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:	August 26, 2019

Table of Contents

1 Introduction

1.1 Reasons for- and problems with Human-Robot collaboration

Today's world is highly automated in each and every aspect of our lives. One of the most important advancements within the last 30 years where robots. They allow us to minimize costs, while maximizing output, quality and efficiency. With the rising need of agile production plants, classic robotic systems do not seem to fulfil all requirements anymore. Industrial robots do have numerous abilities but adapting to new situations easily is certainly not one of them. In contrast, the human's strength always was exactly that - to learn quickly. Thus, Human-Robot (HR) collaboration is an inevitable step towards the future of humanity.

Robotics needed to adapt to the new situation where humans and robots share a perimeter, thus compliant robots where developed. For machines to feel natural it is not enough to be compliant and not hurt humans. As Gary Klein et al. stated, communicating intent is a key issue in effective collaboration within teams [1].

Augmented Reality (AR) can help us to communicate with robots in three dimensions. 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. Both robotics and AR have been heavily worked on for the last 20 years separately, but their combination has mostly remained untouched. Lately though, it has become a new focus area in research. While it is known that AR can improve communication in certain use cases [3], actually developing them is not as easy, since there is a jungle of technologies and frameworks. Since most development environments require rely on source code to be written, the reusability is quite low, which may result in re-doing the same steps over and over again, requiring software engineers along the way to do everything. Thus, the time-to-market is long and expenses are high.

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 over 30% within the next 10 years. One possible reason could be, that acquiring the necessary skills to develop software applications takes a long time and is not viewed as an easy path. Specifically AR applications currently require a wide variety of skills. On top of that, many processes and steps have to be repeated for each new AR-application and it is increasingly hard to build upon other peoples work.

The lack of programming talent is also seen in the robotics industry, which requires its developers to posses an even wider set of skills, including software development, mechanical and the basics of electrical engineering. Due to the lack of developers the speed of innovation and time-to-market may either be slowed down or the companies will have to pay about 20% above market value for their employees [5].

1.2 Parametrised development as an approach to solve AR-developer shortage

Today, many industrial robots are only parametrised and not actually programmed, which results in lower requirements for employees and thus in less time and money spent on training. Employees only set certain locations, points and actions utilizing an easy to understand User-Interface. Applying a similar methodology to the development of Augmented Reality applications could lead to similar effects. If non-programmers could setup, configure and thus develop Human-Robot interfaces involving AR and robot controlling all by themselves, without the need to write a single line of code, the cost efficiency and speed of innovation can be improved.

This bachelor thesis aims to research the feasibility and practicability of such an abstract development environment. First there will be a ”State of the Art” section, where current research and progress is displayed. From there some gaps will be identified, a concept proposed, implemented and tested in a real application.

2 State of the Art

The State of the Art chapter will highlight technologies and products that highly influenced this thesis. First, Augmented Reality itself will be explained and some basics will be laid out, which is followed by a paragraph about current AR-Standardisations, which is important since they help developers to speed up their processes because many decisions do not have to be made repeatedly. Later the combined field of AR and Robotics will be described in detail. How it can use Augmented Reality technology, which projects succeeded in doing so and what benefits it brought. A part about parametrised thinking and other disciplines that already work as such represents the last component of this thesis. A final remark describes how this thesis tries to combine all explained technologies or principles into an innovative way of programming AR applications.

2.1 Augmented Reality and its standardisations

This bachelor's thesis evolves around AR and its use cases. It is utterly important to clearly define some terms before using them thoroughly. AR technology must not be confused with Virtual Reality (VR). Whereas VR completely immerses users, the former still allows the real world to be seen together with superimposed 3D objects projected into the user's view. T. Azuma crafted 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. Namely, the system has to combine real and virtual objects, be interactive in real time and 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".

After having cited a definition about AR and answering what it is, this paragraph describes what underlying principles exist. Two types of AR technologies are in use today. Firstly, there are see through displays with strong user immersion and secondly monitor based approaches like smartphones with cameras. A large variety of producers [7] release new AR-Devices regularly. In this bachelor's thesis the term AR should be understood as head-mounted

superimposing Augmented Reality devices with see through displays, such as the Microsoft HoloLens [8], that actually create the illusion of three dimensional holograms being projected into the real world.

Standards often contribute to the development and usage of technologies substantially, firstly by unifying used tech-stacks, thus potentially reducing complexity and secondly by helping new members to navigate in a complex environment. There are not many international standards on AR since the field's popularity still is very young. Nevertheless, C. Perey et al. [9] examined existing standards and some best-practice methods for AR. They clearly identified some gaps in the AR value-chain. The researchers lay out numerous standards for low level implementations such as geo-graphic location tracking, image tracking, network data transmission etc.

Figueroa et. al [10] researched about a "Conceptual Model and Specification Language for Mixed Reality Interface Components". They want 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 smaller behaviour actions of UI control elements. Although their work goes into the right direction, it lacks certain features that robot-human AR interfaces need, and contains unwanted components like pseudo-code. "3DIC" is a specification language that cannot be fully processed automatically, since pseudo-code is not executable.

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

2.2 AR in Robotic applications

2.2.1 AR during development and testing

Wolfgang Hönig et al. examined three different use cases of Augmented Reality [11]. The research team primarily focused on the benefits offered by the co-existence of virtual and real objects during the development and testing phase. 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, thus, saving money and space [11]. 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, less expensively or with improved safety for all participants involved (humans and machines). Similarly, Wünsche et al. have created a software framework for simulating certain parts of robotic systems [12]. These researchers from Auckland worked on methods to combine real world and simulated sensor data and navigate a real-world robot in the combined environment.

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 [13]. In all previously cited cases, Augmented Reality was not used to enhance the interaction between humans and robots but to mitigate the current challenges in developing robotic systems. Very early work by Milgram et al. [14] shows that even the most basic implementations of AR technology, with the objective to improve the information exchange, enhance the bidirectional communication 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 close manner, 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 to communicate the before-mentioned information in natural ways.

In order to solve this problem, Walker et al. [15] explored numerous methods to utilize Augmented Reality to improve both efficiency and 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" [15]. The conclusion was that spatial Augmented Reality holograms are received much more intuitively than simple 2D projected interfaces.

2.3 Parametrised Development

Numerous disciplines utilize parametrised development environments heavily. For example the CAD software *CADENCE* (integrated electrical circuits) offers parametrised cells to optimize the development process [16]. 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 set. The software then calculates its properties and behaviour at runtime. Generally there is no coding skill required.

