Bachelor's thesis

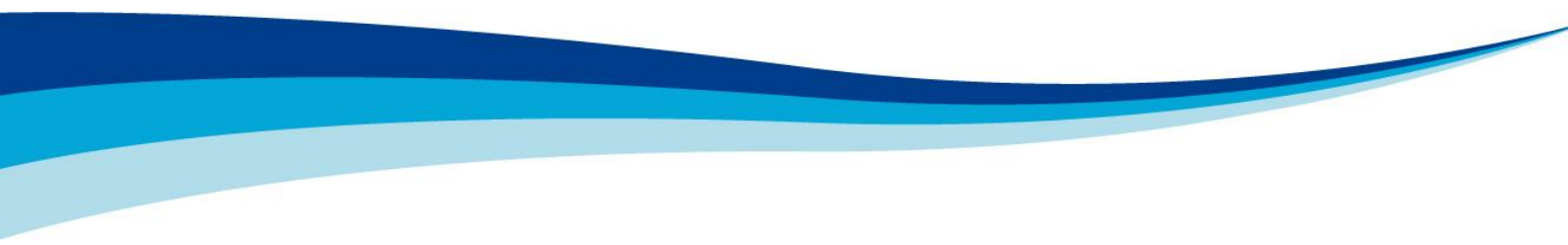
Information Technology

Internet Technology

2014

Ezio Melotti

THE NETWORKING SUB-SYSTEM OF THE VIRTUAL EUROPEAN MARS ANALOG STATION



TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Internet Technology

2014-10-24 | 63 pages

Patric Granholm

Ezio Melotti

THE NETWORKING SUB-SYSTEM OF THE VIRTUAL EUROPEAN MARS ANALOG STATION

In the latest years, Mars colonization has been a topic of interest and active research for several organizations such as the Mars Society. The Italian branch of the Mars Society started working on a virtual reality simulation of Mars (called V-ERAS) that can be used to experience life on Mars, train astronauts, and for several other purposes.

The goal of this thesis is to design a networking sub-system that can be used to connect different devices and stations used for the simulation. The design needs to consider several different aspects, including the architecture of the system, the software and frameworks used, and how these devices and stations communicate with each other.

In order to determine the best solution, a number of different approaches have been identified, and the most promising ones have been tested with simplified scenarios to verify that they can indeed be implemented within the system.

The tests revealed that the Tango framework was the most effective solution, since it had all the required features and demonstrated to be stable and reliable. The MORSE framework also proved to be a good candidate, but it still had issues that need to be resolved before it can be considered again.

The final design described in the thesis will be adopted by the V-ERAS project and tested with a complete setup. If necessary, the design will be adapted to correct any problems that might arise. If the project proves to be successful, other organizations will also be able to benefit from it.

KEYWORDS:

networking, Mars, virtual reality, simulation, Tango, Blender, MORSE, sockets, ERAS, V-ERAS

CONTENTS

FIGURES

LIST OF TERMS AND ABBREVIATIONS

Avatar	A virtual 3D representation of a human, used within the V-ERAS simulation to represent a crew member.
Blender	Blender is an open-source 3D animation suite developed by the Blender Foundation (Blender Foundation, 2014). In addition to the 3D modeling, Blender also includes a game engine and a physics engine that can be used to animate realistically the avatar. Blender also supports Python scripting, allowing the interaction with other Python libraries and frameworks, such as PyTango.
Blender standalone runtime	An executable file created by exporting a Blender scene. Unlike the embedded runtime — that runs the simulation within Blender — the standalone runtime does not include the Blender interface and does not allow the editing of the scene, but only presents a (possibly full-screen) window with the simulation.
callback	A callback is a function that is passed to another function that is then expected to call the callback at a later moment. In case of event-driven programming, a callback might be associated with a specific event, and whenever the event is triggered the callback will be executed.
C3	The Command, Control, and Communication system being developed by the Italian Mars Society. See the Background information section for more information.
CORBA	Common Object Request Broker Architecture, a standard defined by the Object Management Group (OMG). It is designed to facilitate the communication of systems that are deployed on diverse platforms.
ERAS	The European maRs Analog Station for advanced technologies integration project, developed by the IMS.
ERAS station	The term “ERAS station” refers to the Mars analog station (i.e. the habitat) that the Italian Mars Society is planning to build. Not to be confused with V-ERAS station.
EVA	An Extra-Vehicular Activity performed by an astronaut outside a spacecraft or habitat. This includes both spacewalks and activities on the surface of another celestial body, such as Mars or the Moon.
FMARS	The Flashline Mars Arctic Research Station, a Mars exploration analog research facility established in 2000 in the Canadian Arctic by the Mars Society.
HLA	HLA (High-Level Architecture) is a specification for software architectures that defines the management and deployment

