Master-Arbeit am Institut für Informatik der Freien Universität Berlin,

Dahlem Center for Machine Learning and Robotics

Development of a framework prototype for the creation of HLA-standard based simulation software for the simulation of animal groups

Gregor Barth
Matrikelnummer: 4642373
gregor.barth@fu-berlin.de

Betreuer: Prof. Dr. Landgraf
Eingereicht bei: Prof. Dr. Rojas

Berlin, 07.03.2019

Abstract

The goal of this thesis was to create a HLA-based software framework for the creation of distributed simulations which will be used in animal behavior experiements conducted by the Biorobotics Lab of the Freie Universität Berlin. For this target specifications based on the expected use cases have been specified, coupled with a market analysis. This led to the decision to develop a custom software. The thesis describes the created software, the key elements of its architecture and its main features, as well as evaluating several performance aspects.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

07.03.2019

Gregor Barth

Contents

1 Introduction

Distributed simulation software has become a common tool for handling the increasing complexity in the analysis and design of modern systems and scientific models. On the premise that no single simulation can fit all use cases, simulation frameworks were developed. The High Level Architecture (HLA) is a well-established and widely used software standard for distributed simulations and has been used in a large variety of fields, ranging from physical science models to emergency situation simulations.

The Biorobotics Lab of the Freie Universität (FU) Berlin conducts experiments in the field of animal behavior analysis and animal robot interaction. Simulation software is used to pretrain robot behavior to ease the integration of robots into groups of animals, to analyse recorded data or to simulate potential experiment scenarios.

This masters thesis aims to develop a software prototype for a HLA-based simulation framework as well as a working example of a simulation of a school of fish for the FU Biorobotics Lab First the target specifications will be established. In the following part, a market analysis will be conducted in order to decide wether an off-the shelf product or a self-developed solution will be used. Then the developed software and its main components will be presented in detail. Finally, the software will be evaluated in regards to its performance and fulfillment of the target specifications.

2 Target Specifications

2.1 Current Situation

The Biorobotics Lab of the Freie Universität Berlin conducts experiments for behavioral analysis on living fish and bees. Many experiments use robots and require a certain degree of acceptance of and cooperation with the robot from the animals. A robot displaying more natural behavior is more likely to be integrated into a group of animals. Actions pretrained or learned in a simulation give robots a more natural starting behavior.

2.2 Target Audience

The target audience is made up of three main groups of people, based on the current use cases in the Biorobotics Lab. The first group are computer scientists working with the framework to create the needed simulations for either their own exriments or experiments of cooperating biologists. Writing new simulation modules or editing existing ones as well as extending existing features as needed will be done by members of this group.

The second group will be said biologists using the simulator for their experiments or the verification of data generated by those. While good coding skills might be present, they can not be expected in general in this group.

The last group are all other users who use the software for multi-agent simulations or the visualization of data. Like for the second group, only minimal experience and understanding of coding can be expected.

2.3 Problem Analysis

Kravari and Bassiliades conducted a survey in the JASSS (Journal of artificial societies and social simulation) in which they compared various agent-based simulators [1] and formulated five general criteria to compare the simulations with. In this thesis these categories shall be used to describe simulation software and to judge how well it fulfills the requirements of this work.

Platform properties
The description of the platform properties includes information about the developing organisation, its primary domain, release system, licensing models or if the software is developed as a proprietary product or open source project. The developed software is supposed to be used in the Biorobotics

Lab of the Freie Universität Berlin. The research of the lab is based on different types of experiments with varying agent types, scenarios and parameters. In order to handle such a diverse field of application, not a single simulator is needed, but rather a simulator framework which allows to quickly create individual simulations. The framework has to provide good performance while simulating different groups of agents, good scalability and should be easy to extend in its functionality. Open-source software is preferred over proprietary preferred over proprietary products.

Operating ability
Performance of the simulation software, robustness, programming languages used and operating systems required are properties describing the operating stability of a software. As a working example and for the experiments currently run at the Biorobotics Lab a simulation of a school of fish in predetermined scenarios is required. For the most common use case small groups of around 5-25 individual agents must be simulated in a predefined and non-dynamic environment. The agent's behavior must be easily replacable. In addition, created data has to be logged into files with the .csv format or similarly common formats. At the same time a replay feature, which runs a simulation based on previously logged data, is required to verify past simulations or data from real fish. The simulation must be written in the C++ programming language to comply to the performance demands of the Biorobotics Lab. Finally, the software shall be cross-platform compatible to allow as many members of the Biorobotics Lab as possible to use it.

Usability
Usability criteria depend on the target audience the most. Adequate simplicity and learnability as well as compatibility to established software standards ensure the ease of use of the software. Only limited coding skills or none at all can be expected from the target audience. In addition the personnel working with the software could change frequently. The user interfaces have to be designed accordingly and the tool chain must be adequately documented. Coding interfaces must be well defined, stable, as few as possible and well documented as well. Configuration interfaces and other points at which users can alter the program's behavior should be as self explanatory as possible. Since a permanent customer support is outside of the project's scope, this communication with the user has to be substituted with a thorough documentation.

Pragmatics
Installation of the software should be possible with limited IT skill sets. The simulator itself and all required software libraries should be installed via simple scripts or common graphical tools like installers. If off-the-shelf software is bought, its popularity or degree of distribution is of lesser importance than technical maturity. If the simulator is self-developed, proper

testing is required to ensure maturity of its features. The total cost of acquiring or developing the software must stay within the budget constraints of the Biorobotics Lab.

Security Management
The last category describes security measures available or implemented by the simulator, as for example end-to-end security during communication, platform security or other security features. End-to-end security is not rquired at the current stage of research. Trust models between agents, as described by [2], are still not widely spread in agent-based simulations but will become more important for this project once the simulation has to manage uncertain agent data or a large number of parties participating in the simulation. At the current development stage the software does not require the implementation of above security features, but must allow for that at a future point of time.

2.4 Conclusion

Based on the discussed problems the final specifications can be drawn. A modularized simulation framework with good scalability and extendability is required. The framework must provide toolkits to create various simulations and simulation components which work together via well defined interfaces and data types. To assure cross-platform compatibility and sufficient robustness, standardized code features and designs should be used. Extended security features are not required at this stage of the research project but the software must allow their implementation in the future.

3 Market Analysis

Simulations of all kinds have been in high demand for years and it is therefor no surprise that many different simulation softwares are available on the market. Choosing an off-the-shelf product is a viable option to acquire the needed software as well as is creating a custom software based on existing software standards. In this chapter a selection of available products and software standards will be analyzed and evaluated.

3.1 Analysis of off-the-shelf products

A large variety of different simulation softwares is available on the free market. Even works and sources (e.g. [1], [3])which compare only the most relevant products already list 20 or more items. Due to the limited scope of this thesis a cut down selection will be presented. The selection contains items which represent different groups of agent-based simulation software: FIPA (Foundation for Intelligent Physical Agents) as well as HLA-standard compliant software, Java based simulators, a browser based software, products for high performance large scale simulations and a simulation framework which has been successfully used in the RoboCup competition. The available software products will be evaluated according to the criteria formulated in the previius section (2.3), platform properties, operating ability, usability, pragmatics and security management.

In addition a few criteria for exclusion are formulated. First, due to ethical reasons, no software developed strictly for military use, for example A-Globe [4] or AnyLogic [5], will be used.

Second, the budget of the Biorobotics Lab has to be considered when purchasing software. At the time this thesis was started, the Biorobotics Lab of the Freie Universität had a fincancial budget of up to 400€. This rules out any software with acquisition or licencing costs above that threshold, like for example AgentBuilder [6] (single licence costs range from 449€ - 895€).

The third criterion for exclusion is the automated generation of code via graphical user interfaces (GUI). Based on experience the generated code usually is very hard to read and maintain. This kind of code generation is also limited to agent types or code templates provided by the software. The research project at the Biorobotics Lab demands a very large variety of possible simulations and therefore would be hindered too much by this kind of limitation.

JADE
The Java agent development environment (JADE) is a free open-source sim-

ulation framework completely written in Java [7]. The development is industry driven and JADE is being distributed under a LGPLv2 licence by the copyright holder Telecom Italia. According to Kravari and Bassiliades JADE is the most popular FIPA-compliant simulation framework at the current time [1]. JADE uses a software component architecture and utilizes the agent paradigm established by the FIPA standard. The three agent attributes required by the FIPA agent paradigm have been implemented: Agents are autonomous in their decision making and show social behavior through interaction with other agents. They are proactive and do not only react to external events but rather can decide on future actions by themselves.

The behvior of the agents has to be coded in custom java files. The management of the agents as well as the controlling GUI is handled by the JADE middleware. A peer-to-peer (P2P) communication model following the specifications of the FIPA standard is used for all agent communication. Configuration data as well as agent creation and destruction can be handled via a GUI but its features are limited and no visualization of the actual simulation running is available. The last point is especially important as the software desired by the Biorobotics Lab requires visualization features, for example to validate logged data. JADE offers basic security features like Https communication user rights management.

While JADE offers a well tested and popular simulation framework complying to the well established FIPA standard, it still lacks several features required by the target specifications. The missing visualization, lacking web connection features and a very limited GUI make interaction with JADE difficult for users without sofisticated programming skills. At the same time many features JADE offers are not present in the formulated target specifications and will remain unused. All in all JADE does not meet the set requirements.

JADEX

Developed by Distributed Systems and Information Systems Group at the University of Hamburg, JADEX is an extension of the JADE framework distributed under a LGPLv2 licence [8]. It represents a collection of tools and libraries to create custom FIPA-compliant simulations instead of a finished simulation software. The agent model is based on the BDI (Belief, Desire and Intentions) behvioral model [9]. The framework already offers methods and classes to manage agents, but the individual behavior has to be coded in java classes and compiled by the user. The simulation configuration, environment data and agent configurations are stored in xml files and parsed by the framework at runtime. The general software structure resembles a hierarchical service component architecture (SCA) in order to model complex agents and agent management features. The internal time

service concept regulates time via a central coordinator in a synchronized fashion. While this type of time management would be acceptable for the current requirements, the general structure of JADEX offers few possibilities to add different types of time management.

Online and offline documentation of the API are offered, as well as beginners guides and coding examples. JADEX offers a security management for inter-agent communication. Every platform running a part of the simulation incororates a shared secret system [10]. All inter-agent communication is piped through gateways which enforce the security features like message fingerprinting and shared secrets [11]. This kind of security setup is above average compared to most offered simulation software and seems to offer adequate protection for simulations with non-sensitive data.

While JADEX offers a solid toolset for the development of individual simulations and requires no financial investment, the actual development of a simulation according to the needs of the Biorobotics Lab requires too much overhead and offers little saving of labour. The enforced BDI agent model and limited time management options reduce the extendability of the software and limit the tpyes of simulations the Biorobotics Lab could create with this framework in the future. In conclusion JADEX is not a suitable candidate.