In architecture Parametric Design is taking overhand 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 constrains allows architects to generate building structures easier and more efficient [17]. Adapting to a changing environment during the design process is much easier, since a substantial portion of work can be completed by algorithms and software applications. Additionally the reusability of components increases massively due to the very formal way of representation. Architecture studios also begin to include Virtual and Augmented Reality technologies actively in their design processes to further incorporate and better understand these new design styles called *Parametric Design* [18]–[20].

2.4 Current methods programming industrial robots

To understand where and how the content of this bachelor's thesis fits into the world of robotics, this chapter will describe the current workflows with robots, how they are configured and programmed. It is important to know that almost all industrial robots are programmed in three distinct ways.

1.) Teaching 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 [21]. These devices basically are touch-tablets fitted for industrial use with emergency stop buttons, more durable materials and so forth. They allow the operator to jog (term for steering the robot manually using the TP) the robot into certain configurations, save the positions and offer a number of other possibilities. For easier 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 needs a substantial amount of training and can generally not be done by untrained people.

Teach Pendants are great for trivial and simple tasks, that do not require lots 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. Important to note is, that parametrised programming already is an industry standard in industrial robotics for some brands of robots.

The other two methods are used for more complex tasks and production lines and each have their Pros and Cons. While Offline Programming is very precise and powerful, it does require a substantial amount of schooling time for the operator. Teaching by Demonstration is the exact opposite. Most people could succeed quickly, but the complexity of achievable tasks and the precision is limited.

2.5 Parametrised AR in Robotic applications

Many industries adopted Augmented Reality in their daily process already. It's advantages and challenges are clearly known and being worked on. There are numerous examples of AR helping in different environments and tasks [22]–[24]. Developing AR applications is generally speaking still a very costly task. On the other hand there is the design principle of "Parametrisation" that may involve increased initial investment but can lower the cost of changes and adaptions to changing environments. Combining the perks of both AR technology and parametrised architecture may bear great potential.

3 Concept

This chapter will explain what requirements were defined for the to be built system and how it will be designed and setup. At first, I will define a goal of this project before writing about the user and system requirements following the V-Model [25]. Then an example Use-Case should help to further understand the goal and purpose of this thesis. For further reading the system will be called "Parametrised Augmented Reality Robot Human Interface" (*PARRHI*).

3.1 Goal, Requirements and Use Cases

As described in section 1.2 this thesis presents a possible approach to solve the previously described problems (see section 1.1). It is the main target of this bachelor's thesis to remove the necessity of high software engineering skills to develop reasonably complex Augmented Reality Robot Human Interface applications, maintaining the quality of the outcome and even increasing the degree of resuability. This might result in lower development costs, lower struggle to gather software engineering talent and even in a shorter time to market.

To succeed in achieving this goal, a specific set of requirements has to be defined and documented in a formal way. To gather these requirements the V-Model developed by the Federal Republic of Germany was used [25].

3.1.1 User Requirements

Before defining the User Requirements the system's end user has to be defined first. The characteristics of the actual end-user of *PARRHI* might be someone who:

- Knows the basics of text editing software,
- has no to little skills in software engineering,

- is a knowledgable employee in bigger factories
- has the goal to develop AR applications for easier collaboration with industrial robots.

It is to mention, that the end user will later be called "Developer", since this person develops AR-applications with the *PARRHI* system. Having an idea of the end-user, the user-requirements can be defined. The user wants to:

- Develop a AR-applications without software engineering skills
- Have the tools necessary to create medium complex applications for use cases such as tutorials, maintenance instructions or other teaching purposes
- Build upon other people's work or projects
- Launch the AR-application on a suitable device
- Possibly use the same development framework on different types and brands of robots

3.1.2 System Requirements

Deriving from the user requirements, one can define what the system should be capable of doing:

1.) allow programming in a simple and intuitive format.

2.) have readable and intuitive feedback on every user input.

3.) have a programming input in a non-binary text format that allows copy and paste reproduction.

4.) support building for hand held mobile devices and head mounted Augmented Reality glasses.

5.) allow bidirectional communication with the real (robot...) and virtual world

6.) allow documentation of the application's workflow

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.

3.1.3 Possible Use-Case: Factory Maintenance

This section depicts a possible use case example for the PARRHI system, emphasising how and why it could be used. To first establish a context, one could imagine a huge auto mobile factory with hundreds of industrial robots assembling certain parts of a car. A factory like this, might want to repeatedly check on their robotic infrastructure to recognise problems and thus avoid any future downtime. The difficulties here are, that one generally does not want a robot to not be in operation, due to the high opportunity costs of it, but on the other hand non-collaborative robots are dangerous and can inflict damage on humans and other materialistic property, which means that the robot should be switched off or at least in a safety mode while humans are around. In conclusion, the operator should be quick and precise in their task without letting down their security safe guard.

Despite being expensive, finding employees that have the necessary skills is not an easy task. Hence it would be interesting to quickly teach new personnel how to succeed in these very specific cases. An Augmented Reality application might be the right approach for cases like these. For every specific maintenance task an employee knowledgable enough could develop AR applications, which lead a group of less-skilled workers through their tasks step by step without ever compromising their safety.

Developing such a high number of AR-applications is not really plausible with current frameworks. Despite the knowledgable employee probably not having enough programming skills, it simply might be too expensive, slow and too hard to adapt to changes. A system like PARRHI would allow the skilled employee to write these applications themself without the need of software engineers. Since many tasks might be similar to one another, the employee could re-use most of the applications program again.

PARRHI applications will be able to involve the Real World's data in their workflow, which allows the application to know where the robot currently is located and command the maintenance worker to move away. Having the possibility to also influence the Real World, the application could hinder the robot from moving, while the employee is in a certain danger area. All that, while showing instructions in AR to easily understand and complete the task.

The newly centrally developed applications could then be distributed to numerous other personnel, who can then fulfil their tasks with less training in a shorter time. In essence, using systems like PARRHI might reduce the development costs of AR-applications substantially, which could open a wide set of use cases, that were untouched before due to the high costs. This might allow for a much better economically usage and a much wider set of real world applications for Augmented Reality apps.

3.2 PARRHI Concept

The following chapter will now explain the concept of the PARRHI system. I will first start with a general overview of the concept, before shortly explaining each component. After that, the information flow in the system will be clarified.

