implicitly referenced actions in addition to the ones defined with the `actions` parameter.

These implicitly referenced actions do not need a name parameter since their execution is defined by their parent trigger.

```
1 <!-- Define Trigger -->
2 <DistanceTrigger name="DTrigger" point1="..." point2="..." distance="15"
      actions="...">
3  <!-- Define Action -->
4  <ChangeUITextAction text="You have reached your target successfully!"/>
5 </DistanceTrigger>
```

Another possible solution to reduce the confusion and complexity within the parametrised program would be a graphical editor, which generates the XML document. The syntax defined in the parametrised program is an almost perfect fit for UML activity diagrams (for reference, see Fig. 5.1.).

5.3.2 Solution to Problem 2: PARRHI's Limitations

With the PARRHI system the developer can only perform actions, which are directly supported by the different types of actions. Whenever an action, which is not available in the PARRHI system, should be performed, the system reaches its limits. The same is true if the application should display holograms in different shapes than spheres and cylinders.

This problem could be overcome by allowing the developer to define custom implementations of certain components within the parametrised program. In the simplest case, this implementation could be provided in a specific *C#* source code file. The developer would get access to all parameters provided by the PARRHI system, and would also be able to reference objects defined within the parametrised program.

This way, the developer could setup and solve the simple parts within the parametrised program with the supplied object types, and implement everything else on their own. For example, if the developer needs a very specific trigger to be defined, they could achieve that, by directly implementing them in source code. The developer would still not care about any AR functionalities and can still focus on his domain knowledge.

The defined custom components could then be used in the parametrised program as any other type of trigger, provided they are implemented with the correct properties.

The real-life usage could be, that developers continuously extend their custom implementations, and use them like normal elements in the parametrised program. In larger companies, one software engineer could take over the task of implementing these specific functionalities, and less skilled or specialised developers could use these custom elements in their parametrised program.

6 Technology Transfer

This chapter will describe what measurements had to be undertaken, in order to extend the PARRHI system as it is, for a wider variety of use cases and functionalities. The structure of this section will be grouped by technologies or components within the PARRHI implementation.

6.1 Replacing the Target Machine

In the provided implementation a Fanuc CR-7iA/L robot was chosen as a demonstrator for the PARRHI concept. Of course, there is a big variety of desirable targetable machines, like other robot types from Fanuc or other suppliers, other machines like industry 4.0 machines, etc.

The PARRHI library is written in a very modular manner. This means, that extending or replacing certain parts of it is very easy. To replace the target machine, one would only have to extend the Input/Output modules and the Real World Model, to be able to communicate with the desired machine.

At this point, I have to separate three different cases. Case 1 would be to communicate with another robot in the Fanuc product family. Case 2 is to communicate with an industrial robot of another brand and finally case 3 would be to include machines which are not industrial robots.

Case 1 is very easily achievable since we built a working communication system for Fanuc controllers. In this case, only the Real World Model has to be updated, since the forward kinematics is based on this specific robot. The modification to involve other Fanuc robots would thereby only mean, to adapt the forward kinematic model's vectors and matrices as defined in section 4.3.3. Currently, this model is defined in source code, since in my specific implementation example I only had one robot to work with. So a generalisation would have

been a slight overkill. Theoretically, one could simply expose the forward kinematics model via an Interface, and let knowledgable developers implement their own robot.

Case 2 is a bit more tricky since the communication problem has to be solved again. One could reuse the Robot Library (see section 4.5), and only replace the other endpoint of the network socket with the new robot. The controlling and parsing of the commands on the robot side would have to be rewritten on the specific system. Our current KAREL / TP solution only works on Fanuc Robot-Controllers and not on any other type of industrial robot. Then again the forward kinematic's model would need an adaption to the new robot too.