CERTI
CERTI is an open source runtime infrastructure (RTI) implementation, developed by ONERA, the french aerospace lab [12]. It provides basic RTI functionality in a cross platform compatible software distributed under a GNU General Public License (GPL). CERTI is mostly written in C++ and, since several features defined by the HLA-standard are missing, only partially HLA-compliant. The most prominent features are the possible multi-level resolution representation of entities and the availability of a high performance variant of the code. The code structure implements the Trusted Third Party (TTP) architecture as a security measure. Communication security is provided by the use of the Generic Security Services Application Program Interface (GSS-API). The creators still maintain the software irregularly and claim to have an active open souce community around the project.

CERTI offers a solid basic implementation of an RTI with an adequate feature set. Certainly the code quality is lacking in several areas. Developing a simulation with CERTI is additionally hindered by the quality and low quantity of available documentation. These two points in conjunction make the use of CERTI for the needs of this thesis too labour-intensive and thus CERTI an unsuitable software option.

Spark
Spark is a C++ multi-agent simulation framework for 3D environments, developed and tested for the RoboCup competition for autonomous robots [13]. The main focus during its development was flexibility through replaceability of individual components.

Agents are created through a reflective factory pattern and organized in a tree structure in scene graphs. Their behavior is defined through a custom scripting language. Both these features together allow the user to dynamically add functionality to the simulation at runtime in a plug and play fashion. Spark uses the ODE (open dynamics enginge) physics library by default. It offers two time management modes, both are event-driven systems using discrete time steps. Agents require UNIX-specific capabilities but otherwise have little limits in their programming. The simulation can be split into a central simulation engine and machines connected via tcp, running agent code. Aside from code documentation, new users have access to an extensive wiki.

Spark has been used by RoboCup participants for several years and a modified version was used to run actual tournament matches in the 3D simulation league. The project is actively maintained and available as open-source software. While planned, Spark is not cross-platform compatible yet and only runs under Linux. No dedicated security features are known to the author.

While Spark is a robust and well tested software, it is mainly focussed on simulating robots in a sports-like environment. Components like game management agents, simulated sensors and actuators are included as default and 3D rigid body objects with complex physics simulations are expected. Handling or modifying these features for the specifications of this work would include an additional high workload while the use of these features for this project would be limited.

OpenSim:
Developed at the RMIT, OpenSim[14] is a simulation framework written in Java, not based on any simulation standard. The developer's main goal was to create a framework for integrated simulations, which is not as limited as established standards like HLA or FIPA, by combining agent-based simulations with other models. Serial as well as concurrent updates of shared agent objects are possible through the use of an integration manager and a conflict resolver. The former merges updates and also detects conflicting updates of these shared resources. The latter handles flagged conflicts, usually by forcing the involved simulation components to rerun the last time step with different parameters. A time manager allows for different time models to be used for agent synchronization. Certainly, no code example was available at the time of writing this thesis. No information on its degree of use in the

industry or academic community could be found.

OpenSim's main features deal with integration of very different, already existing simulation components into one system. Since the software to be developed for this thesis will be made from scratch, many problems the integration features of OpenSim were made for, can be avoided by design already. Although its feature list is impressive, due to the expected overhead and because no code is available at this time, OpenSim is not a suitable candidate.

NetLogo
NetLogo was developed as a stand-alone Java application for learning and researching with agent-based simulations [15]. It is available as open-source software wich was first introduced in 1999 and has been updated annually. The internal logic of the simulated agents is coded in the NetLogo language, which builds on the Lisp-based Logo language. Lisp offers basic simulation functionality with its turtle and patches models. NetLogo extended these features to allow the simulation of more than one agent. All new models the users create are based on that basic functionality, which limits the type and complexity of simulations that can be created. Time management is limited to a time driven clock mode with fixed, synchronized time steps (p.3 [8]). NetLogo is not officially compliant with any official simulation standard like FIPA or HLA. No security measures are know for any of the application layers.

NetLogo claims to have a "low threshold, high ceiling" approach and targets users with little or no programming experience. While it is certainly easy to learn, the overall available functionality is too limited for the requirements of this thesis, especially the agent management is not flexible enough.

NetLogo also offers a web implementation called NetLogoWeb, available as website [16]. It offers only premade models which can be adjusted. Models, GUI and underlying logic are constructed from already available building blocks and no coding is necessary. While this is very nice for teaching the basics of agent-based simulations, it is far too limited for the requirements of this thesis as well.

Swarm
Swarm is on of the oldest and most widespread open-source simulation platforms. It was developed by the Swarm Development group of the SantaFe institute in 1994 , published under a free GPL licence. [17]. The general structure models most involved components as agents objects: scenarios, observer classes, the agents themselves, etc. Agents are grouped in swarms which in turn can contain nested swarms and even model another environment for each of them. The other main design feature is the strict division between agent models and toolkits which allows the experimentation

with and obersvation of the agent models. Time management is limited to event-driven simulations advancing in discrete steps. The entire software was initially written in Objective C but based on demands of its community now also offers a collection of Java libraries. The individual agents have to be coded and compiled in Java, Objective C and internal Swarm code. This threefold interface certainly is rather complicated to use and the GUI only offers limited functionality in agent management and very limited visualization. The performance is rather poor compared to simulation frameworks based on more modern technologies and its scalability is limited [1]. No security features have been implemented so far. Swarm runs under Linux and Windows but offers no true cross-platform compatibility. The software is not compliant with any official simulations standard like FIPA or HLA.

Considering all features and limitations of Swarm the conclusion can be drawn that Swarm is not a viable candidate for the Biorobotics Lab. Creating and configuring individual agents is rather complicated and requires a steep learning curve. While the main features, the nested swarms, are an interesting and potent concept, they are not needed for the current projects of the Biorobotics Lab. The low performance, poor GUI features on the other pose a considerable problem.

FLAME
The Flexible agent modeling environment (FLAME) is a modelling environment developed and maintained by the University of Sheffield [18]. The initial software was designed for medical simulations but by now it has been used for various types of simulations, for example the EURACE economy simulation project of the European Union [19]. As of version 1.5 the software is available under a free OpenMIT licence.

Due to its medical origin FLAME was conceptualized and optimized for large scale simulations. It is therefore based around a multi-processor parallel architecture and automatically produces parallelized code which supposedly allows FLAME to utilize to computational power of super computers effectively. Possible deadlocks are handled by the use of the single program multiple data (SPMD) coding model. For additional performance, especially for 3D simulations, GPU support is available for NVIDIA graphics cards. Agents are modelled as x-machines which act as finite state machines with their own dedicated memory. All communicated data is read from and sent to a global message board through a unified message passing interface (MPI). Basic agent types and their functions are created by a template engine based on premade templates and set flags. All agent states and their transitions, communicated messages, configuration parameters, data models and other agent properties are defined and coded in xml files and C functions which are parsed at run time. The software is not cross platform compatible and includes no security features. A thorough documentation is available.

FLAME is focussed on high performance in large scale simulations on super computers and graphics cards. The features it offers in this area are well beyond what the target specifications demand though and would go largely unused. At the same time the required coding of agents primarily in XML files makes the creation of individualized simulations cumbersome and is limited by the template engine. Finally, the need for cross-platform compatibility is not met.

Conclusion
After the evaluation of several different simulators and simulation frameworks it can be concluded that none fit the requirements of this project well enough. Most of the reviewed products either focus on specific areas like medical, logistical or engineering simulations and their modification would be too labour-intensive. Additonally many require the use of built-in features which are not included in the target specifications and offer no direct use or advantage, thus again, increasing the work load unnecessarily. Finally, no platforms offered the desired modularity or scalability. This leads to the conclusion that the creation of a custom simulation framework is a desirable course of action.

3.2 Analysis of available design schemes

Even though no off-the-shelf product fulfills the requirements of this project in a satisfactory manner, not the entire software has to be created from scratch. Several design schemes are available for the creation of agent-based simulations and can be used as a basis for a custom software. In the following section two of the most widespread and actively used software standards will be discussed.

3.2.1 FIPA-AA

The Foundation for intelligent physical agents (FIPA) is a body for developing standards for the communication of heterogeneous agents in agent-based systems. It was founded in 1996 and included several academic institutions as well as many commercial companies as its members. Several software standards were created by the FIPA, including the Abstract Architecture (FIPA-AA) [20]. The abstract architecture mainly focuses in the inter-agent communication, defining structures and definitions for message types, communicated data, message encryption and other relevant aspects. The goal is to connect agents across different systems and environments through unified gateways and interfaces. The required communication functionality is offered through services whith which individual agents can have contracts. The definition of agents if abstract and quite broad; in theory anything can

be an agent, even services. Agents therefor not only represent simulated objects but also functions related to that object or rquired to manage the simulation. This design puts a high amount of functionality directly into agents and requires several software agents to represent a single physical object. Inter-agent communication is handled through an asynchronous peer to peer system with a centralized naming directory. The messaging process is quite complicated and involves several layers as well as multiple custom languages like ACL, KIF [21] or SL [22]. Inter-platform communication uses CORBA IIOP or IBM's MQ messaging system by default. Due to the lack of a centralized communication manager, an agent must know or find every single receiver of its message. This in conjunction with required multiple encoding of each message creates high runtime costs and a large overhead, especially in larger simulations. It seems that FIPA-AA was made with large-scale simulations in mind but at the same time does not offer a proper scaling concept.

So in conclusion the FIPA abstract architecture offers a design scheme for a simulation framework, but for our use case only gives very rough guide lines. Since our target software will not utilize many of FIPA-AA's features, yet a large amount of time will have to be spent implementing its complex structures or updating its outdated technologies like CORBA, it is clear that the overhead connected to implementing the SIPA-AA is too large and costly.

3.2.2 HLA

The High Level Architecture (HLA), as described by the IEEE standard 1516-2010, is an free and open international standard for the creation of general purpose simulation frameworks for distributed simulations. The design prioritizes interoperability of various simulations and reuse of models in different contexts [23] in distributed simulations and has become a well established standard in the industry.

The main structure is a service bus topology which connects a central management with various simulation participants, called federates [24], via loose coupling. The centralized management handles communication, plug and play of components, data distribution, time mangement and other essential tasks. The actual agents are managed by the federates. Inter-federate communication consists of pure data exchange, the reasoning behind a federate's action is not communicated. Messages are communicated as serialized object packets via a publish/subscribe system. Only the communication interfaces between federates and the central management is defined, but otherwise the standard imposes almost no restrictions for the internal coding of federates or agents.