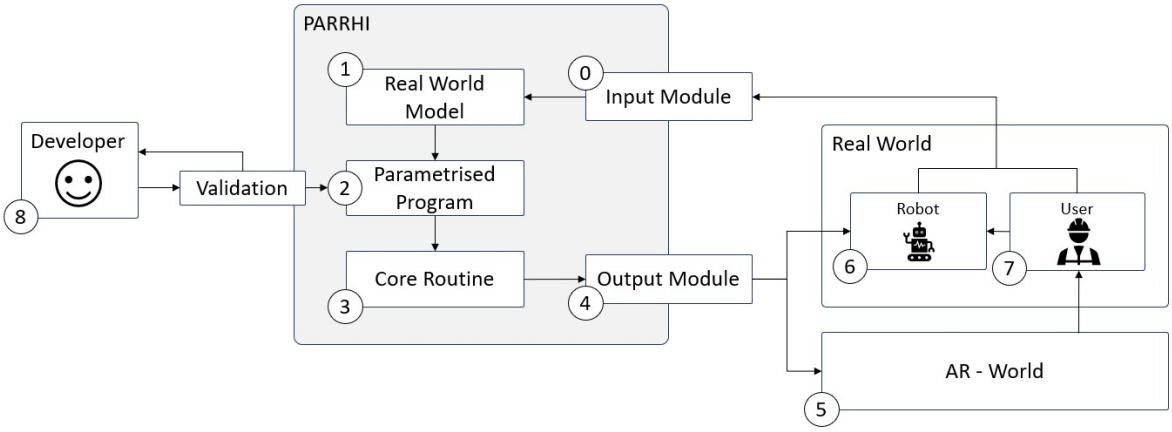

Figure 3.1: PARRHI general concept

Fig. 3.1 depicts PARRHI's concept, which consists of three main parts. The left hand side shows the developer and his workflow. The grey box in the middle is the PARRHI system itself. Lastly, the right side visualises the environment that surrounds the PARRHI system in both virtual and real space. I will now go into detail and systematically explain each component from (0) to (8), before describing the information flow between them.

0.) Input Module: The Input Module (0) is responsible for collecting 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: In order to utilize data from the Input Module (0) it has to be processed. The Real World Model (1) has a deeper understanding about the real world and helps to extract useful information from the data inflow. 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 for 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 by PARRHI itself at runtime. There are two types of parameters. First, there are the ones that are provided by the Real World Model (1) (for example the discussed robot's (6) joint positions) and secondly any object

defined in the Parametrised Program itself (2) can act as a parameter for other objects in the program. These objects can be 3D AR-Hologram definitions, logical instructions or a variety of events and actions. For a more thorough explanation of the Parametrised Program see section **??**.

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 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 create 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 would use the finished application to fulfil their task, which could for example be to do some work following instructions.

8.) Developer: This person develops the application's workflow and behaviour. Their output is the Parametrised Program (2) as a text-based document, which is validated before being fed into the PARRHI system.

It should now be clearer what each component is responsible for. After having laid down the fundamentals, the following paragraphs will track the information flow between the components. The order will be a bit different here, since it will be built up in a chronological way. Hence, I will first start with the Developer (8) crafting the Parametrised Document and then continue with the standard runtime cycle of the PARRHI system.

After having defined what the AR-HR-Interface application should achieve, the Developer (8) crafts the Parametrised Program (2). At this point, the Developer pours his knowledge and expertise into the system, so that other people like the User (7) can profit from it. The text-based hierarchically built document will be validated first, before it is granted access into the PARRHI system. If the validation fails, the Developer (8) receives error messages accordingly an can reiterate over their document until it validates successfully and behaves as the Developer (8) wants it to.

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 onto 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 Parametrised Program (2), which is written using placeholders (parameters), comes into play. The latter 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. At last it evaluates all events, and triggers their actions if needed. These actions might be commands for the AR-World (5) (for example to show or 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 into 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 things. They see the Real World in front of them. There is the Robot (6) and a surrounding environment. The Robot (6) might be 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. 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 could be instructed by holograms to move into a safe-zone. At the same time, the User (7) might control the Robot (6). At this point it is to mention, that the Output Module (4) enjoys priority over the User's (7) input when commanding the Robot (6).

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 environment either by direct commands 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 should give a relative detailed overview about the concept behind PARRHI. The following chapters will now explain each part in an even higher degree of detail, with the goal being, that almost no questions about the concept stay unanswered. The order will be similar to the explanation of the information flow in the PARRHI concept, meaning, that the Input Module (0) and the Real World Model (1) will be explained first, followed by a very thorough explanation of the Parametrised Program (2) with all its objects and definitions. In the end the Core Routine (3) and the Output Module (4) with the Real and AR-World (5) will finish off the concept chapter.

3.2.1 Preparation of Parameters

One main strength of the *PARRHI* concept is its disclosure of its knowledge of the outside world to its inner components via parameters. Two steps are necessary to do that. First, an Input Module (0) has to automatically retrieve real world information. Since the latter is hard to comprehend for computer programs and might not be usable for the Developer's purpose, some model knowledge (1) should be implemented to process the input data in some way. Generally, models are a more abstract version of the object it is supposed to describe. Besides being easier to understand, models can often be explained in a mathematical way, which is of course perfect for computer programs.

The complexity of collecting Real World data strongly depends on the use case. I decided to limit my scope to the collection of three information pieces, since it is not the main focus of this thesis. These three pieces are the user's position, the robot's joint angles and it's gripper state.

Gathering the user's position relative to the robot is the first step. For AR applications, it is utterly 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. To synchronise all coordinate systems, the centre of origin for *PARRHI* applications is always the robot's base. 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 of course 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 via some interface, 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, an old problem in robotics with known mathematical tools and ways to solve it (at least in the case of basic industrial robots). In principle, one can calculate a robot's tool point via every joint angle and some knowledge about 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 outputs 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, which the Developer ((8) in Fig. 3.1) of the *PARRHI* system crafts. It contains parametrised, hierarchically structured data, that defines the behaviour, feel and look of the final AR-HR-application.

The Parametrised Program contains instructions and definitions which from now on will all be called to "objects". All these objects have a certain set of parameters that are needed to fully define their function, visual appearance or behaviour. These parameters can either be previously defined objects, data from the Real World Model or constant values. By using other objects as parameters, one 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.

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

Table 3.1: *Input Data* 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 (fix, 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. With the help of Variables an application could keep track of something by counting events, or also implement state machines that jump between modes.