	of a global simulation made of distributed simulators. Each simulator (called a federate) is connected to other simulators through the Run-Time Infrastructure (RTI) — a fundamental component of HLA that provides a set of software services that are necessary to support federates to coordinate their operations and data exchange during a runtime execution.
HUD	HUD stands for Head-Up Display, and it is a term used to indicate any display used to show information to the user without requiring him to look away from his usual viewpoint. On a space suit, information might be displayed on the inside of the visor.
IMS	Italian Mars Society, the Italian chapter of the Mars Society.
Jive	Jive is a tool used with Tango to browse the database of the devices and to test them. Every Tango server needs to be registered on Jive before it can be used.
JSON	JavaScript Object Notation, a standardized and human-readable serialization format.
Kinect	The Kinect is a motion sensing device developed by Microsoft that uses range cameras to provide full-body 3D motion capture. See the Body-tracking sub-system section for more information.
The Mars Society	The Mars Society is a non-profit organization. Its purpose is to further the exploration and settlement of Mars.
MARS	The Mars Analog Research Station program, developed by The Mars Society. Its goal is to provide prototype habitats that can be used by scientists and engineers to simulate living on Mars.
MDRS	The Mars Desert Research Station, a Mars exploration analog research facility established in 2000 in Utah by the Mars Society.
Middleware	A middleware is a software component that allows or facilitates the interaction and communication between other components that would otherwise be incompatible.
Mock	In testing, the term “mock” refers to a software component that replaces and imitates the behavior of a different component (such as a function, a class, or even a server or a device). Mocks are particularly useful when they are used to replace parts of a systems that would be difficult to test directly: for example, a component that reads data from a specific hardware device can be replaced by a mock that reads recorded data.
MORSE	MORSE is a generic simulator for academic robotics. It focuses on realistic 3D simulation of small to large

	environments, indoor or outdoor, with one to tenths of autonomous robots (Anon., 2014a). MORSE is based on Blender, and can be configured and controlled with Python.
Motivity	The Motivity is an omnidirectional treadmill designed and developed specifically for the V-ERAS project. See the Body-tracking sub-system section for more information.
multiprocessing	A Python module available in the standard library that can be used to spawn and interact with multiple processes. It also offers some high-level facilities such as pipes and queues that can be used to exchange data between the processes.
Oculus Rift	The Oculus Rift is a virtual reality headset, being developed by Oculus VR. It is capable of producing stereoscopic 3D images and also includes sensors used to track the head position so that the images can be updated accordingly to the head movements. At the time of writing, the final version of the headset had not been released yet, but prototypes are available.
patch	The term “patch” can indicate both a type of file that can be used to modify part of a source code, or the act of applying a patch, or, more generically, the act of altering or fixing a program by changing its source code.
POE	Post Occupancy Evaluation is “the process of evaluating buildings in a systematic and rigorous manner after they have been built and occupied for some time” (Preiser, et al., 1988).
pymorse	A Python library used to interact with MORSE. pymorse uses sockets to establish a connection with a running MORSE simulation and to communicate with it, and can be used to update and retrieve the position, rotation, or other data about the objects in the scene.
PyTango	A Python library used to interact with Tango. PyTango can be used to access the Tango bus and communicate with all the Tango servers that are connected to it. It also provides all the basic building blocks that are used to develop new Tango servers.
Raspberry Pi	The Raspberry Pi is a small (about the size of a credit card) single-board computer (Anon., 2014b). It features a 700 MHz CPU, 256 MB of memory, USB and HDMI ports, and it is able to run Linux distributions. Different extensions and peripherals (such as monitors, keyboards, microphones, etc.) can also be connected to it. The low size, energy consumption, and cost make it particularly suitable to be used in a space suit.
Regression test	A test whose objective is to ensure that an error that has been previously fixed is not reintroduced.

SCADA	SCADA stands for Supervisory Control And Data Acquisition. A SCADA system uses coded signals over communication channels to control remote equipments and acquire data from them.
Scene	In Blender, the term “scene” is used to refer to a specific environment and the set of 3D objects within it.
Smoke test	A smoke test is a very basic form of test that only verifies the most basic behavior of a component. For example, a smoke test can be used to verify if a component can be started without errors, without checking for anything else.
stdout	stdout stands for “standard output” and is one of the three standard streams (together with stdin and stderr) that are automatically associated with a process. Everything that is printed by a process is written on stdout. By default, stdout is connected to the text terminal used to start the process and all the text written on stdout is printed on the terminal, but the output can also be redirected or read by another process.
Sub-process	A process spawned by another process (usually called the parent process) is called sub-process (or child process). The parent process can control, interact, and terminate the child process.
subprocess	A Python module available in the standard library that can be used to create sub-processes and interact with them.
Tango bus	The Tango bus is a software bus used by Tango as a communication channel between all the Tango servers. Every information written on the Tango bus can be accessed and read by all the other servers.
Tango server	Tango servers are one of the most important components of Tango. A Tango server is responsible to exchange data between a device (e.g., the Oculus Rift or the Kinect) and the Tango bus. They can also be used to export data originating from a software — rather than hardware — component on the Tango bus.
Test suite	A collection of test cases used to verify to correct functioning of a software. A test suite can include different kind of tests, including unit tests and integration tests.
Unit tests	A test used to verify the correct functioning of a single unit of source code, such as a function or a class. Several unit tests can be collected in a test case.
Integration tests	An integration test is responsible to verify the functioning of a group of units that have already been tested independently and how they interact with each other.

V-ERAS	The Virtual ERAS project, developed by the IMS. See The V-ERAS project chapter for more information.
V-ERAS station	The term “V-ERAS station” refers to the combination of a computer, a Motivity omnidirectional treadmill, an Oculus Rift, a Kinect, a Raspberry Pi, and possibly additional devices. The station can be used by a single crew member, and three stations are planned to be used during the simulation. Not to be confused with ERAS station.
VR	VR stands for Virtual Reality, a computer-simulated environment that can either be real (e.g. a specific location on Earth or on another planet) or imaginary. The user can interact with the environment using one or more senses, depending on the level of immersion provided by the simulation.
ZeroMQ	An open source, multi-platform, multi-language, high-performance asynchronous messaging library (iMatix Corporation and Contributors, 2014).