The framework itself scales well for simulations of different sizes [25, p.9]. Only the PCI communication can become a bottleneck with increased communication volumes, but solutions like shared-memory frameworks exist [26], basically allowing the software to use parallel simulation techniques. No specific commnication security measures are part of the standard, but can be added.

The required features, as defined in the problem analysis (2.3), are a subset of the features HLA offers, but the unrequired features do not need to be implemented, which means the required work scales well with the size of this project. In addition, the framework offers enough modularity for further developments of this software to meet the needs of future projects.

3.3 Conclusion

In this chapter eight different simulation software products and two well established software standards have been examined. All eight off-the-shelf softwares did not meet the target specifications set for this project in an acceptable manner. Most options had to be ruled out due to the lack of essential features, low code quality or limited extendability. Too much overhead and the need to implement unused features were also main issues with most contestants, but especially with the FIPA-AA standard. The HLA standard offers all features listed in the target specifications and at the same time does not require the implementation of unused features, decreasing the development cost considerably. Since the HLA standard is available for free and offers enough modularity for our needs, it is a valid base for the project. A custom simulator framework based on the HLA standard will be developed.

4 The HLA-Standard

4.1 Brief history of HLA

Developed by the Defense Modeling and Simulation Office (DMSO) and the US Department of Defense (DoD) at the end of the 1970's, computer-based simulations became more popular as an tool for military analysis. SimNet was one of the early US defence programs to develop a distributed simulation network in the 1980s. When the program ended, the produced software was capable of simulating military platoon-level scenarios [27] distributed over ceveral computers. As part of SimNet the Distributed Interactive Simulation protocol (DIS) emerged and tried to define a coherent format for messages passed between heterogenous simulations. It achieved that by defining standardized message formats, the Protocol Data Units (PDU) [28]. Although various PDUs for different purposes were defined in the IEEE Standard 1278, the focus was still strictly military.

While the priority of SimNet was the training of the connection between weapon and operator, aggregate simulations for command level training were needed. This led to the development of the Aggregate Level Simulation Protocol (ALSP) in 1990. ALSP set the foundation for new concepts later used in the High Level Architecture, for example time management, interactions and a separate infrastructure model [29].

Based on the experiences with DIS and ALSP the development of the HLA was started in 1995. In 1997 the goal to create a platform for connecting different simulations in one framework was achieved with the release of the first version. After the release as IEEE standard in 2000 (IEEE 1516-2000), the HLA gained wider visibility and acceptance which also led to civilian use of the framework [30]. The standard is updated to this day, with the latest revision being from the year 2010.

4.2 Modern application of HLA

Despite being of strictly military origin, the HLA has also seen widespread use for many different civilian simulation purposes. The main users are the US military [31] as well as several NATO partners like Germany [32]. Civil uses include simulations run by the NASA [25], medical simulations [33] or manufacturing optimization [34].

4.3 Overview of HLA

HLA combines a composable set of interacting, heterogeneous simulations in a distributed simulation [31, p.142]. Oğuztüzün and Topçu define a distributed simulation as follows:

"[...] distributed simulations, which are composite simulations that consist of multiple individual simulations distributed over a network of hosts." [35, p.10].

Three IEEE-Standard documents define the High Level Architecture: the framework rules [23], the interface specifications [36] and the Object Model Template [37]. As can be seen in image 1, the HLA framework uses a service bus topology. The required three components, the Runtime Infrastructure (RTI), the Object Model Template (OMT) and participating applications, called federates [38, p. 2295], form a distributed simulation, called a federation.

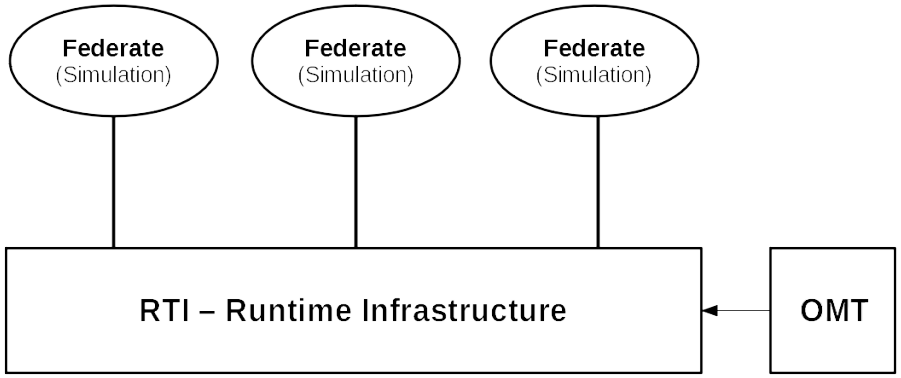

Figure 1: Basic structure of the HLA design

Ten basic rules, to which all participating applications must adhere, define the runtime procedure of a simulation and basic communication procedures.

All participating applications are called federates. Federates can fulfill many different functions: they can be a complete simulation, a supporting utility like a data logger, or an interface to a live player. Federates are the elements which create, communicate and process all data.

The Runtime Infrastructure (RTI) functions like an operating system for the simulation. It checks if all elements follow the ten HLA rules and provides

functionality through eight basic service groups to handle inter-federate interaction and runtime management of the simulation. These include methods for time management, data distribution or object management. All these services are accessible for federates through well defined interfaces.

The Object Model Template (OMT) defines which information will be communicated within a simulation and how they have to be defined and documented. It consists of three types of elements: the Federation Object Model (FOM), Simulation Object Models (SOM) and the Management Object Model (MOM) [37, p.20]. The OMT provides a standardized form and syntax for the creation of these object models, usually XML files, and their documentation in table form. The actual content of object models can be defined freely. The FOM describes all data which can be exchanged within the federation. Additionally, each federate has its own SOM describing which federation it wants to join and detailing which data it provides and requires. Every SOM must be a subset of the FOM to ensure a functioning communication. The HLA interface specification defines the MOM as follows:

"The MOM provides facilities for access to RTI operating information during federation execution. These MOM facilities can be used by joined federates to gain insight into the operations of the federation execution and to control the functioning of the RTI, the federation execution, and individual joined federates." [36, p.247].

It thus allows the creation of managing federates which can control how the RTI works or how much access to the RTI's services other federates have.

The technical definition of communication within the HLA framework is not very detailed and leaves the choice of protocol to the implementer [39, p.824]. Possible protocols are Unicast UDP/TCP or Multicast. Two types of messages can be sent: object updates and interactions. Object updates contain data about objects which persist within the simulation, for example agents. Interactions are one-time messages, sent and received without a direct relation to a data object.

Several important aspects of a software are not defined by the standard on purpose. While code examples exist for C++ and Java, no specific langugae is required for any implementation [31, p.143]. In addition, no specific operating system is required and the implementation of a federate is completely left to the user.

5 The Simulator Software

5.1 Overview

In this chapter the general aspects of the implementation of the HLA-standard for this prototype and the used tools shall be described. Details of the implementation and a description of the produced software modules are discussed in the following sections.

C++ was chosen as programming language for this project, as it is a well documented and widespread coding language and offers all required features as well as sophisticated options for performance optimization. In addition all software at the Biorobotics Lab the simulator will interact with either already has been or will be coded in C++ or will have a wrapper provided. The simulator software is available for download through a git repository and can be built with a streamlined build proess based on cmake. Additonally required software packages are documented and can be installed automatically through provided shell scripts. While not part of this thesis, it is still worth mentioning that the entire tool chain is regularly tested on a continious integration server within the lab.

Since the goal of this thesis is to produce a HLA compliant software, the degree of compliance has to be analyzed. The implementation of this HLA-based prototype in general follows the ten rules defined by the IEEE document [23]. But since the developed software is a prototype and time constraints as well as a limited availability of workforce have to be considered, three of the defined features and rules either have been simplified or left out.

Rule 1:
"During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification." [23, p. 21]

The above rule 4 has been simplified in this prototype. While in general the interface specifications are adhered to, some of the optional parameters have been omitted from functions when they were not needed. All parameters strictly required by the specification have been used though, to ensure HLA compatibility.

Two other rules are also deviated from:

Rule 8:
"Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs." [23, p. 22]

Rule 9:
"Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of instance attributes, as specified in their SOMs." [23, p. 22]

Both rules concern features which are not needed in this version of the software and therefore do not apply. The required modules of the RTI have not been implemented. Still, the required features can be easily added at any point in the future.

The HLA standard also implicitly requires the presence of several software modules to provide required functionality. The MOM provides service calls to gain information about the current state of the simulation or to change it. Methods to change the simulation state for example include service calls to subscribe or unsubscribe federates to object classes or to change ownership of objects. This functionality should be bundled in a manager federate which executes the appropriate calls without involving the federates. Changes in a federate's access to information or external influences on the simulator world can be modelled with this omnipotent federate. At this point this kind of functionality is not required and therefor no manager federate has been implemented yet, but could be added at a later point of time.

5.2 Simulator Structure

This implementation of a HLA framework is divided into three distinct parts: the RTI, the individual federates and the group of classes providing the connecting interfaces and base classes between the two, called the Lib-RTI. This structure allows to modularize the code very well and adheres to the basic HLA structure of RTI, data bus and connected federates. The RTI can be updated and optimized without any need for change in the federate code and vice versa.

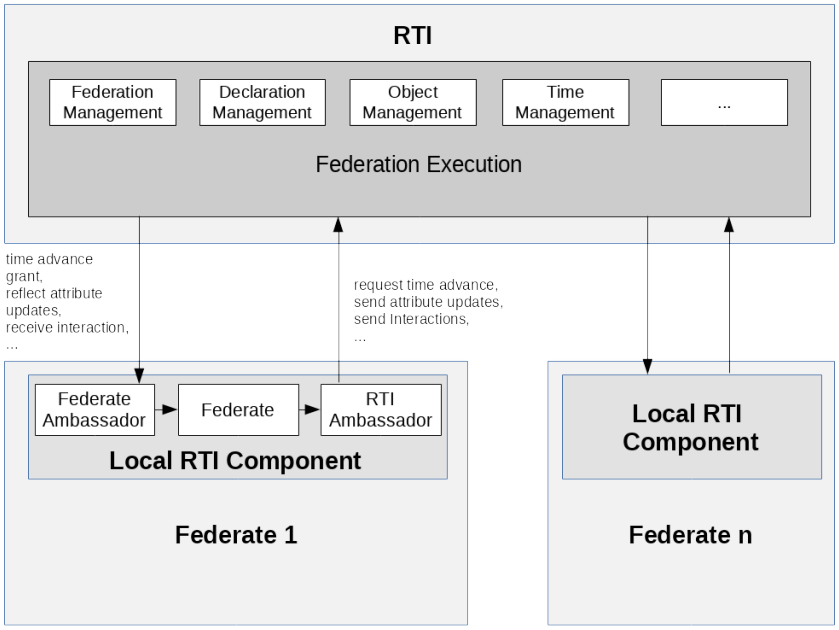

Figure 2: Basic structure of the communication between RTI and federates

One of the defining features of the HLA standard is its communication model. While no specific protocol is enforced, the general structure of the communication is supposed to allow many federates with vastly different tasks and implementations to communicate with each other. Three basic options are commonly used for distributed simulation applications: Client-server model, peer to peer (P2P) and hybrid peer to peer [7, p. 2].