Case 1 and Case 2 do not require any changes to the PARRHI system itself. The Parametrised Program could be used as it is. In case 3 though, this would likely change, since conceptually another machine is not the same as an industrial robot. The Distance Trigger, for example, would not make any sense if the target machine does not have any moving parts. This means, that in addition to the communication module (Input / Output Modules), one would probably have to define new triggers and actions for the target machine, what requires changes to the PARRHI library itself. Since the PARRHI library is implemented with element base classes and inheritance structures, the implementation of custom triggers and actions is rather easy.

6.2 Replacing the AR / 3D Engine

The current implementation uses the Unity Game Engine as a host. Since all the application logic (PARRHI Library) and also the Robot Library is implemented in engine independent .NET libraries, one could easily switch to any other 3D engine, which is capable of building for Augmented Reality Devices and is able to include .NET dynamically linked libraries (DLLs).

The new implementation would have to implement three main aspects: Firstly, the engine has to host the PARRHI and Robot libraries, which is easy, as long as the engine's scripting environment supports .NET libraries. Secondly, the AR-Output Module has to be rewritten for the new engine. In my implementation, this only took about 30 lines of code. The AR-Output module only instantiates new Holograms in the engine's environment and displays them accordingly. Lastly, the image tracking problem would have had to be solved again. The used image tracking library (Voforia engine, see section 4.4.1) supports the Unity Engine, iOS, Android and UWP platforms. In principle, any library could be used.

For example, the Unreal Engine 4 [42] allows *C#* DLLs to be imported, and can thus work easily with the PARRHI libraries.

6.3 Replacing AR Device

Transitioning to another AR device is probably the easiest of all technology transitions. Unity itself supports building for multiple devices including mobile phones, desktop apps, HoloLens, consoles and more [43]. Basically, the new device only has to possess an accessible camera, which can be used for AR applications, and there has to be a 3D engine, which supports the device.

7 Conclusion

This final chapter concludes this bachelor's thesis. After summarising the project and its most important components, some future work is identified.

7.1 Summary

Developing AR applications requires a certain set of skills including the knowledge of different frameworks, programming languages and engines. In industry, professionals of their field often do not have any or enough experience in AR development, which either results in no applications being developed, or Software Engineers being hired, which increases communication efforts, costs and possibly the time-to-market.

The presented framework called "PARRHI" (Parametrised Augmented Reality Robot-Human Interface) tries to solve the problem of interdisciplinarity by enabling people who do not possess deep skills in AR development or programming in general to quickly and easily develop and execute AR applications for their specific domains.

The approach is to create an abstract layer above the technology layers, able to handle the AR and real-world communication components. The developer, who is a person with the specific domain knowledge wanting to create AR applications, can now focus on the application's workflow and let the PARRHI system manage all other aspects like image tracking, AR, wireless communication with the real world and more.

To further support the developer, the PARRHI system follows a concept called "parametric thinking". This means that the program written by the developer is parametrised using placeholders. These placeholders can then be assigned data which the PARRHI system discloses to the program, or also other objects defined in the parametrised program.

In conclusion, the developer's required skill set is limited to their specific domain knowledge. The PARRHI system handles all other aspects of AR Human Robot Interfaces on its own.

7.2 Future Work

Although the proposed PARRHI system seems to be an interesting and promising approach, it is by all means far from being a highly advanced implementation. As its evaluation showed, there still are some problems in terms of user-friendliness that need to be addressed. An industrial implementation of PARRHI therefore needs further development before successful deployment.

Before undertaking further development, a thorough evaluation with a higher quality standard and many more participants that fit into the role of the developer has to be performed. In addition, the requirements should be revised and documented formally in cooperation with industry partners to verify their applicability. Using the new knowledge as a starting point, certain abstractions would have had to be made to the core concept of the PARRHI system.

8 Bibliography

[1] G. Klein, P. J. Feltovich, J. M. Bradshaw, and D. D. Woods, "Common ground and coordination in joint activity", *Organizational simulation*, vol. 53, pp. 139–184, 2005.