1 INTRODUCTION

This chapter introduces the objectives of this work and the necessary background required to understand its context. It also describes the research method used to reach the results and the structure of the thesis.

1.1 Background information

The work described in this thesis is part of the ERAS and V-ERAS projects, which in turn are part of the MARS program.

The Mars Analog Research Station (MARS) program is an international effort spearheaded by The Mars Society (The Mars Society, 2014a), and its main goal is to provide habitat prototypes on Earth similar to the ones that might be used on Mars. These habitats are used to test supply requirements, mission hardware, and the ability of crew members to work together under Mars-like settings (The Mars Society, 2014b).

There are currently two habitats:

- the Flashline Mars Arctic Research Station (FMARS), located on Devon Island in the Canadian Arctic (The Mars Society, 2014c);
- the Mars Desert Research Station (MDRS), located near the southern Utah town of Hanksville (The Mars Society, 2014d);

The European maRs Analog Station for advanced technologies integration (ERAS) project (The Italian Mars Society, 2014b) is an extension of the MARS program and it is being developed by the Italian Mars Society (IMS) (The Italian Mars Society, 2014a). The goal of the project is to address the “Five Show-stoppers for Mars” (Cohen, 2011) identified by the scientific community as:

- hypogravity;
- radiation;
- need for regenerative and bioregenerative life support;
- martian dust;
- planetary protection (forward- and back-contamination);

The Virtual ERAS (V-ERAS) project is part of the ERAS project and its goal is to provide a simulated — rather than real — habitat that can be accessed through the use of virtual reality.

In addition, both ERAS and V-ERAS will take advantage of the C3 (Command, Control, and Communication) system that is being developed by the Italian Mars Society. The C3 system goals are to:

- Monitor and control the environment and subsystems of the planetary habitat.
- Monitor and maintain crew health and safety.
- Communicate with mission support, robots, and EVA crew members.
- Support data processing related to the mission objectives.
- Host the core part of the crew operations planning and scheduling support system.

1.2 Purpose of the thesis

The goal of this project is to develop the networking sub-system of the V-ERAS project. The networking sub-system has three main tasks:

- To enable network communication among the PC and the hardware devices used within a single V-ERAS station.
- To enable network communication among the V-ERAS stations used during the simulation.
- To ensure that the simulation is coherent among all the instances.

Several crew members must be able to simultaneously access the simulation from different V-ERAS stations (located in the same room), see each other, and

interact. It is also important that the environment, the positions of the avatars, and all the other elements of the simulation are updated in real time, and that they are always consistent and do not diverge over time.

1.3 Research method

In order to determine the best approach to implement the networking sub-system, the following steps have been followed:

1. Determining the goals and main issues that might arise.
2. Enumerating all the sensible approaches.
3. Filtering and determining the approaches that are likely to work best.
4. Verifying that it is possible to adopt them and test how well they work.
5. Determining and using the approach that proved to work best.

Most of the available and existing approaches are well known. However, given the magnitude and complexity of the project, considerable research is still necessary. The V-ERAS project uses several different frameworks and software, and some of them already provide networking components that can be used instead of the more traditional and low-level approaches. These components, nevertheless, have different features and performances, and their interactions might cause unforeseen problems.

In addition, for some of these, there is very little documentation available so testing and experimenting is often the only way to determine the feasibility of a possible solution. The integration of all these technologies and frameworks is also something that has not been implemented before, and the project itself is pushing the limit of current technologies.

1.4 Structure of the thesis

This thesis is organized in five chapters:

1. Introduction: provides a general overview and background information about the thesis.
2. The V-ERAS project: describes more in detail the V-ERAS project.
3. The networking sub-system: describes the possible designs of the sub-system, its goals, and the main issues and possible solutions that can be adopted, including their advantages and disadvantages.
4. Implementations testing: describes in detail how the different implementations have been tested and how they performed.
5. Conclusion: describes the final design of the networking sub-system, based on the results collected during the tests.

2 THE V-ERAS PROJECT

Virtual ERAS (V-ERAS) is the latest project of the Italian Mars Society. Launched in early 2014, it is meant to provide an immersive virtual reality (VR) simulation of an ERAS station.

The project has several different objectives. The two main objectives are to provide a relatively cheap and effective way to test the habitat design and the technologies involved, and train a crew of scientists and engineers.

The project also comprises several sub-systems that interact together. While this work focuses mostly on the networking sub-system, it is also important to understand where it fits in the project and how it interacts with the other sub-systems.

This chapter provides more information about V-ERAS.

2.1 Objectives

There are six macro-objectives identified by the IMS for the V-ERAS project:

1. To validate and improve the design of ERAS and other analogue or real mission habitats.
2. To provide effective planetary exploration crew training.
3. To provide an effective test bed for key-enabling technologies.
4. To provide an environment able to maximize scientific productivity of researchers.
5. To ensure effective transfer of results to the scientific community.
6. To ensure effective outreach.

2.1.1 Design validation and improvement

One of the most important objectives of V-ERAS is to validate the design of the ERAS habitat. This includes the number, placement, and dimensions of the different sections of the habitat and any furniture or hardware (such as tables, chairs, working desks, monitors, doors, greenhouse, toilets, etc.).