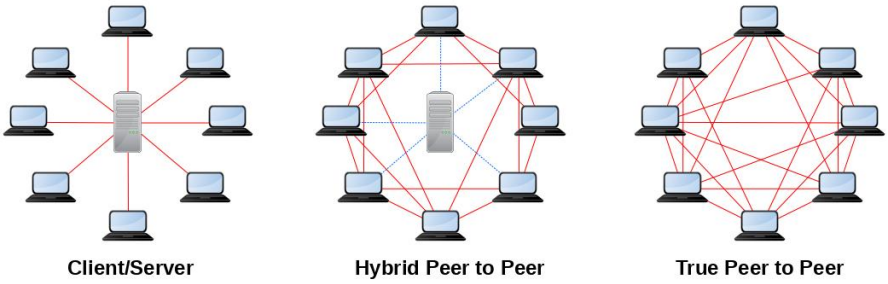

Figure 3: The three main communication network types

The client-server model makes a distinction between the client and the server role. Servers provide services but are completely reactionary and wait for client requests. Clients define the flow of the system and initiate all communication. They can communicate with the server, but cannot speak to other clients directly. At the same time clients can join and leave a system while servers usually provide a certain guarantee of stability. The internet is a typical example of this kind of structure. The opposite would be a peer to peer system. It contains no distinction between roles and all participants can initiate communication, send or answer requests. Every node can enter or leave the system at any time, as well as discover other nodes which creates a fully distributed system. In contrast to the client-server system, in which clients must know their servers but no other clients, a P2P-system must provide all mechanisms for discovering other clients and being discovered, entering and leaving as knowledge about other clients can be distributed completely arbitrarily amongst them. True P2P networks include fully autonomous clients and are completely decentralized. Its complexity can grow exponentially with additional clients joining, making it hard to ensure coherency of the system. An offspring of true P2P systems are hybrid P2P systems. They include special nodes which offer services that simplify the lookup of information about clients and their offered services. This not only reduces network traffic but also increases the security of the system as all clients have to register with these special nodes. Certainly, the robustness of the system depends on the availability of these special nodes, making them a primary target for attacks.

The HLA standard defines a structure resembling the server-client model. A central RTI functions as server and federates as clients. While it is possible to create a hybrid P2P network within in a HLA simulation, as the EODiSP project has shown [40], the expected scale of the simulation run in this prototype did not warrant such a design. Therefor, a basic client-server structure was chosen.

For the implementation again a modular apporach was chosen. All network communication functionality is bundled in a connector module which handles the client and the server side of the communication. The federates and the RTI communicate through remote procedure calls (RPC), which are sent as TCP/IP messages. An external library, the librpc[1], was used for the implementation of the rpc communication as it offers a C++ RPC framework with little overhead and without automatic code generation. With this setup a satisfactory decoupling of the simulation logic and networking modules was achieved.

While the RTI and the overall structure of the simulation framework have to follow the rules set up by the HLA standard, federates offer much more options for their design. Aside from the requirement to implement the Lib-RTI, the federate have no preset internal structure. All communication to and from the RTI goes through two ambassador classes to enforce a uniform interface. This freedom of design also led to the decision put several essential simulation functions into federates. Logging, visualization or the handling of specific user input are just a few examples. Such a specialized federate can be adapted to any simulation and its needs specifically. It can also incorporate an external library and use its functions as required by each individual federation. The modularization and decoupling resulting from this design choice overall make the software easier to update and maintain.

[1] http://rpclib.net/

5.3 RTI

The runtime infrastructure is the central management unit of the simulation. All of its managing abilities are accessed by federates through service calls which are grouped into seven groups: federation management, declaration management, object management, ownership management, time management, data distribution management and support services.

The RTI itself can be a single software module or can be split into a central runtime component (CRC) and a local runtime components (LRC). This split allows to distribute the computational load by running each LRC on a different node or machine. The EODiSP project used this structure for its HLA implementation [40]. The design choices made during the implementation of the RTI can heavily influence the overall performance, as Watrous et al. [42, p. 2ff] have shown. The amount of required messages during mandatory steps like federation creation, object declaration or ownership management depends largely on the choice wether the RTI is implemented distributed or in one module. As Watrous et al. showed, a non-distributed approach improves performance in most use cases and since the since the expected default use cases in the Biorobotics Lab do not involve simulations with agents with high individual computational cost, this approach was chosen.

In the following section the function of the RTI service groups and their general implementation shall be described.

Federation Management
The first group of service calls, the federation management, handles the two most fundamental interactions with federates: joining and resigning. Additional functionality like creating synchronization points and restoring simulations from thems are also in this group, but were not implemented in this prototype.

Declaration management
Every federate joining a federation has to declare his intentions on generating and requiring information as defined in its FOM files. A federate can publish information, subscribe to it or do both. The declaration management offers services for federates to announce which object classes and interactions they want to communicate in which manner. Additional methods also allow federates to unpublish or unsubscribe in the case they are leaving a federation. In order increase performance, joined federates, object classes, attributes, interactions and their respective handles are stored centrally, giving the RTI a ledger function. The alternative, distributed approach, letting each federate store data about existing other federates, makes federates less dependent on the RTI but increases the communication volume considerably. The HLA

standard offers an advisory system in which the RTI advises federates on wich data to publish, subscribe or register, but as Watrous et al. showed [42, p. 3], the cost of such a system can be very high. Therefor the centralized approach was chosen.

Object Management
This group of HLA services deals with the registration, deletion and updating of object instances and the sending and reception of interactions. Object instances represent concrete agent objects belonging to an object class. Distributing data updates about object instances makes up the bulk of the network communication between federates. These methods in this service group form the primary data exchange mechanisms. [36, p. 101] After every federate has declared which object classes it publishes and subscribes to, individual object instances have to be made known to the rest of the simulation. An object instance can represent any type of agent. Other federates can discover object instances which qualifies them to receive data updates for the objects. The object management also provides service calls for federates to request data updates from other federates and to change settings concerning the communication behavior of individual simulation participants. Aside from the latter, all methods are elementary components of any HLA simulation and have been implemented according to HLA specifications. It is important to note, that while the RTI stores handles of object instances and attributes, it does not save any data values. The size of the data cannot be known beforehand and could cause serious performance issues.

Ownership Management
In order to ensure data consistency the HLA standard introduced an ownership concept. Agents and other simulation objects which are communicated amongst federates are owned by a federate, as specified by framework rule 5:

"During a federation execution, an instance attribute shall be owned by, at most, one joined federate at any given time." [36, p. 21]

This helps to avoid inconsistencies from several federates publishing information about the same object and increases the security of the simulation. The RTI's ownership management offers services to change the ownership of objects and attributes, as also enforced by rule 8 (5.1). A practical example would be a traffic simulation with federates managing single busses and their passengers. Upon chaging from bus to another, the ownership of a passenger agent would be changed as well.

The simulation in this thesis only includes one federate generating agent data and during the execution of the simulation no situation requiring an ownership change will occur. Therefore no ownership management has been implemented. This, together with the RTI checking the ownership of pub-

lished objects upon their registration with the RTI, makes the simulation automatically fulfill rule 5.

Time Management:
The purpose of time management is to synchronize the federates participating in a simulation, coordinate their advancement of time and deliver messages between federates in a coordinated fashion [35, p. 64]. Upon entering the federation each federate aligns itself with the time axis of the RTI, creating an internal, logical time for itself. All messages being sent among federates include a timestamp, which allows the time management services to enforce the distribution of messages in an ordered way. Time in the simulation can stand in one of three relations to wall clock time [31, p. 145]:

paced, independent time advances: Federates pace their internal time to match the wall clock time, but do not coordinate it with the RTI or other federates. Typical use cases are "Human-in-the-loop" federates.

paced, coordinated time advances: In this method a federate's logical time is a derived function of wall clock time as well, but also synchronized with the RTI. This apporach works best for simulations including interaction with physical hardware or live human interaction.

non-paced, coordinated time advances: Federate times are not synchronized with wall clock time but are coordinated with the rest of the simulation. Analytical simulations designed to just finish the program execution as fast as possible often choose this approach.

Aside from the relation between simulation time and wall clock time, a strategy how to advance time has to be chosen as well. The HLA specifictaion again offers three options: step-based, event-based or optimistic advancement [43, p. 45].

step-based advancement: Federates advance time in steps of fixed length. They calculate values and handle incoming messages up to the point of time marking the end of the current step.

event-based advancement: All communication in this mode must be timestamped. A federate only advances time to the timestamp of the next incoming message. The execution of federate code is therefor not based on fixed time intervals.

optimistic advancement: The first two options are also known as conservative time management. Optimistic time advancement is not restricted by the behavior of other simulation participants. The federate will process incoming messages regardless of timestamp order and will calculate values ahead of the simulation time index. This will generate events in the future of the RTI time index. If the optimistic federate receives updates which invalidate

some of its own messages or processing steps, it will revert back to its state at the time index of that message.

The last configurational component of the time management is the setting wether a federate is time-regulating, time-constrained, both or neither [35, p. 65]. This setting regulates, how strongly a federate is involved in the time management of the federation. Only a time-regulating federate can advance time for the simulation. It does so by sending messages with timestamps and by asking the RTI for time advancement, depending on the time advancement strategy chosen. Time-contrained federates can not send timestamped messages and depend on regulating federates to advance their logical time. It is possible for a federate to be both, regulating and constrained, at the same time. They can regulate time normally but are also dependent on other regulating federates. By default a federate is neither and does not actively participate in the time management of the federation. Such a federate has complete freedom in how they manage the advancement of their own time. The HLA interface specification offers services for federates to change their status during runtime.

After the different possible mechanisms of time management within a HLA-based simulation have been described, it must be determined which of them are used in this prototype. Two types of relation to wall clock time are supported: *paced, coordinated* and *non-paced, coordinated*. If the simulation is visualized, the first mode is used. The user must be able to follow the visual output and therefor all federates advances with a set frame rate. If the headless mode is used and no visiualization is involved, the simulation will try to finish its calculations as fast as possible, using the unpaced and coordinated mode. Only one federate, the master federate, is *time-regulating*. It sends the requests to advance time to the RTI which then advances time for all other federates. In this way all components are synchronized and use a *step-based time advancement*.