[2] C. Laschi, B. Mazzolai, and M. Cianchetti, "Soft robotics: Technologies and systems pushing the boundaries of robot abilities", *Sci. Robot.*, vol. 1, no. 1, eaah3690, 2016.

[3] M. Billinghurst, "Augmented reality in education", *New horizons for learning*, vol. 12, no. 5, pp. 1–5, 2002.

[4] U. D. o. L. Bureau of Labor Statistics. (2018). Occupational outlook handbook, software developers, [Online]. Available: `https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm` (visited on 03/28/2019).

[5] H. J. Baker. (2017). 2018's software engineering talent shortage — it's quality, not just quantity, [Online]. Available: `https://hackernoon.com/2018s-software-engineering-talent-shortage-its-quality-not-just-quantity-6bdfa366b899` (visited on 04/25/2019).

[6] R. T. Azuma, "A survey of augmented reality", *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 355–385, 1997.

[7] A. Minds. (2019). Endgeräte für ar mr, [Online]. Available: `https://www.augmented-minds.com/de/erweiterte-realitaet/augmented-reality-hardware-endgeraete/` (visited on 04/25/2019).

[8] C. Perey, T. Engelke, and C. Reed, "Current status of standards for augmented reality", in *Recent Trends of Mobile Collaborative Augmented Reality Systems*, Springer, 2011, pp. 21–38.

[9] P. Figueroa, R. Dachselt, and I. Lindt, "A conceptual model and specification language for mixed reality interface components", in *VR 2006, In Proc. of the Workshop "Specification of Mixed Reality User Interfaces: Approaches, Languages, Standardization*, 2006, pp. 4–11.

[10] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics", in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 5382–5387.

[11] I. Y.-H. Chen, B. MacDonald, and B. Wunsche, "Mixed reality simulation for mobile robots", in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 232–237.

[12] R. S. Magazine. (2018). The grand challenges of science robotics, [Online]. Available: `https://robotics.sciencemag.org/content/3/14/eaar7650` (visited on 04/25/2019).

[13] P. Milgram, S. Zhai, D. Drascic, and J. Grodski, "Applications of augmented reality for human-robot communication", in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, IEEE, vol. 3, 1993, pp. 1467–1472.

[14] M. Walker, H. Hedayati, J. Lee, and D. Szafir, "Communicating robot motion intent with augmented reality", in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, ACM, 2018, pp. 316–324.

[15] V. Grozdanov, D. Pukneva, and M Hristov, "Development of parameterized cell of spiral inductor using skill language", Apr. 2019.

[16] M. Stavric and O. Marina, "Parametric modeling for advanced architecture", *International journal of applied mathematics and informatics*, vol. 5, no. 1, pp. 9–16, 2011.

[17] H. Seichter and M. A. Schnabel, "Digital and tangible sensation: An augmented reality urban design studio", 2005.

[18] F. D. Salim, H. Mulder, and J. Burry, "A system for form fostering: Parametric modeling of responsive forms in mixed reality", 2010.

[19] X. Wang, P. E. Love, M. J. Kim, C.-S. Park, C.-P. Sing, and L. Hou, "A conceptual framework for integrating building information modeling with augmented reality", *Automation in Construction*, vol. 34, pp. 37–44, 2013.

[20] B. A. R. Associtation. (2018). Robot programming methods, [Online]. Available: `http://www.bara.org.uk/robots/robot-programming-methods.html` (visited on 04/07/2019).

[21] P. Diegmann, M. Schmidt-Kraepelin, S. Eynden, and D. Basten, "Benefits of augmented reality in educational environments-a systematic literature review", *Benefits*, vol. 3, no. 6, pp. 1542–1556, 2015.

[22] P. Salamin, D. Thalmann, and F. Vexo, "The benefits of third-person perspective in virtual and augmented reality?", in *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, 2006, pp. 27–30.

[23] S. Henderson and S. Feiner, "Exploring the benefits of augmented reality documentation for maintenance and repair", *IEEE transactions on visualization and computer graphics*, vol. 17, no. 10, pp. 1355–1368, 2011.