3.2.2.2 Points

Points essentially are a 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.) Fix-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 according parameters. Thus, the system updates these objects repeatedly with Real World information, that was 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 **Fix-Point** has static coordinates and thus fixed values are used to define their coordinate parameters (see figure 3.2). It could be used to setup holograms that visualize certain spacial environmental constraints that do not move.
Definition Parameters: *name : string, X : float, Y : float, Z : float.*

Robot-Points are defined by two indexes of the robot's joints and one scalar value that will be explained later (see figure 3.3). The application's developer does not have to understand the robot's kinematics and simply uses the joint-indexes 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 feeds this data into the Robot-Points. The final

point's position 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 it 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 the Robot-Points, the *PARRHI* system 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 periodically updates the Camera-Points with the new location 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.*

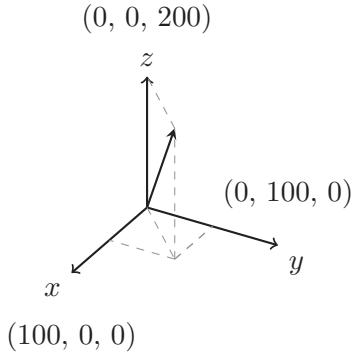

Figure 3.2: Fix-Point example

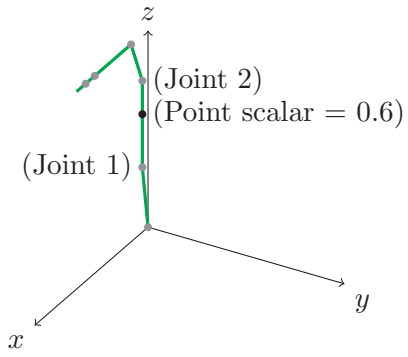

Figure 3.3: Robot-Point example

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 thus updated by the data from the Real World model, holograms are indirectly also 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, respectively

taking one or two points and a radius as input parameters to define its size and position. Other attributes describe its 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 set to the point it was defined with (point1), whereas a cylinder always connects the two points of its definition (point1 and point2). All described types of points in section 3.2.2.2 can be used as input parameters for the location. This is the first example of Parametrised Program 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 utilized 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 have a boolean expression, which is checked each cycle. 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 definitions of them have some attributes in common, which are listed in the table 3.3. Of course, specific trigger types have additional

parameters to define their function. Almost all trigger need other objects as parameters for their definition.

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 reasonably capability numerous different but easy to understand triggers are available for the application's developer. In table 3.4 is a complete list of all defined triggers.

Distance trigger 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 user jogged the robot's TCP into a wished position by using a *Fix-Point* an waiting for them to come close to each other. Second the user's movement can be monitored by utilizing 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*. One could implement a counter for certain events, 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]*

Table 3.4: Trigger types

Name	Input Parameter	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 $t_{enabled}$	$t_{tr} \geq t_{enabled}$

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 on, the timer starts immediately. 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 numerous 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 a quick overview about 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 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** enables setting the visibility of holograms at runtime. If a hologram represents a region for a tutorial step, it can be hidden after the user's task is completed. The new scene can then be setup by displaying new holograms that guide the user's way. Another possible scenario would be, to display a warning boundary, if the user moves into a forbidden zone. This can be achieved by combining Distance trigger and Set-Hologram-State actions.
Definition Parameters: *name:string, onHolograms:[Hologram], offHologram=[Hologram]*

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

When using *PARRHI* the user is presented a GUI that shows text and some other few options. The **Change-UI-Text action** allows to change this displayed text. There are numerous obvious use-cases 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 on, or get disabled by triggering as described above. The Set-Trigger-State action has the ability to (re)activate disabled triggers. 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 will assume that the coordinates are defined in joint-space and will act accordingly.
Definition Parameters: *name:string, target:Vector6, mode:char*