Data Distribution Management

The origin of the HLA framework are large scale military simulations, including many different types of agents dispersed in big scenarios. As every involved federate would send and receive updates for the agents, network traffic would inevitably increase exponentially. In order to reduce the network load, data distribution management was introduced. It allows agents to be dynamically assigned to virtual areas, which can correlate to areas in real world 3D space, but do not have to. Information shared through updates can be declared exclusive to certain regions, limiting the number of receivers and in turn the amount of network messages sent. The base scenario of this prototype involves a single fish tank with a size of 100cm x 100cm, populated by one to two agent classes. With a maximum of six federates communicating at a time, the amount of messages sent is low. It

can be concluded that a complex data distribution system is not necessary at this stage of the project. The code structure still allows the introduction of data distribution management at any future point.

Support Services
This group contains miscellaneous services primarily handling lookup functions, for example which object classes a single federate has subscribed to. Aside from these implemented methods, the HLA standard also defines services to manipulate regions and to check update rates. Both of these have not been implemented as this prototype does not require that functionality.

5.4 Lib-RTI

The Lib-RTI consists of several components: a federate base class, a runtime infrastructure ambassador (RTIA), a federate ambassador (FedAmb) and several data type definitions and utility modules. Its purpose is to provide a connection endpoint for the communication between federates and the RTI, as well as ensuring that all federates possess the basic, required functionality to interact with other components of a HLA simulation. Sometimes this group of classes is also called Local Runtime Component (LRC), for example in the EODiSP project [40].

The federate base class defines elementary functions every federate needs: joining and leaving a federation, parsing configuration and SOM files, declaring objects and communication behaviors. Every custom federate must inherit from this base class to ensure compatibility with the rest of the simulation.

A runtime infrastructure ambassador provides methods to call all functionalities of the RTI through a defined communication interface. It streamlines the messages and method calls between heterogeneous federates and the RTI, but does not handle encryption or actual network management. All messages to other federates are sent directly to the RTI and distributed from there. This apporach improves performance as it guarantees that a federate sending data updates does not need to send additional messages if more federates subscribe to its data.

Federate Ambassadors contain the other endpoint of federate-RTI- communication. Requests, interctions and callbacks made by the RTI to the federate are handled by this class. All calls which can be handled in a generalized way are processed directly in the federate ambassador. Other calls are translated into a format the individual federate can handle or are forwarded to the appropriate internal functions.

5.5 Federates

A federate is a simulation component providing a distinct functionality set to the overall simulation. As its definition is rather vague, it can therefor fulfill a variety of roles. While a single federate can include all desired features of a simulation, it is often more feasible to use several federates in conjunction and give each one a specific task. In order to ensure that all federates get incorporated into the simulation without any problems, a basic interface for a federate is needed. Every single federate must include a tiny library described in the previous chapter (5.4), called the Lib-RTI, which provides the basic functions of a federate.

5.5.1 Master Federate

The master federate handles several management functions for the federation. It is not an official requirement in the HLA-standard to have such a managing federate, but it is easy to see why bundling such tasks in one place is reasonable. Through a parsed configuration file the general setup for the simulation execution is defined. It contains the scenario to be loaded and the framerate at which the simulation will run. Any desired additional configuration parameter can be added in this way, if future projects require them. If any of these information has to be communicated to other federates, for example which scenario has to be loaded, the master federate will do so through interactions defined in its FOM file. The information which interactions all other federates must subscribe to in order to interact correctly with the master, is included in the documentation of the simulation.

After the initialization of the simulation the master manages the start and stop of the execution by handling according interactions it subscribes to. These interactions can be based on any type of user input, console commands, GUI elements or others and will be discussed in the following sections. A standard update loop, as is common in simulation software or also computer games, is used to execute the simulation.

As already described in RTI section (5.3), this prototype uses a time-stepped, synchronized time model. The master federate is the only time regulating federate in the federation, which means it is the only federate initiating time advances. It does so by issuing a time advace request to the rti in every update step. While every federate may have its own internal time, only the time of the master federate regulates the flow of the program and advances time in regulated steps. Hence, the simulation is step-based. The RTI receives the time advance requests and if a time advance is granted, forwards that to all federates. Because all federates always advance the same

amount of time, in our case one frame as enforced by the master federate, the simulation is synchronized.

5.5.2 Animal Swarm Federate

The Animal Swarm federate is the federate the entire federation is built around. Its purpose is to generate and publish data from simulating a group of ca. 1-25 fish agents in a scenario resembling a fish tank. The federate structure is split into two main parts: the federate class and the agents on one side and the agent's behavior on the other side. The actual animal swarm federate class is responsible for creating and initializing all agents based on settings from a configuration text file. It also advances the time for each agent based on the RTI time grants and collects data about the current state of each agent, which is then published to the rest of the federation. All agents implement a base interface to ensure compatibility. The animal swarm federate knows which data fields the agents hold and how to access them. The actual calculation of the data happens in the external behavior modules, invisible to the federate. This division of agent and behavior logic allows to create modular agent groups and quickly switch out behaviors of agents without the need to recompile any code. This additionally creates the compatibility to run different types of agents with the same behavior plugin. The Biorobotics Lab employs robots for some experiments, which have been interfaced with the simulator in this way.

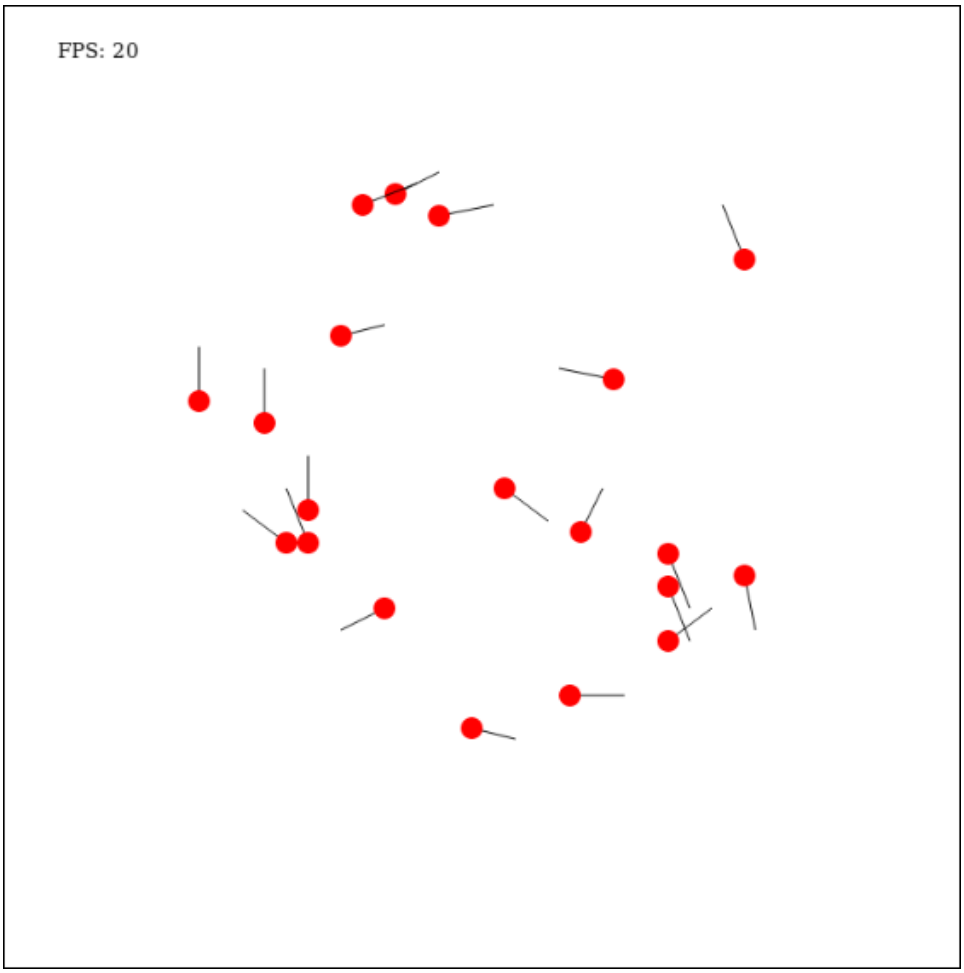

Figure 4: A simulation of 10 individual fish agents.

The prototype at the moment only offers the possibility to load behavior plugins made with the Qt[2] library, but interfaces and loaders for other plugins types can be added in the future. The behavior of the fish agents can be modelled in several ways but for this project an apporach based on the work of I. D. Couzin[44] was chosen. The paper has over 1000 citations and the described model can therefor be seen as well established. Within the Biorobotics lab it is also used as a sample model. The implementation of this model[3] was done by M. Maxeiner, a member of the Biorobotics lab.

Since it is not the focus of this work, the Couzin model shall only be described briefly here. In short, the model divides a fish's behavior into three states,

[2]https://www.qt.io/
[3]https://git.imp.fu-berlin.de/bioroboticslab/robofish/ai

based on zones. Every fish has three spherical zones around him. If other fish enter the innermost zone, the central fish will increase his distance to them. It will align itself towards fish in the second zone and try to close the disance to fish in the outer zone. These three simple behavior states or rules create a cohesive school of fish.

Scenarios are an essential part of any simulation as well, representing the world all simulated agents act in. The HLA standard does not enforce a distinct way of handling scenarios, but instead offers ways for federates to communicate about scenarios. This leads to the conclusion that the federates have to handle scenario management themselves. Like other federates, the animal swarm federate receives the information which local scenario file to load through an interaction from the master federate. This central distribution ensures that all federates run the same scenario. Uniform description languages for scenarios do exist, for example the MSDL [41], but are very complex and would entail a large overhead for such simple scenarios as are currently used in this project. The scenario file is written in the XML format and contains all necessary information about the scenario: dimensions of the simulated world, static objects in the scenario world, etc. In this prototype the scenario is an empty fish tank with the dimensions of 100cm x 100cm.

5.5.3 Visualizer Federate

Quite naturally computer-based simulations generate large quantities of data. In order for humans to understand, filter and process these large quantities of data a good visualization is required. At the same time every simulation produces different kinds of data which require different kinds of representation. This leads to the conclusion that even for a software prototype running a simulation with rather low complexity, a visualization framework which allows to easily create many different visualizations, is required. The visualizer federate is structured to provide that kind of framework.

Since the visualizer is a separate federate, possibly running on a different computer than the data producing federates, the coupling between the simulation model, and the visualization model has to be considered. The simulation model includes the current state of the simulated data, values of agent attributes, settings and most importantly the time index. The visualization model describes the current state of the visualization. There are two ways in which these models can be coupled: online or offline [45]. Online coupling means the simulation and its visualization work in parallel and need synchronization. In offline coupling, sometimes also called post-run coupling, the simulation finishes all data creation first and the data is only

visualized afterwards. The FUSimulator is capable of both modes as the visualizer federate does not specify a data source. Live updates can be handled as well as prerecorded data from local files. It also has to be considered that the coupling is not limited to 1:1 relations. Rather a $m : n$ relation between m data producing federates and n visualization federates can be generalized. Each visualizer filters the data it needs for its own view of the simulation. The communication required to implement the coupling can be handled completely by HLA service calls. Attribute updates and interactions allow for synchronized data exchange and impose no limits on a $m : n$ relation between communicating federates.