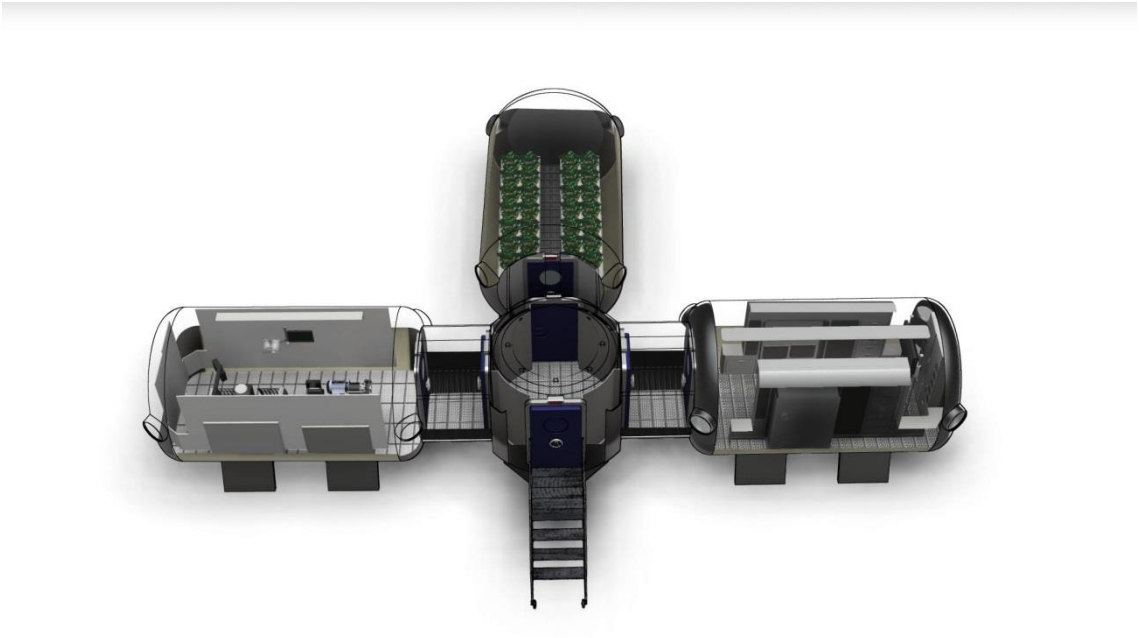


Figure 1. A prototype of the ERAS habitat.

During the simulation the crew members will be able to determine whether the design of the habitat is ideal, or if there are any problems that can be addressed.

The virtual environment will provide an easy way to edit and experiment with different design, and could also be used by similar projects to test the design of their habitats.

2.1.2 Crew training

Another major objective of the project is to train crew members.

All the crew members need to be familiar with the station and with the location of any hardware that might be needed. In addition, they need to learn and practise different procedures,such as compression and decompression while entering or leaving the habitat.

During the simulation, they will be able to practice EVAs (Extra Vehicular Activities), learn how to use the space suit, and even interact with rovers.

2.1.3 Test key-enabling technologies

There are several technologies used and designed specifically for the project. The simulation will allow to test most of the software that will be used on the real station, including the C3 (Command, Control, and Communication) system.

In addition, other type of technologies, such as man-machine interaction and device interfaces, can also be tested.

2.1.4 Maximizing scientific productivity

In order to run the station effectively, a series of procedures and protocols need to be defined and followed. These include communication between the engineering support crew and mission crew, before, during, and after the simulation.

A common and effective research platform involving different potential users also needs to be defined and tested.

2.1.5 Transfer of results to the scientific community

Another important objective is to collect and share with the scientific community the results and lessons learned during the simulation. This includes mission reports, Post Occupancy Evaluations (POE), key design information, and articles on scientific journals.

2.1.6 Ensure effective outreach

Finally, V-ERAS will be used to increase awareness about the work being done to colonize Mars, and educational events that include both observation and participations to the simulation will also be organized.

2.2 Architecture

The simulation requires specific hardware and software. These include both preexisting and custom-made hardware and software.

Each station comprises:

- a PC;
- a Motivity omnidirectional treadmill;
- an Oculus Rift VR headset;
- a Kinect sensor;
- a Raspberry Pi with a mounted E-Health sensors platform;

Figure 2 shows a rendition of four V-ERAS stations:

Figure 2. A representation of four V-ERAS stations.

In addition, all the stations used during the simulation are connected through a LAN network.

The software sub-systems include:

- 3D simulation;
- VR headset integration using the Oculus Rift;
- full body and hand gesture tracking using the Kinect and the Motivity;
- crew health monitoring using the Raspberry Pi;
- the networking sub-system;

2.2.1 3D simulation sub-system

The 3D simulation sub-system is at the core of the VR simulation. It is based on the open-source 3D graphics and animation software Blender.

This sub-system is responsible for creating the 3D environment. This includes all the models (the habitat, avatars, rovers, objects, etc.), the animations and interactions between the actors (avatars and rovers), and the physics simulation (handled by the Blender physics engine).