The **Set-Robot-Hand-State action** behaves as one would expect and simply open or close 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 14).

```
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. It's 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 that are 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 their new coordinates, that they have to monitor.

With all new values set, it 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 trigger(3). The executing actions result in a group of commands, that will be sent to the Output Module, which then handles them from now on.

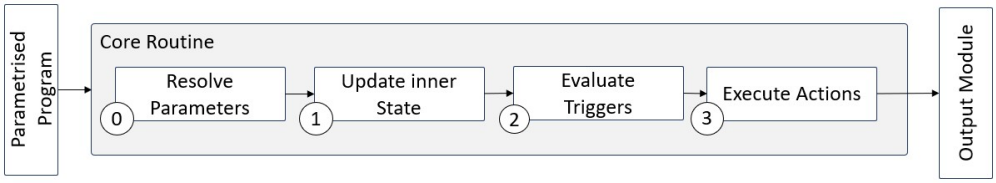

Figure 3.4: Core Routine Workflow

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 shortly 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, that 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, with possibly 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 my 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 a OS neutral protocol has to be defined for this type of communication. In addition to that, operating systems on Robots tend to be sealed towards the outer world making at safe against unwanted access but at the same time hard to interconnect to other systems. This is why, Florian Leitner (a colleague at TUM) and I defined a custom protocol for the bidirectional communication between the PARRHI system and the Real World Robot. To get more information about the implemented protocol and system to communicate between PARRHI and the Robot, refer to section 4.5.

4 Implementation

In this chapter I will write about some implementation aspects that were interesting or problematic during the development of the PARRHI system. For that, I will use a bottom down strategy, meaning, that I will talk about the general setup of the implemented system, and then drill down into the components. I will explain what software was used, why I made these decisions and whether or not they where good or bad in hindsight.

4.1 PARRHI Hardware

For the Augmented Reality purposes, I decided to use the first version of Microsoft's Augmented Reality glass called HoloLens [26]. The HoloLans has all needed 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 [27] was gracious enough to gift the TUM chair "Automation and Information Systems (AIS)" a free CR-7iA/L collaborative, 6 DOF, industrial robot [28] and a R-30iB controller [29]. The R-30iB controller can either be programmed in Fanuc's TP-Language or in 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 gifted us a collaborative Co-act EGP-C [30], that seamlessly integrated into 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, so that this bachelor's thesis could be realised.

4.2 PARRHI Software

Software Implementation wise, the PARRHI system is split up in three big components. First there is the PARRHI library, secondly the Robot library and lastly the Unity engine. The PARRHI library is the most intelligent component, since it contains all the logic for the Real World Model, the Parametrised Program, the Core Routine and commands the Input and Output Modules (see fig. 3.1). The Robot library has the tools necessary to communicate with the Robot in the Real World and is used by the Input and Output Modules. The third component *Unity* hosts the PARRHI runtime library, and handles the AR-World, network communication etc.

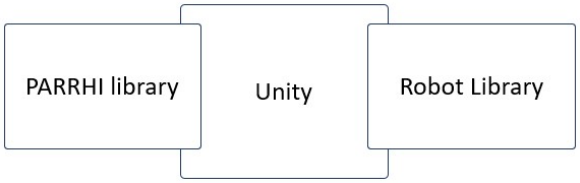

Figure 4.1: PARRHI implementation setup

4.3 PARRHI Library

The PARRHI system was programmed in Microsoft's .NET framework version 4.6.1 [31] using *C#* [32]. Besides being my personal preference, this decision was made because the Unity game engine is scripted in the language *C#*. The PARRHI library itself is a *Class Library* project [33] and has three main tasks.

1.) Import the Parametrised Program and perform some operations on it. The Parametrised Program is a XML [34] based document. After importing, it is validated using an XSD [35] document. Finally the PARRHI library has to extract useable *C#* objects from the deserialized XML document and setup 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 libraries main interface to the hosting environment or the caller 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 it can be seen, it basically holds references to all objects of all types, 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.

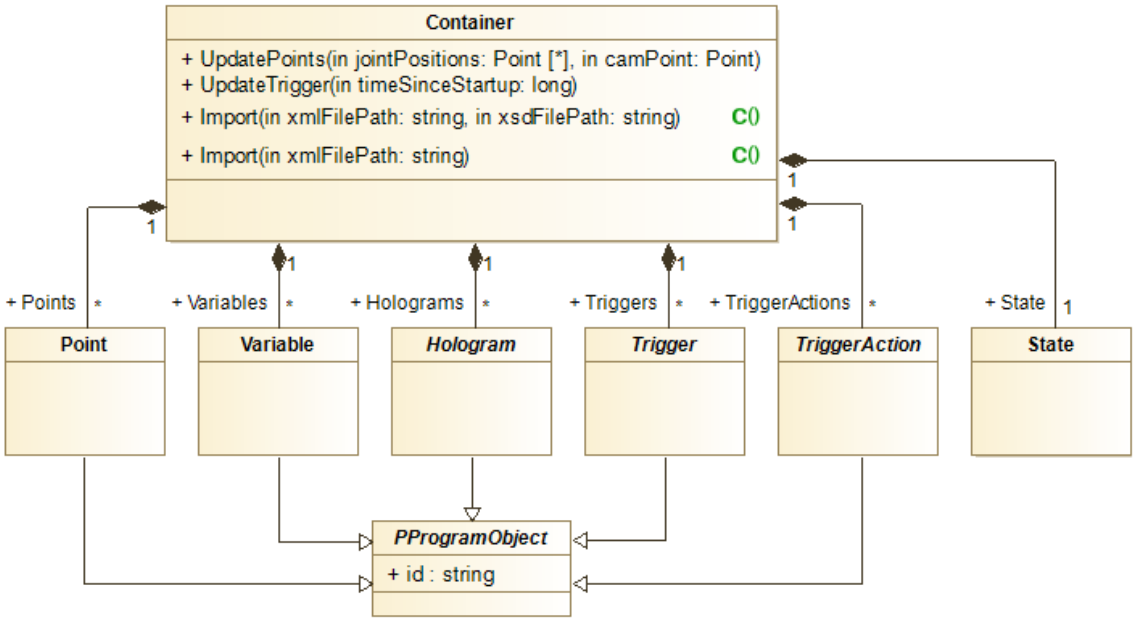

Figure 4.2: Container Class Diagram

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. In the case of a simulation, I can then simply set this delegate 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 exact same reasons as with the `Robot` class - to create independence from the host.

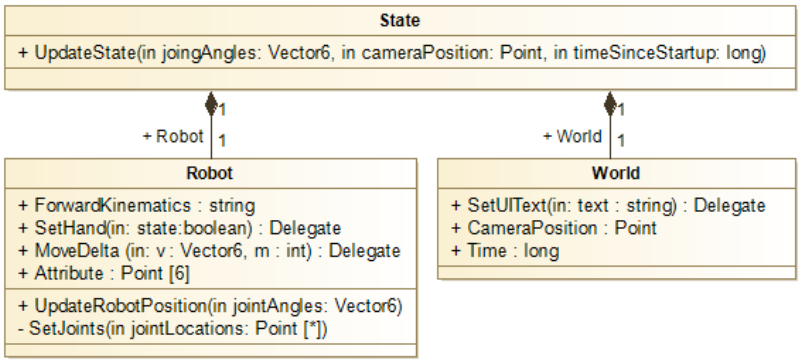

Figure 4.3: UML Class Diagram of State class

These two classes are the main interface to the library's host. They contain multiple sub classes 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, I first created abstract classes of each object type, and then created concrete implementations for each sub-type. The PARRHI interpreter only utilises the public methods that are disclosed in the abstract classes. I chose

this approach, 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 (for example) use any type of Point for the definition of a Hologram. The abstract Point class provides all data that is needed for the Hologram's definition. 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 setup.

The Variable class is very straight forward and literally only contains an integer value, hence, it would be too much to draw a class diagram for it. 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 3 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.

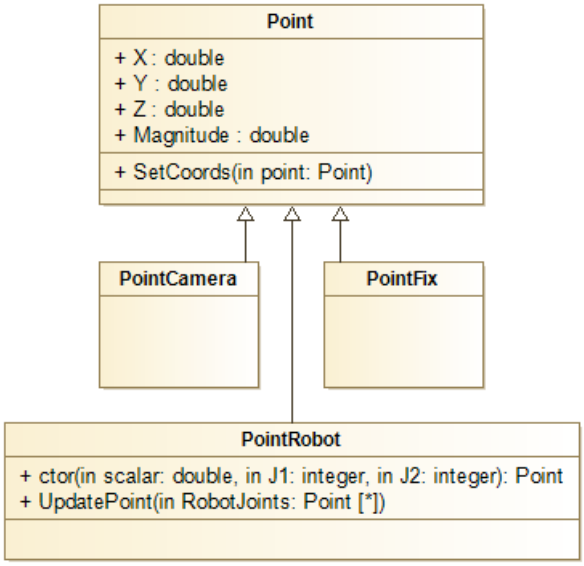

Figure 4.4: UML Class Diagram of the Point class

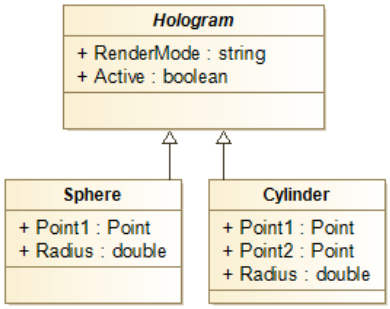

Figure 4.5: UML Class Diagram of the Hologram class

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.

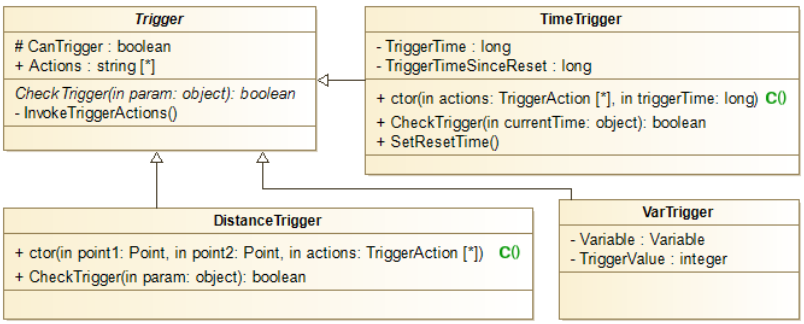

Figure 4.6: UML Class Diagram of the Trigger class

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 class structure here. 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` objects 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 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 needed three coordinates, which completes the command and makes it executable. If a `Vector6` object is given, then the command can be sent without adjustments.

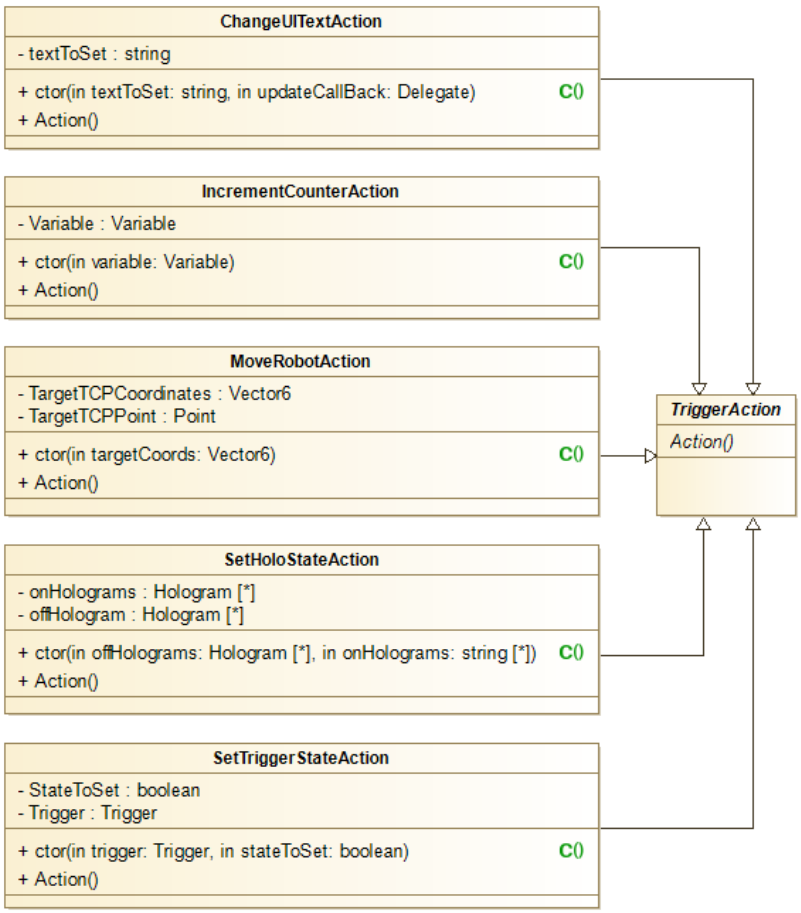

Figure 4.7: UML Class Diagram of the Action class

4.3.2 Importing and interpretation of the Parametrised Program