[24] T. J. McCabe, "A complexity measure", *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[25] M. Corporation. (2019). Microsoft hololens, [Online]. Available: https://www.microsoft.com/en-us/hololens (visited on 05/15/2019).

[26] F. Ltd. (2019). Fanuc, [Online]. Available: https://www.fanuc.eu/uk/en (visited on 05/15/2019).

[27] ——, (2019). Fanuc cr-7ia/l, [Online]. Available: https://www.fanuc.eu/de/en/robots/robot-filter-page/collaborative-robots/collaborative-cr7ial (visited on 05/15/2019).

[28] ——, (2019). Fanuc r-30ib, [Online]. Available: https://www.fanuc.eu/de/en/robots/accessories/robot-controller-and-connectivity (visited on 05/15/2019).

[29] Schunk. (2019). Schung gripper co-act egp-c, [Online]. Available: https://schunk.com/shop/de/de/Greifsysteme/SCHUNK-Greifer/Parallelgreifer/Co-act-EGP-C/c/PGR_3995 (visited on 05/15/2019).

[30] M. Corporation. (2019). Microsoft .net framework, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/ (visited on 05/15/2019).

[31] ——, (2019). Microsoft c-sharp, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/ (visited on 05/15/2019).

[32] ——, (2019). Microsoft .net class library, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/tutorials/library-with-visual-studio (visited on 05/15/2019).

[33] W3C. (2008). Xml specification, [Online]. Available: https://www.w3.org/TR/xml/ (visited on 05/15/2019).

[34] ——, (2009). Xsd specification, [Online]. Available: https://www.immagic.com/eLibrary/ARCHIVES/TECH/W3C/W090430S.pdf (visited on 05/15/2019).

[35] R. M. Murray, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[36] U. 3D. (2019). Unity game engine, [Online]. Available: https://unity3d.com/unity (visited on 05/18/2019).

[37] Vuforia. (2019). Vuforia ar engine, [Online]. Available: https://developer.vuforia.com/ (visited on 05/19/2019).

[38] M. Corporation. (2019). Microsoft mixed reality toolkit, [Online]. Available: https://github.com/microsoft/MixedRealityToolkit-Unity (visited on 05/19/2019).

[39] Fanuc. (2019). Fanuc karel programming language, [Online]. Available: https://www.fanucamerica.com/support-services/robotics-training/CourseDetails.aspx?CourseNumber=KAREL-OP (visited on 05/19/2019).

[40] M. Corporation. (2019). Xml scheme document tool, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-schema-definition-tool-xsd-exe (visited on 05/20/2019).

[41] P. organisation. (2019). Perl programming language, [Online]. Available: https://www.perl.org/ (visited on 05/20/2019).

[42] Epic. (2019). Unreal engine 4, [Online]. Available: https://www.unrealengine.com/en-US/ (visited on 06/27/2019).

[43] U. Technologies. (2019). Unity multiplatform support, [Online]. Available: https://unity3d.com/de/unity/features/multiplatform (visited on 06/27/2019).

Declaration

The submitted thesis was supervised by Prof. Dr.-Ing. Birgit Vogel-Heuser.

Affirmation
Hereby, I affirm that I am the sole author of this thesis. To the best of my knowledge, I affirm that this thesis does not infringe upon anyone's copyright nor violate any proprietary rights. I affirm that any ideas, techniques, quotations, or any other material, are in accordance with standard referencing practices.

Moreover, I affirm that, so far, the thesis is not forwarded to a third party nor is it published. I obeyed all study regulations of the Technical University of Munich.

Remarks about the internet
Throughout the work, the internet was used for research and verification. Many of the keywords provided herein, references and other information can be verified on the internet. However, no sources are given, because all statements made in this work are fully covered by the cited literature sources.

Munich, July 12, 2019

Eric Vollenweider