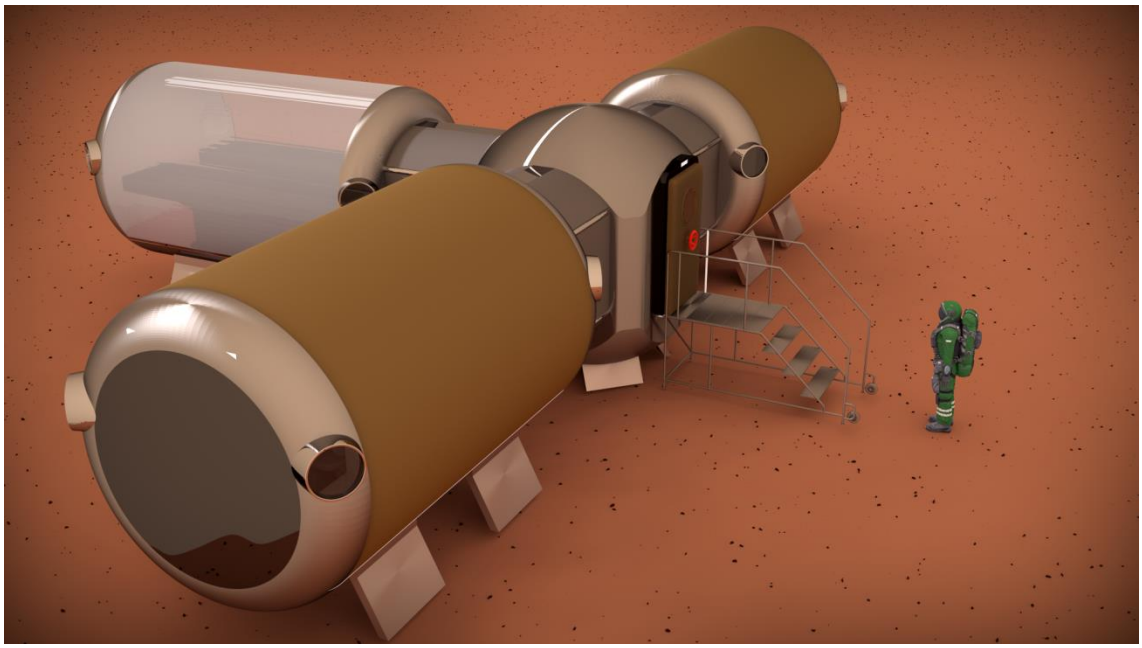


Figure 3. The habitat and an astronaut in the simulation.

Tests are also being performed using the MORSE simulator but its adoption has not been confirmed yet. MORSE already provides several of the features required by V-ERAS, including basic models of avatars and rovers, and a

convenient API that can be used to control them. Being based on Blender, it is also easy to extend MORSE with already existing models.

2.2.2 VR headset sub-system

The 3D simulation sub-system is connected to an Oculus Rift headset that is used to display an immersive 3D virtual reality environment to the user. The Oculus Rift also uses sensors to determine the head position and sends it to 3D simulation sub-system, in order to have the head camera within the simulation following the head movements of the user in real-time.

2.2.3 Body-tracking sub-system

The body-tracking sub-system uses two different hardware devices — the Kinect and the Motivity — to track the position and movements of each crew member. This includes both full-body movements (walking, running, turning, crouching, etc.) and hand gestures (waving, grabbing, pushing, etc.).

The Kinect is a motion-sensing device that uses range cameras to provide full-body 3D motion capture. The data collected by the Kinect are processed and sent to the 3D simulation sub-system that is then responsible to reproduce the same positions and movements done by the crew member within the simulation.

The Motivity is an omnidirectional treadmill designed and developed specifically for the V-ERAS project. The Motivity allows crew members to move freely in every direction while standing in the same place, thus avoiding the need of spacious rooms and facilitating the body tracking. The Motivity is a passive component — it does not communicate with other sub-systems, and its only role is to physically support the user.

2.2.4 Health monitoring sub-system

The goal of the health monitoring sub-system is to collect and analyze data relative to the health status of the crew member through a number of different

sensors. These include electrocardiogram, body temperature, blood pressure, galvanic skin response, airflow, oxygen in blood and possibly other sensors. The sensors are connected to the E-Health sensor platform (Libelium Comunicaciones Distribuidas S.L., 2014) that is mounted on a Raspberry Pi but they can also be integrated in a space suit that could be used during the simulation.

The data are then used to determine the conditions of the user, and can also be used to predict the trend and warn beforehand about potentially dangerous situations. This information can also be displayed to the users using a head-up display (HUD) or a virtual screen within the simulation, and also outside the simulation on a regular screen.

2.2.5 Networking sub-system

The networking sub-system is based on the Tango framework, it enables the communication among the devices and among the V-ERAS stations, and ensures that all the user see a consistent simulation.

Tango is an object-oriented distributed control system based on CORBA and ZeroMQ the can be used to develop SCADA (Supervisory Control And Data Acquisition) systems.

In the V-ERAS project, the main role of Tango is to handle the interaction with the different hardware devices used, but it is also being considered for the communication among the V-ERAS stations.

The networking sub-system, and in particular the communication among the stations is the focus of this work, and its functioning is described in detail in the following chapter.

3 THE NETWORKING SUB-SYSTEM

The networking sub-system covers two distinct — but related — areas:

1. Communication among the different stations.
2. Communication among the different hardware devices within a single station.

While the second area is already partly covered by the use of Tango, the first area is open to different possible solutions. In addition, the architecture and interaction of both, and the technologies used need to be chosen in a way that guarantees optimal interoperability.