[This small subsection will describe how the xml is imported, how the needed information is extracted and how the described classes are instantiated - not yet finished, but will be quite short.]

Importing and validating the XML document is very easy, since the used .NET framework natively supports that. The implementation of the PARRHI Core Routine is also quite straight forward, which is why I will not talk about it too much. More interesting is the Robot's forwards kinematics, which will be presented now.

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) $\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}$ of the robot's TCP. If the joint space is six dimensional, then generally speaking (ignoring cases like singularities) the task space is also six dimensional and 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 green axes represent the robot, the black vector packs of three represent the coordinate systems $\phi_i = (x_i, y_i, z_i)$ and $\vec{l_i}$ represent the axes defined in the coordinate system ϕ_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 b.

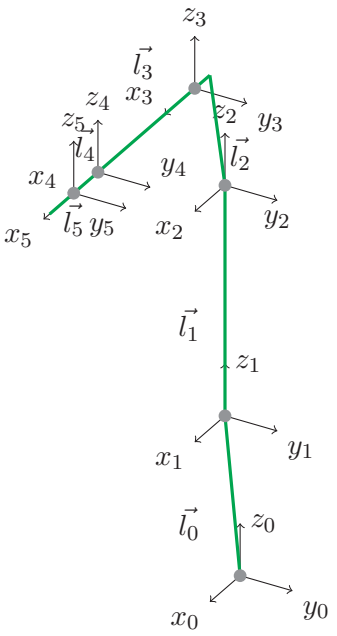

Figure 4.8: Robot forward kinematics coordinate systems

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 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 end 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 framework, it could be done quite easily by adding one additional world-coordinate system, where all robot base systems are embedded into.

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 matrix ϕ_i has to be adjusted, and $\vec{l_i}$ still stays constant. If one included translational, linear joints, the vector $\vec{l_i}$ would depend on some variable, 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 since that is how I learned it the past few years. 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. I only needed to swap the z and y coordinates of each joint position.

For further reading into this topic see [36].

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 [37]. Within recent years the company behind Unity, called Unity Technologies, made efforts to become a de facto standard for AR and VR development. Although there are other engines that have similar capabilities, I chose Unity, since I already had two years of experience while working at the chair of Automation and Information Systems at TUM, and because it is scripted in *C#*.

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

4.4.1 Image Tracking

For Image Tacking I used the Vuforia Engine [38]. This engine had a rather simple setup in Unity and worked relatively seamlessly in Unity. Compared to other engines and libraries the Vuforia Engine first of all worked, had a good documentation and a user friendly configuration. An important aspect is, that for non-commercial use it is free of charge.

The PARRHI implementation makes use of two image markers. One of them is attached directly at the robot's base and is of course 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 [39] 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 an 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 following chapter will explain how we accessed the controller's data, transferred the data to the HoloLens and generally communicated with the robot. 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 at the same chair as I am. Since his thesis is being written at this very moment, I cannot reference to any source of his.

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 where 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 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 actually setup.

4.5.1 Robot Communication Architecture

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

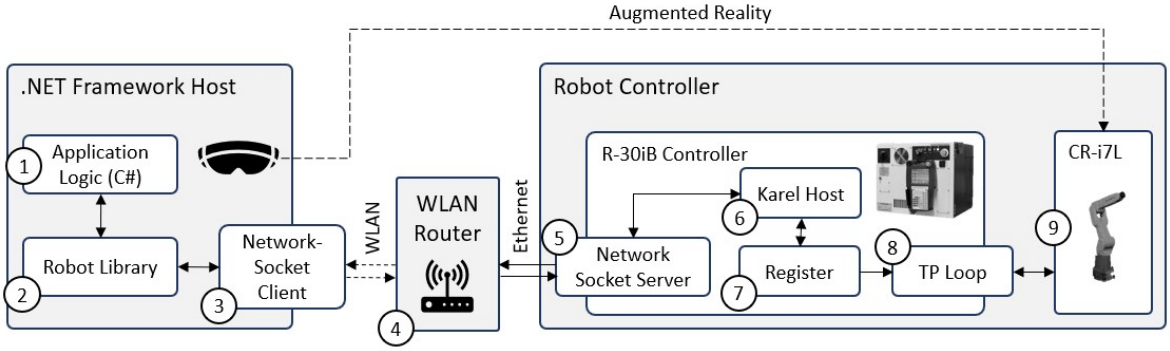

Figure 4.9: Robot library setup

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, that follows certain rules, so that both the KAREL Host and the *C#* Library can parse them for the important information.

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 CAT cable.

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

6.) KAREL Host: A programm written in KAREL [40] 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, 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 set mode.

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

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 in these two programming components.

Now 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 about 10 commands to the Application. A command could be the wish to fetch the robot's joint position, or to move the robot into a certain 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 in order 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).

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 is a simple "Fetch Command", 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 absolute or relatively to the momentary 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 relatively easy 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 basically is one large loop, that reads the Register (7) and acts on its instructions. There are some problems that 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 any more. Cases like these have to be thought of and handled.

Fig. 4.10 shows the TP Loop's (7) 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 (`Init()`) 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 a clear behaviour at each launch of the program.

The *Launch ARobot_Server* action starts the KAREL program (6), 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 an 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 mode.

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

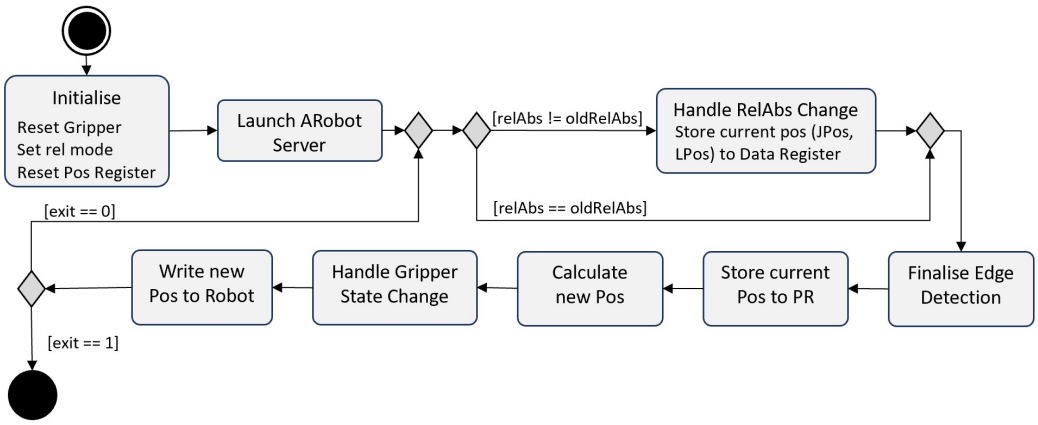

Figure 4.10

and allows the client (.NET library) to send an exit command to the robot. This allows the controller to safely shutdown the network socket and the TP Loop.

4.5.4 Robot Communication Outcome

Basically, the presented architecture allows us to execute any pre-defined function in a fraction of a second. The whole process is actually 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. Generally, this system could be extended quite easily.

4.6 Parametrised Program Implementation and Validation

One last interesting component in the PARRHI implementation is the Parametrised Program. It was clear quite early on, that the document needs a hierarchical structure and must be validated, since the PARRHI system has some presupposed assumptions about the document.

XML [34] immediately comes into mind here. 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 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 sub elements.