While only one visualizer and two data producing federates will be present in this prototype, it is still desirable to create a structure which utilizes the distributed nature of HLA-based simulations. Ideally, the visualization is not tied to specific hardware as data producing federates might be. Furthermore several users at once should be able to access the visualization of a running simulation, with as few limitations as possible in regards to the devices used. An easy to imagine use case would be any simulation running for several days. Therefor the chosen approach to accomplish this versatility is to use a web-based visualization which renders a graphical user interface through HTML. The minimum requirement for any device would be the ability to run a web browser, which most modern computers, tablets or smartphones fulfill. In addition, the visualization data could be broadcasted to numerous receiving devices, enabling multiple users at once to review the data of a simulation.

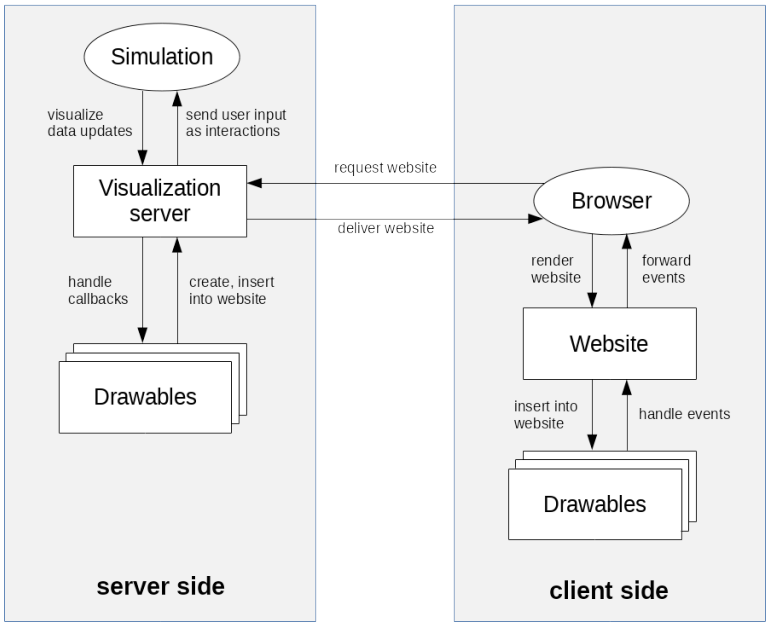

Figure 5: Basic structure of the visualization framework

The implementation of these requirements is as follows: The basic structure is divided into C++ server side and a web-based client side. On the server side the visualization federate starts a websocket server instance which handles all connections. The open-source websocketpp library[4] is used for the implementation of the required features. By entering the specified URL in the web browser, the user sends a request to the websocket server and in turn will receive a website. The served website is written in HTML but is dynamically created through C++ code in the visualizer federate. A small library of common HTML objects like buttons, lables or a canvas is provided and allows the user to build a website to his desires. Using standardized HTML code here ensures modularity and robustness. All objects are inserted into the HTML code of the website automatically. Once the website has been sent out to the browser, it can be rendered like any other webpage.

[4]https://www.zaphoyd.com/websocketpp

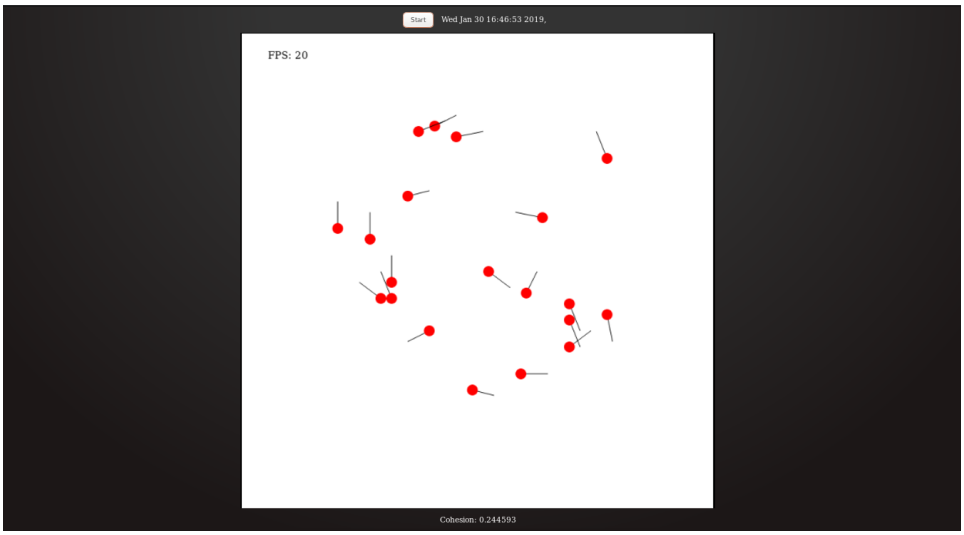

Figure 6: A simple web GUI for the simulation.

Of course, some form of interaction between the web elements and the simulation is required. To achieve this, the built-in interaction methods of HTML objects are used. While methods handling click events, changing captions and others are already present on the client-side HTML objects, they need to be mirrored on the server side as well. Therefor every object can be given callback methods for certain events like user input. As example, a button to start the simulation can receive a callback method for its onClick event. Every time the button is clicked, it will send a signal to the server which then runs the correct method bound to that event, in this case a method starting the simulation. The events are processed through a javascript file added to the website. They are communicated through JSON objects which are parsed by Javascript on the client side and by the json_pp library[5] on the server side. The callback C++ methods are stored as function pointers and evaluated by the visualizer class.

This setup allows the user to program many different web-based GUIs, according to his needs. For this prototype a limited selection of objects has been implemented, but all objects of the HTML standard are possible. 3D visualizations could be added as well by using the WebGL library. The resulting web page can be accessed through any regular web browser and allows more than one user at once to do so. The downside of this approach is the dependance on the rendering pipeline of the individual browser. For simulation visualizations with critical time constraints or a dependance on exact real time rendering of complex models, a different approach, for example with a local rendering engine, would be more advisable.

[5]https://github.com/deftek/json_pp

5.5.4 Statistics Federate

All federates so far either produced or subscribed to data, but the statistics federate is an example for a data processing federate. It subscribes to agent data and calculates statistical values based on it. The calculation results are then published again and visualized by the visualizer federate. In the case of the fish agents, the statistics federate calculates the average distance between the fish.

5.5.5 Terminal Controller Federate

Using a GUI is only one option to handle user input. The two most common options for user input are graphical inputs and input via a command console. A use case for the latter would be running the simulation for pure data generation without the need for any visual representation. The terminal controller federate handles the second use case and is an example for alternative control methods available to the user.

After the simulation program has been started, the terminal controller asynchronously waits for user input to actually start the simulation execution; in this case only a simple string input is required. The commands for starting or stopping the simulation execution are published as interactions, similar to the visualizer's behaviour. Other input methods are possible and could be implemented in the same way by creating a dedicated federate for them.

5.5.6 Logger Federate and LogPlayer Federate

One of the most common demands for simulation software is to persistently store generated data. The logger federate fulfills this demand and can write any data the simulation produces into text files; in the case of the FUSimulator position and orientation data of all fish agents is logged. A uniform logging format allows the comparison between data created in a real world experiment and data created from a simulation. In addition, a replay function provides the ability to analyze and debug recorded data. Theoretically, usecases for many different file formats are possible, but in this prototype the CSV-format was chosen as it is a widespread and standardized file format. The federate itself can log any data which can be communicated within the simulation or it generates itself. The current default behaviour is to log data it subscribed to through its FOM file, but other options like custom configuration files are imaginable.

The logplayer federate provides the reverse pipeline compared to the logger

by channeling data from a log file into a running simulation. In a default simulation setup at least one federate creates data and provides it to the other simulation participiants. The logplayer replaces these federates and data values from a text file are distributed to the simulation instead of live generated ones. The technical implementation is rather simple: During its update steps the logplayer parses a CSV file line by line with basic C++ I/O functions and converts the read data into HLA-compatible formats. The values are then distributed to the simulation through regular HLA-based publishing methods.

6 Simulation Evaluation

In the previous sections the target specifications and the implementation of a software prototype for a HLA-based simulation framework has been described. In this section the software will be evaluated in regards to the features demanded in the target specification section (2). The performance will be analyzed, especially the scalability, as growing numbers of agents can create various bottlenecks during code execution. Finally, the extendability will be evaluated through the example of a student project.

6.1 Feature Evaluation

The target specifications list three main technical requirements: the software must be written in C++ to ensure compatibility with other Biorobotics Lab software, established software standards and file formats must be used and the financial constraints must be adhered to. Most of the software has been written in C++, including all modules interfacing with other software, only the visualizer federate uses web techologies. Used libraries, like boost for C++, are all well established libraries. The used data formats for logging or configuration files are csv and xml. Both formats are standard file formats and have been widely used for years. Because it was either self developed or used open source libraries, the entire software did not generate any financial costs and staid within the budget limits defined in the market analysis section (3.1). So in conclusion all the technical requirements have been met.

The simulation must be able to handle different agent types, each with the option of running a multitude of behaviors. This requirement has been met. The software imposes no restrictions on the type of agent any federate is using. Through the unified data exchange interface of the HLA standard, any type of agent data can be communicated within a simulation. This prototype also employs external behavior plugins for its agents, which are interchangeable. The next basic requirement was that many different scenarios must be usable. Through the use of simple text files in the common XML format, many kinds of scenarios can be created and distributed. This in combination with the fulfilled first requirement allows to run all scenarios applicable for the current projects of the Biorobotics Lab. Recorded and logged data is required to be saved in easy to exchange file formats, to facilitate the connection of different software projects within the research group. By using CSV or plain text files for these tasks, this goal was achieved. In order to ensure longevity of the simulation software and to allow its reuse for different research projects, a modular software structure was made a requirement. The separation of management and simulation participants enforced

by the HLA standard provides this modularity at the top level. Splitting different functionalities like data generation, visualization or logging into several federates provides modularization on the lower level as well. Adding frameworks for the individual federate's functions, like it was done for the visualizer, also improves reusability and modularity. It can be concluded that modularity has been achieved to a satisfactory degree.