Another main goal of the networking sub-system is to ensure that the simulation is coherent among all the instances. Since many factors are involved during networking communication, it is difficult to guarantee that all the stations receive the same network packets at the same time — some packets might get lost or they might be received at different times or in a different order. Even if most of these problems are minor, they might add up while running the simulation for extended periods of time, causing inconsistencies that might even compromise the whole simulation.

Finally, the following factors also need to be taken into account to determine the best solution:

- compatibility and interoperability with the other technologies used;
- features provided;
- latency;
- stability.

This chapter starts by analyzing the possible architectures and implementations and further discusses which solutions can be employed to maintain coherence within the simulations.

3.1 Architecture

Different architectures can be used to enable communication among several machines, but the two most common ones are:

1. server-client;
2. peer-to-peer;

3.1.1 Server-client

The advantages of a server-client architecture are:

- Ease of maintaining coherence (the servers tell all the clients what to do).
- Ease of solving conflicts (the server takes the decisions in case of conflict).

However, there are disadvantages as well:

- One of the machines has to act as a server and adding an additional computer will result in additional costs and network overhead. Re-using an existing machine will avoid this, but it will cause overhead for that machine.
- Additional network overhead and possibly lag.
- Different software required for the server and the clients.

The following figures show two possible ways that can be used to implement a server-client architecture.