```
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 actually 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" active
       ='false'/>
13   <Sphere name="Sphere0" point="RobotPoint0" radius="25" renderMode="
       transparent" active='true'/>
14  </Holograms>
15  <Events>
16   <Trigger>
17    <DistanceTrigger name="DistanceTrigger0" canTrigger='true' point1="CamPoint"
        point2="RobotPoint0" distance="15.5" action1="HoloStateAction0"/>
18    <VarTrigger name="VariableTrigger0" canTrigger='true' varName="Variable0"
       triggerValue="2" action1="IncrVarAction0"/>
19    <TimeTrigger name="TimeTrigger0" canTrigger='true' timeSinceActivation="120"
        action1="TriggerStateAction0" action2="ChangeUITextAction0"/>
20   </Trigger>
21   <Actions>
22    <IncrementCounterAction name="IncrVarAction0" intVar="Variable0"/>
23    <SetHologramStateAction name="HoloStateAction0" state='true' holograms="Zyl1
        Sphere1 Sphere2 Zyl2"/>
24    <SetTriggerStateAction name="TriggerStateAction0" triggerName="Trigger1"
       canTrigger='true'/>
25    <ChangeUITextAction name="ChangeUITextAction0" text="Tutorial step nr. 4"/>
26    <MoveRobotAction name="MoveRobotAction0" pointTCP="FixPoint0" />
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. For this deserialisation a validation should be performed, in order 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 [35]. 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 in the importing / validation process. Since writing these documents can be enormously 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 to all input XML files.