Aside from the technical characteristics and the implemented features, usability is an important factor when evaluating a software. In the target specifications chapter (2.2) three groups of potential users were identified. For two out of the three groups minimal experience and understanding of computer science has to be expected, yet these user groups must still be able to use the simulation. This goal was achieved by allowing the interaction with the software by two means. The software modules participating in the simulation can be started from a terminal. The commands are short and include additional information about their usage and available options. If the visualizer federate is used, a web GUI provides this functionality through a button element. Both methods are intuitive and simple. The required knowledge on using simple webpage elements can be expected from all users. The configuration of individual federates is handled with simple text files and documented. This allows users without specific knowledge to adjust the simulation to their needs. All relevant information, required to program new federates or extend existing modules, are documented as well. Overall, the steps taken to increase usability cater to the needs of the target user groups and lower the required skills to use the FUSimulator to a minimum.

6.2 Performance Evaluation

In this section the performance of the software prototype will be analyzed. The focus will be on the performance of the RTI implementation since the currently present federates are not representative for future federates. All visualization includes rendering graphical elements through the rendering pipeline of a web browser. The performance of that pipeline depends on the individual browser implementation and is outside the scope of this work. To avoid the distorting influence of the browser, all performance analysis will be run in headless mode, without any visualization.

One of the targets specified for this prototype in the target specifications (2.3) was adequate scalability. A predictable bottleneck is the increase in messages sent with growing agent numbers. For testing this, a variable amount of agents and fixed parameters for hardware, scenario and frame count were used. Behavior plugin loading as well as the actual execution of any behavior was skipped so only the load created by RTI interactions could be measured. No frame rate restriction was used. With the current setup a

linear growth of cost was to be expected as for each additional agent update messages would have to sent to all subscribing federates.

The simulation was run on a single machine in order to avoid external factors like networking hardware to influence the measurement. The scenario was set to be an empty fish tank with the dimension of 500cm x 500cm. The most used scenario in the Biorobotics Lab is 100cm x 100cm, but in this experiment large quantities of fish will be simulated. To allow the proper application of the Couzin behavior model for all fish, more space was required. The used hardware was a Intel Core i5-3210M CPU with 4 cores of 2.50GHz and 8GB of RAM. Every simulation ran for exactly 1000 frames and with identical settings. The software allows for the simulation to be paused but all simulation executions were run without using that feature. A linear increase in execution time is expected with a growing number of agents. The degree of the linear increase depends on the number of federates subscribing to the fish agent data. Since one publishing and two subscribing federates will be run in this experiment, a linear increase of $y = 3x$ will be used as reference. All all runs were executed over several iterations and the results averaged.

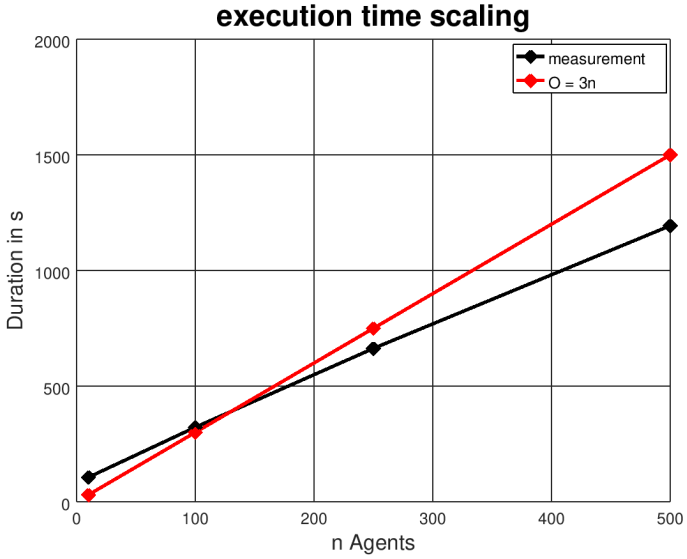

Figure 7: Runtime analysis for varying amounts of agents

As can be seen in figure 7, the expectations were met. The execution time grew in a linear fashion, but even staid below the expected values. This shows, that the communication between RTI and federates creates little overhead.

Since this software allows to be run on a single machine or distributed over

a network, both modes will be compared in their performance. Generally communication cost must be balanced with the gain from using multiple cpus for calculations. The experiment used a single master federate and three animal swarm federates which handled a varying amount of agents, equally spread among them. The additional computers running the agent federates contained Core i7 CPU with 8 cores of 3.3GHz and 16GB of memory. All other parameters did not change during test runs. The number of agents increased, but access to computational power remained static.

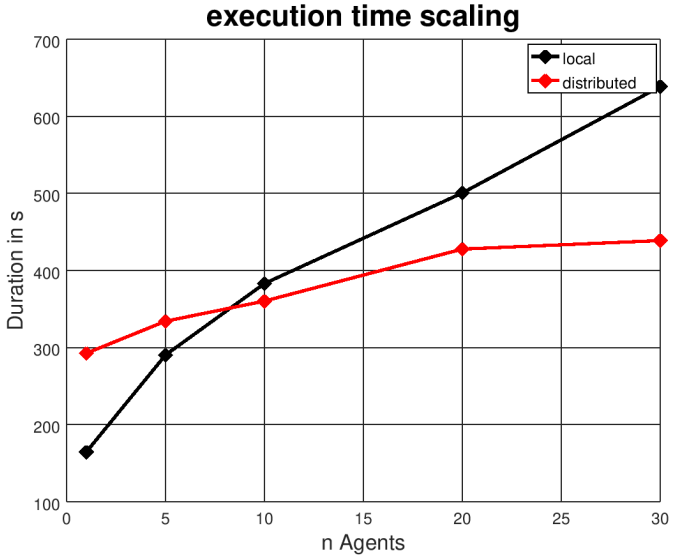

Figure 8: Runtime analysis for 3 agent federates

For a low number of agents per federate, the network overhead is clearly visible. As the overall agent count increases, the execution time of the locally run version increases in a linear fashion. The distributed runtime increases at a much smaller rate. The increase can be credited mostly to the used network hardware. As expected, an intersection between both graphs can be seen, marking the point of complexity at which a distributed simulation becomes advantageous. The exact location of this intersection depends on several factors like used hardware and the complexity of the agent models run. In this example the intersection appears early at around 7 agents. These results prove that the software provides the expected benefits and scalability distributed simulations.

6.3 Extendability Evaluation

Since the created software is a framework designed to allow other users to create their own custom simulations, it is reasonable to test how easily and efficiently custom projects can be programmed with it. In September 2018 professor Landgraf of the Freie Universität Berlin offered a software project as university class on the topic of ”collective intelligence for autonomous vehicles”. One of the given tasks was the implementation of a traffic simulation in which several autonomous cars dynamically share information about traffic impediments like accidents, traffic jams or blocked roads amongst each other. The individual cars were to be modelled as independent agents, including a basic driving behaviour and rules for communication with other cars. In addition a basic pathfinding was required which should allow cars to react to received information about obstacles by choosing a different route. The time frame for the project was two weeks.

The students were able to choose from three different simulator frameworks: Carla [47], SUMO [48] and FUSimulator. The FUSimulator was finally chosen due to its generic setup and lightweight implementation. The large degree of freedom for the implementation of specific features within a federate and the inclusion of various libraries was also a main selling point.

The students implemented their own federate which manages the autonomous car agents, including a custom car agent class. Pathfinding was provided by the addition of the Boost graph library while inter-agent communication was realized through the given HLA features. The scenario class was extended to hold road map data in the open street map format (OSM) and a custom XML parser was written to load it. For the visualization and data logging the existing APIs in the respective federates were used.

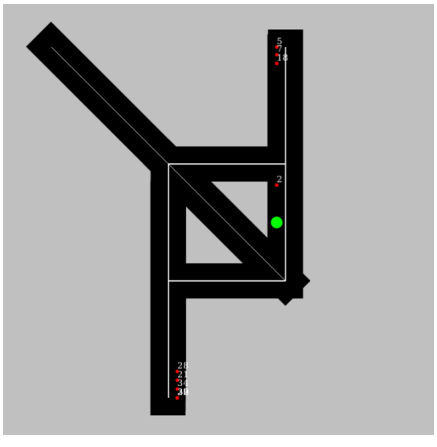

Figure 9: Road network setup 1 [49]

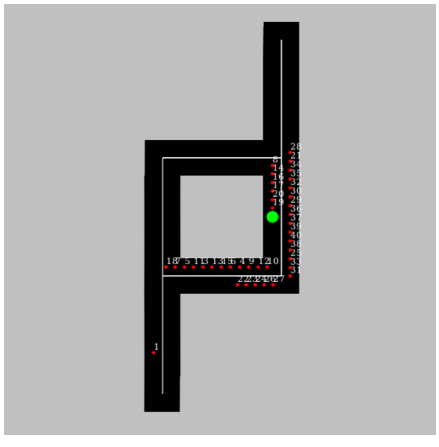

Figure 10: Road network setup 2 [49]

In general the students had very few problems working with the framework. Initially the structure of the HLA standard and the resulting simulator structure had to be explained thoroughly. Due to the complex concepts and terminology of the HLA design, this was required so the students were able to quickly identify the proper places in the code to implement their desired features. Two exemplary traffic flow scenarios worked on by the students are shown in figure 9 and 10.

The students were able to achieve their goal and created a basic working traffic simulation within the given time frame. According to interviews with the students the framework was easy to use after the basic HLA structure had been understood. The code documentation and the simulation wiki were named as adequate sources of information.

In conclusion the framework fulfilled its purpose. It allowed users with no prior experience with the HLA standard or the simulator to create a working simulation with a completely different topic in an adequate time frame. All required tools and functionalities were present and fulfilled their purpose.

7 Summary and Outlook

7.1 Summary

In this masters thesis a framework for distributed simulations was developed. The target specifications were established based on the requirements of the research group of the Biorobotics Lab of the Freie Universität of Berlin. After a market analysis of available software products and design schemes, the target specifications were translated into a concrete software design. Based on the HLA standard for distributed simulations, a prototype software including a working example for a distributed simulation of schools of fish was created. After the evaluation of the software concerning its included features, its performance in different tests as well as its extendability for other research projects, it can be said that it fulfills the target specifications.

With the software created in this thesis, the Biorobotics Lab of the Freie Universität Berlin received a tool to support it in conducting its experiments. The FUSimulator allows the pretraining of behavior used on robots interacting with animals as well as the visualization of generated data for further analysation. Finally it provides an option to simulate experiments and estimate their outcome before conducting them with real animals.

7.2 Outlook

Not all features defined by the HLA-standard are included in this prototype. A further extension of this software could include all of these, depending on the needs of future projects. Especially ownership management, data distribution management and more time management modes should be added as they are required features in many common use cases. The web GUI creation framework used in the visualizer could be extended and turned into a separate visualization library, usable by many different federates concerned with turning data into graphical output. A similar framework approach seems reasonable for the logging and replay functionalities. The third area of possible future improvements would be the specific simulation scenario itself. The currents needs of the Biorobotics Lab do not require more than simulations of fish in an empty tank. But with progressing research more complex experiments will be run. Using dynamically simulated food sources or agents representing predators are just a few examples which could be added easily with the current technology. Global influences like currents or water temperature changes would need to be implemented from scratch. Since a strong emphasis was put modularity during development, all extensions should be easy to add.