1. A classical server-client architecture using an additional machine as a server.

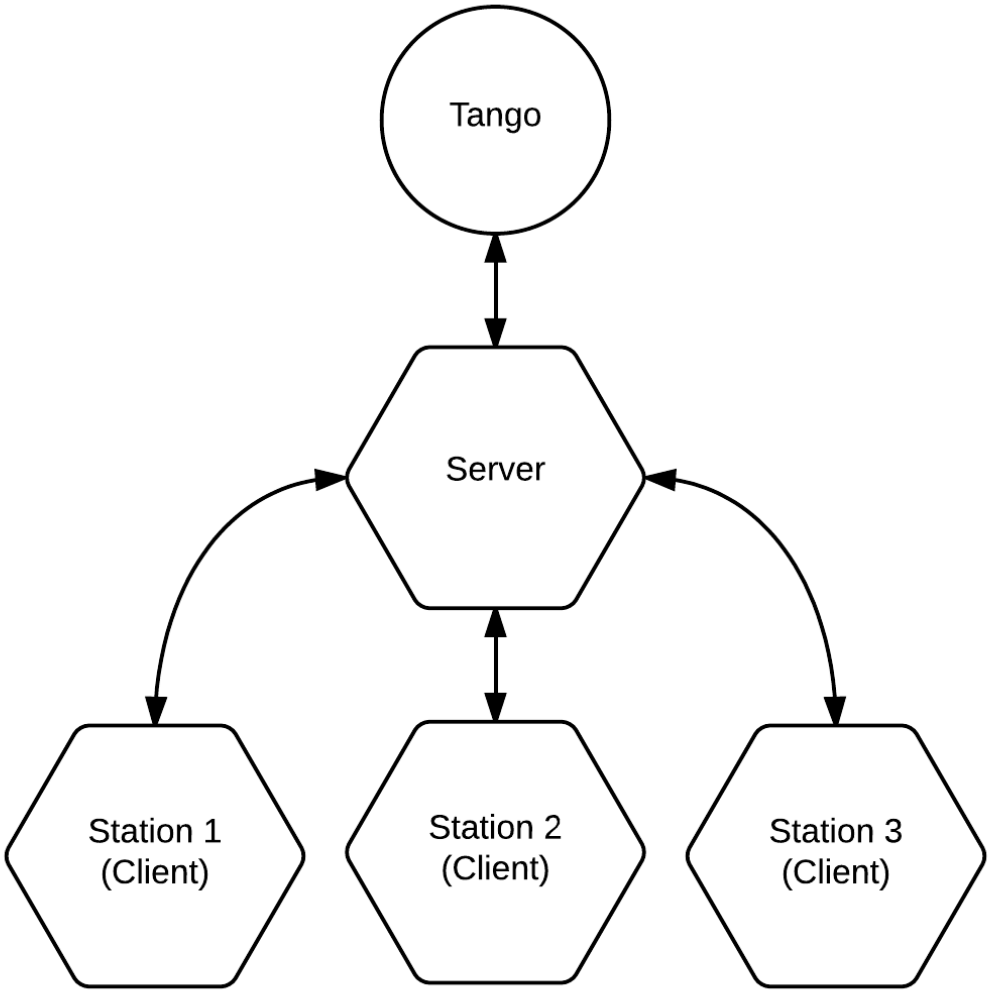


Figure 4. A server-client architecture using a physical server

2. An architecture that re-uses one of the machines as a server.

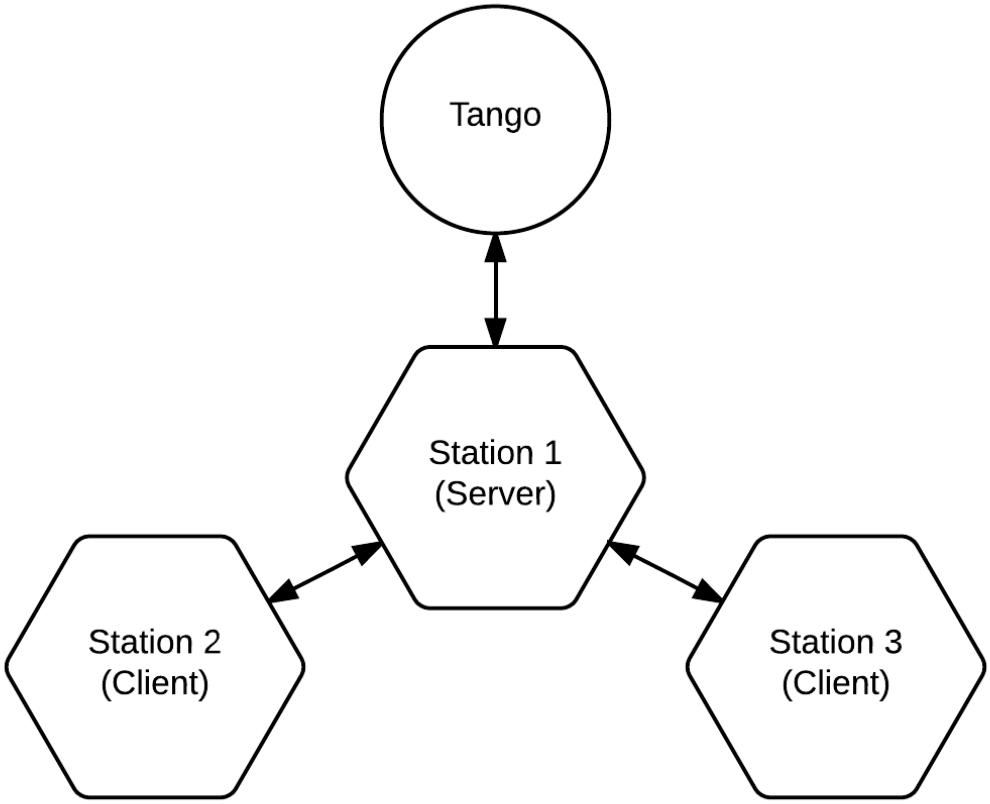


Figure 5. A server-client architecture with one of the machines acting as a server

In both cases Tango only interacts with the server, and the server redirects the messages to the clients. Figure 5 also helps understand how the additional network overhead is caused. With a physical server, both communications between Tango and the clients and from client to client have to go through at least two nodes (one of which is the server). When one of the station is re-used as a server (this only happens for the client stations, as mentioned above), it creates additional overhead on the machine.

3.1.2 Peer-to-peer

The advantages of a peer-to-peer network are:

- less network overhead, less lag;
- same software and configuration running on all the machines;

The disadvantages are:

- more difficulties to solve conflicts (the peers have to reach the same decision independently or have to communicate before proceeding);

Depending on how the communication happen, maintaining coherence might be more or less difficult. On one hand, divergence among the simulation might seem more likely without a central server; on the other hand, a simple architecture may reduce lag and result in higher coherence.

Figure 6 shows how three different stations can communicate among them directly. In addition, the Tango bus connected to all the three stations is depicted at the center of the figure.