Another application developed by Microsoft called ”xsd.exe” [41] 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 [42].

This means, that whenever I wanted to add an attribute to some xml element in the Parametrised Program, I only had to add these attributes in one of my example input xml documents and execute the Perl script, which 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 (depending on the error type) suggests 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 task.

5 Evaluation

Having developed the PARRHI system it is now time to ask, whether or not the end goal was actually achieved. For that, I will shortly summarize the requirements collected in section 3.1.2. The Developer (the person that uses the PARRHI system to create applications) has no to little software engineering skills but some knowledge about the robot industry. The person would like to reuse their previous work and create Augmented Reality Applications for other people to use. The application should transfer some knowledge, or give instructions on how to achieve a certain task.

This evaluation will contain two different parts. First I will try to compare traditional AR programming methods with the PARRHI system, and then I will let other people try to succeed in a specific task that was given to them.

5.1 Evaluation 1: Comparing PARRHI with traditional methods

I will first define a use case for the evaluation, then pose as the "Developer" and actually develop such a AR application. First, I will code it without the PARRHI system, programming everything manually. For future reference, this run will be called the "*manual attempt*". Then, I will attempt to achieve the same goal by using the PARRHI system. This run will be called the "*PARRHI attempt*". Both attempts will be timed and afterwards I will compare the Pros and Cons of each approach.

5.1.1 Evaluation 1: Use Case Definition

The following use case will be the base of my thesis' evaluation. As a context, I will use the same factory as explained in section 3.1.3.

The task, which will be supported by an Augmented Reality Robot Human Interface is the following. First, the employee using the PARRHI system approaches the robot work cell within a range of about 15 meters and will 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 his 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 objects in the robot factory, including the robot itself, the user's position, jogging the robot and walking round. To keep the scope of this evaluation at a reasonable level, I will allow myself to reuse the Robot-Library and the image recognition parts for the *Manual attempt*.

5.1.2 Evaluation Manual Attempt

The manual attempt is defined by programming the task described above manually. I used the same programming language and game engine as I did during the development of the PARRHI system, which means, that I could reuse the complete project setup including libraries and tools. It took about 2.5 hours to develop a reasonably good framework that allows for an easy implementation of the task. For that, I reused the Robot forward kinematics model and the robot library that handles the communication between any .NET Framework application and the robot controller.

The implementation of the actual task in the Augmented Reality application took about 1 hour. I implemented the steps relatively simply, meaning, that I did not put a lot of work into the styling of holograms and kept it at a basic level in general.

I will now summarize the downsides and benefits of the manual attempt, where unfortunately, the developer absolutely must be an experienced programmer, know the basics of forward kinematics, Unity, *C#* development, AR, Image tracking and more. 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, and might need some training time, to get up to speed.

The upside of course is, that any non-trivial task can be achieved relatively easy. If one step in the application involves complex logic or simply some actions that are not supported by the PARRHI system, the developer can simply add the feature on the fly, because they are

developing everything in source code anyway. The downside of this is, that also the most trivial processes have to be worked out in source code every time they occur. 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. That might sound easy, but that process takes about 3-5 minutes and is very error prone.

Depending on the project's architecture, one could get into trouble if some task requirements change. Reusing the accomplished work is possible, but it involves a number of steps, where each and every one can cause errors. In order to go into more detail here, I would have to explain how Unity works, which is not within the scope of this thesis.

5.1.3 Evaluation PARRHI Attempt

The PARRHI attempt means, that the same Use Case as above was be implemented using the PARRHI system. Based on the default template of the Parametrised Program, which contains the basic XML structure and defines the needed objects to animate the robot, it took about 15 minutes to complete the given use case.

The Parametrised Program for this task contains four Point definitions, eight triggers for the individual steps and 21 actions that are executed by the triggers. This use case was easily achievable with the PARRHI system, because all needed actions are supported.

The structure is very easy. There are eight steps that represent eight individual phases in the task. Each step has one trigger, that marks its end and invokes the needed actions to transition into the next step.

5.1.4 Conclusion Evaluation 1

Comparing the two methods described above, I come to the conclusion, that using the PARRHI system definitely speeds up the development process but at the same time might limit the developer's possibilities.

For trivial tasks, the PARRHI system greatly speeds up and simplifies the development process of these applications. A potential developer needs much less time to get used to the programming environment than it would be the case with traditional programming and most importantly does not have to be a software engineer to do so.

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 is due to the big simplification that had to be undertaken in the concept of the Parametrised Program, so that non software engineers are able to develop Augmented Reality Robot Human Interfaces.

This limitation really only applies to specific features that are not supported - not to e.g. a high complexity in the applications workflow. Parallelisation for example is not a problem in the PARRHI system. It would be perfectly fine to have two workflow branches running at the same time.

5.2 Evaluation 2: New Developers

As shorty described in this chapter's introduction, this section will evaluate the systems usability by letting other people try to achieve a certain task. First, I will again define a Use Case for them to achieve and then summarise their findings and feedback.

5.2.1 Evaluation 2: Use Case Definition

The task for the experimentees will be to create an AR HR Interface with the Parametrised Program that helps the User to do the following task:

1.) Approach the robot safely

2.) Jog the robot into a given position in under 1 minute

3.) Display a success or failure message according to step 2

4.) Move away from the robot into a given target area

Each experimentee will be given a 15 minute explanation into the PARRHI system, before attempting to succeed at developing the requested application. I will provide them with a cheat sheet, that shortly summarises all commands and object definitions. After they completed their task, they will be asked to fill out a short questionnaire. I will time their work, evaluate whether or not they actually achieved the goal and collect some meta information

like how long the written program was, how long it took them and how precise they achieved the goal.

6 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] M. Inc. (2019). Microsoft hololens, [Online]. Available: `https://www.microsoft.com/de-de/hololens` (visited on 04/25/2019).

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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).

[14] 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.

[15] 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.

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

[17] 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.

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

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

[20] 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.

[21] 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).

[22] 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.

[23] 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.

[24] 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.

[25] M. Broy and A. Rausch, "Das neue v-modell® xt", *Informatik-Spektrum*, vol. 28, no. 3, pp. 220–229, 2005.

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

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

[28] ——, (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).

[29] ——, (2019). Fanuc r-30ib, [Online]. Available: https://www.fanuc.eu/de/en/robots/accessories/robot-controller-and-connectivity (visited on 05/15/2019).

[30] 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).

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

[32] ——, (2019). Microsoft c-sharp, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/ (visited on 05/15/2019).

[33] ——, (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).

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

[35] ——, (2009). Xsd specification, [Online]. Available: https://www.immagic.com/eLibrary/ARCHIVES/TECH/W3C/W090430S.pdf (visited on 05/15/2019).

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

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

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

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

[40] 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).

[41] 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).

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

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, December 1, 2016

YOUR NAME