Bibliography

[1] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, 2015.

[2] I. García-Magariño and R. Lacuesta, "ABS-TrustSDN: An Agent-Based Simulator of Trust Strategies in Software-Defined Networks," *Security and Communication Networks*, vol. 2017, p. 9, 2017.

[3] J. Schlank, "Simulators - agent-based models." http://www.agent-based-models.com/blog/resources/simulators/. Accessed: 2018-09-30.

[4] D. Sislák, M. Rehák, M. Pechoucek, and D. Pavlícek, "Deployment of a-globe multi-agent platform," in *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006* (H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, eds.), AAMAS '06, pp. 1447–1448, ACM, May 2006.

[5] K. Shin, H. Nam, and T. Lee, "Communication modeling for a combat simulation in a network centric warfare environment," in *Porceedings of the 2013 Winter Simulation Conference* (R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. Kuhl, eds.), (Daejeon 157-032, Republic of Korea), Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology, 2013.

[6] I. Acronymics, "AgentBuilder Pro." http://www.agentbuilder.com/Documentation/Pro//. Accessed: 2018-07-25.

[7] F. Bellifemine, "Jade, a white paper," September 2003.

[8] L. Braubach and A. Pokahr, "The jadex project: Simulation," in *Multiagent Systems and Applications - Volume 1: Practice and Experience* (M. Ganzha and L. C. Jain, eds.), vol. 45 of *Intelligent Systems Reference Library*, pp. 107–128, Springer, 2013.

[9] J. M. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. W. N. van der Torre, "The BOID architecture: conflicts between beliefs, obligations, intentions and desires," in *Proceedings of the Fifth International Conference on Autonomous Agents, AGENTS 2001, Montreal, Canada, May 28 - June 1, 2001* (E. André, S. Sen, C. Frasson, and J. P. Müller, eds.), pp. 9–16, ACM, 2001.

[10] Distributed Systems and Information Systems Group at the University of Hamburg, *Jadex Dokumentation-3.0.0-RC1*, 2016.

[11] Distributed Systems and Information Systems Group at the University of Hamburg, "Jadex active components user guide."

https://download.actoron.com/docs/releases/latest/jadex-mkdocs/guides/ac/08%20Security/. Accessed: 2018-07-30.

[12] E. Noulard, J.-Y. Rousselot, and P. Siron, "CERTI, an Open Source RTI, why and how," *Spring Simulation Interoperability Workshop*, pp. 23–27, 01 2009.

[13] O. Obst and M. Rollmann, "Spark - a generic simulator for physical multi-agent simulations," in *Multiagent System Technologies* (G. Lindemann, J. Denzinger, I. J. Timm, and R. Unland, eds.), (Berlin, Heidelberg), pp. 243–257, Springer Berlin Heidelberg, 2004.

[14] D. Singh and L. Padgham, "OpenSim: A framework for integrating agent-based models and simulation components," in *Proceedings of the 21st European Conference on Artificial Intelligence* (T. Schaub, G. Friedrich, and B. O'Sullivan, eds.), vol. 263, pp. 837–842, IOS Press, 2014.

[15] S. Tisue and U. Wilensky, "Netlogo: Design and implementation of a multi-agent modeling environment," in *Proceedings of the Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence* (C. M. Macal, D. Sallach, and M. J. North, eds.), pp. 161–184, Argonne National Laboratory (ANL), Argonne, IL, jan 2004, updated 2013.

[16] U. Wilensky, "Netlogo web." http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Biology/BeeSmart%20Hive%20Finding.nlogo. Accessed: 2018-10-22.

[17] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The swarm simulation system: A toolkit for building multi-agent simulations," *Santa Fe Institute Working Paper*, vol. 96-06-042, 07 1996.

[18] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, "FLAME: simulating large populations of agents on parallel hardware architectures," in *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3* (W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, eds.), pp. 1633–1636, IFAAMAS, 2010.

[19] R. Allan, "Survey of agent based modelling and simulation tools." http://www.grids.ac.uk/Complex/ABMS/ABMS.html. Accessed: 2018-07-25.

[20] Foundation for intelligent physical agents, *FIPA Abstract Architecture Specification*, 01 2002.

[21] Foundation for intelligent physical agents, *FIPA KIF Content Language Specification*, 10 2001.

[22] Foundation for intelligent physical agents, *FIPA SL Content Language Specification*, 10 2001.

[23] DSMO, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*. IEEE -Institute of Electrical and Electronics Engineers, 2010.

[24] Pitch Technologies, "The hla tutorial, a practical guide for developing distributed simulations." http://pitchtechnologies.com/wp-content/uploads/2014/04/TheHLAtutorial.pdf, 4 2014. Accessed: 2018-10-22.

[25] M. R. Reid, "An Evaluation of the High Level Architecture (HLA) as a Framework for NASA Modeling and Simulation," in *25th NASA Software Engineering Workshop, Goddard Space Flight Center, Greenbelt Maryland, 2000-30-11*, pp. 1–9, 2000. Accessed: 2017-11-21.

[26] D. Teare, "Nasa lessons learned using high level architecture (hla) in engineering design analysis." https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120004025.pdf, 2012. Accessed: 2018-10-12.

[27] D. C. Miller, "SIMNET and Beyond: A History of the Development of Distributed Simulation." http://www.iitsec.org/-/media/sites/iitsec/link-attachments/iitsec-fellows/2015_fellowpaper_miller.ashx, 2015. Accessed: 2017-11-20.

[28] DSMO, *IEEE Standard for Distributed Interactive Simulation - Application Protocols*. IEEE -Institute of Electrical and Electronics Engineers, 2012.

[29] A. L. Wilson and R. M. Weatherly, "The aggregate level simulation protocol: an evolving system," in *Proceedings of the 26th conference on Winter simulation, WSC 1994, Lake Buena Vista, FL, USA, December 11-14, 1994* (D. A. Sadowski, A. F. Seila, M. S. Manivannan, and J. D. Tew, eds.), pp. 781–787, ACM, dec 1994.

[30] C. Boer, A. de Bruin, and A. Verbraeck, "A survey on distributed simulation in industry," *Journal of Simulation*, vol. 3, no. 1, pp. 3–16, 2009.

[31] J. S. Dahmann, R. Fujimoto, and R. M. Weatherly, "The department of defense high level architecture," in *Proceedings of the 29th conference on Winter simulation, WSC 1997, Atlanta, GA, USA, December 7-10,*

1997 (S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, eds.), pp. 142–149, ACM, 05 1997.

[32] R. El Abdouni Khayari, "Simulations- und Testumbegung der Bundeswehr (SuTBw)[Simulation and test environment of the German Armed Forces (SuTBw)]." https://www.unibw.de/itis/sut/sut_beschreibung/at_download/down1. Accessed: 2017-11-21.

[33] M. D. Petty and P. S. Windyga, "A high level architecture-based medical simulation system," *SIMULATION*, vol. Vol 73, Issue 5, pp. p281–287, 1999.

[34] S. Straßburger, *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*. dissertation, Otto-von-Guericke-Universität Magdeburg, 2000.

[35] H. Oğuztüzün and O. Topçu, *Guide to Distributed Simulation with HLA*. Simulation Foundations, Methods and Applications, Springer, 11 2017.

[36] DSMO, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification*. IEEE -Institute of Electrical and Electronics Engineers, 2010.

[37] DSMO, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification*. IEEE -Institute of Electrical and Electronics Engineers, 2010.

[38] Z. Tu, G. Zacharewicz, and D. Chen, "Developing a web-enabled HLA federate based on portico RTI," in *Winter Simulation Conference 2011, WSC'11, Phoenix, AZ, USA, December 11-14, 2011* (S. Jain, R. R. C. Jr., J. Himmelspach, K. P. White, and M. C. Fu, eds.), pp. 2294–2306, WSC, 2011.

[39] A. H. Buss and L. Jackson, "Distributed simulation modeling: a comparison of hla, corba, and RMI," in *Proceedings of the 30th conference on Winter simulation, WSC 1998, Washington DC, USA, December 13-16, 1998* (D. J. Medeiros, E. F. Watson, J. S. C. II, and M. S. Manivannan, eds.), pp. 819–826, WSC, 1998.

[40] P&P Software GmbH, *EODiSP Developer Manuals*, 2006. https://www.pnp-software.com/eodisp/documentation/developerManuals/hlaArchitecture.html.

[41] S. I. S. O. (SISO), *Standard for Military Scenario Definition Language (MSDL),*. Simulation Interoperability Standards Organization (SISO), 10 2008.

[42] B. Watrous, L. Granowetter, and D. Wood, "HLA Federation Performance: What Really Matters?." https://www.sisostds.org/DesktopModules/Bring2mind/DMX/API/Entries/Download?Command=Core_Download&EntryId=27211&PortalId=0&TabId=105. Accessed: 2019-01-29.

[43] DSMO, *RTI 1.3-Next Generation Programmer's Guide Version 3.2*. Department of Defense Defense Modeling and Simulation Office, 2002.

[44] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin, "Effective leadership and decision-making in animal groups on the move," *Nature*, vol. 433, pp. 513 EP –, Feb 2005.

[45] Dorier, Matthieu, "Toward Efficient Coupling of Large-Scale Simulations and Visualization." https://www.researchgate.net/publication/265677871_Toward_Efficient_Coupling_of_Large-Scale_Simulations_and_Visualization, 03 2019.

[46] R. Minson and G. K. Theodoropoulos, "Distributing repast agent-based simulations with HLA," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 10, pp. 1225–1256, 2008.

[47] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: an open urban driving simulator," in *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, vol. 78 of *Proceedings of Machine Learning Research*, pp. 1–16, PMLR, 11 2017.

[48] M. Behrisch, L. Bieker-Walz, J. Erdmann, and D. Krajzewicz, "SUMO – Simulation of Urban MObility: An Overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation* (A. Omerovic, D. A. Simoni, and G. Bobashev, eds.), vol. 2011, ThinkMind, 10 2011.

[49] F. Boenisch, B. Rathjen, and P. Winterstein, "SWP Kollektive Intelligenz für autonome Fahrzeuge (SS18) [SWP Collective intteligence for autonomous vehicles (SS18)]," 2018. Accessed: 2019-01-10.

List of Figures

A Appendix

The code for the FUSimulator can be found at:

https://git.imp.fu-berlin.de/bioroboticslab/robofish/simulator