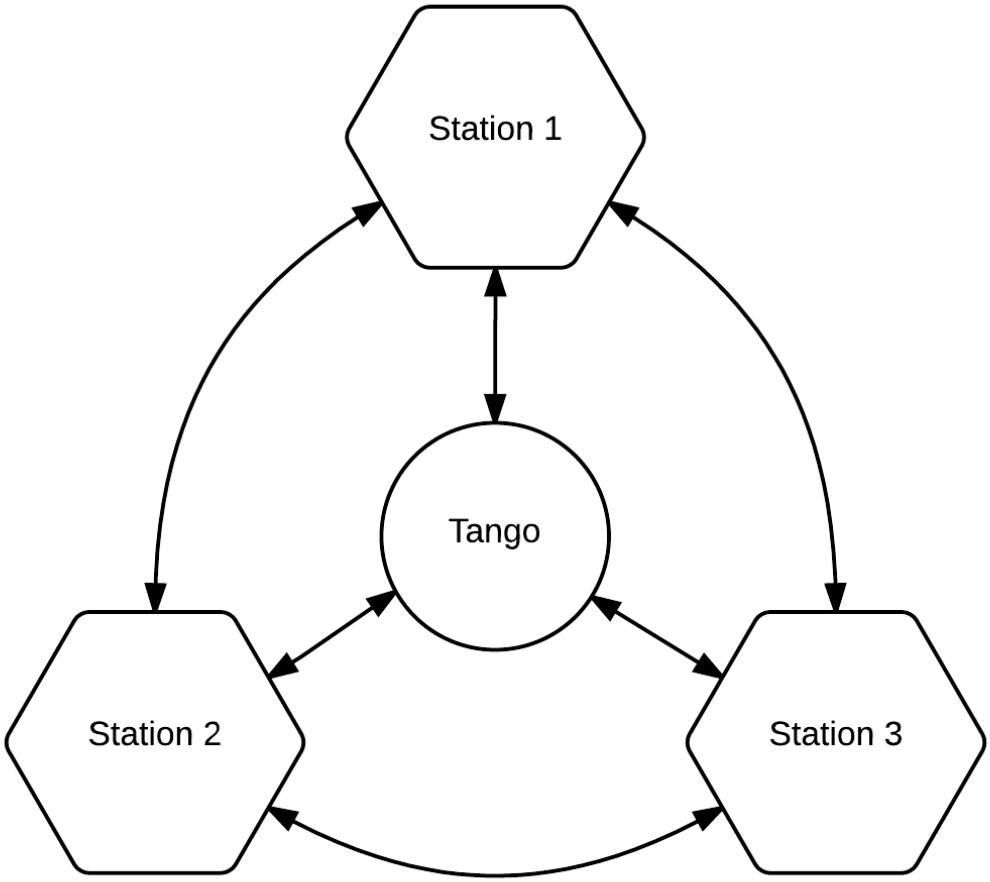


Figure 6. A peer-to-peer architecture

It should be noted that while this might resemble a server-client architecture, the Tango bus is not an actual server and the data that are circulating on the Tango bus are available to all the peers at the same time. This also means that Tango can also be used to handle peer-to-peer connections among the stations, thus avoiding the need of additional direct connections.

It is clear from Figure 6 that any station can communicate directly with any other station and Tango.

3.2 Implementations

The possible implementations being considered are:

1. sockets;
2. Tango;
    a. using polling;
    b. using events;
3. Blender plugins;
4. MORSE multi-node simulation;

3.2.1 Sockets

Using sockets is the most basic approach and it is widely used in a number of software. This is, however, a low-level solution, and requires the reimplementation of several components, such as a protocol used by the machines to understand each other.

Sockets can be used to exchange data between the machines directly from the Blender instances. However, Blender still needs to access the data on the Tango bus.

If a server-client architecture is used, the server could be the only machine reading data from Tango. The server will then process the data and send instructions to the other clients.

Figure 7 shows three V-ERAS stations. Every station includes a PC and several hardware devices (Oculus Rift, Kinect, and Raspberry Pi) that share their data on the Tango bus. Here the Blender instance of the V-ERAS station 2 is acting as a server and using sockets (indicated with dashed lines) to send data to the other two Blender instances. Also note how this is the only instance that reads data from the Tango bus.

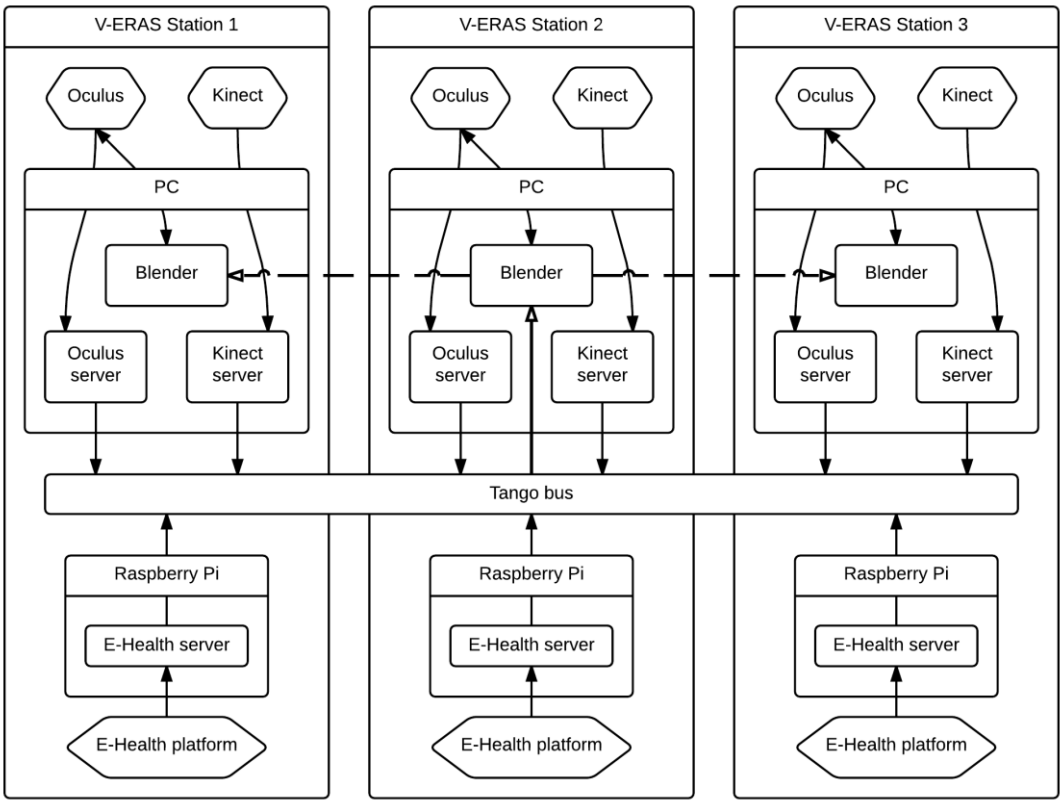


Figure 7. Implementation diagram of a server-client architecture with the Blender instance the PC of the V-ERAS station 2 acting as a server. The Blender instance of station 2 reads data from the Tango bus and sends instructions to the other instances via sockets.

This approach can also be implemented with a separate server machine that reads data from the Tango bus and then uses sockets to redirect the data to all the Blender instances.

If, instead, a peer-to-peer architecture is used, a hybrid approach can be considered. In this hybrid approach, all the peers read data from the Tango bus and sockets are used to ensure the consistency of the simulation by having the peer exchanging information about the objects in the simulation and verifying their correctness. This approach is shown in following Figure 8:

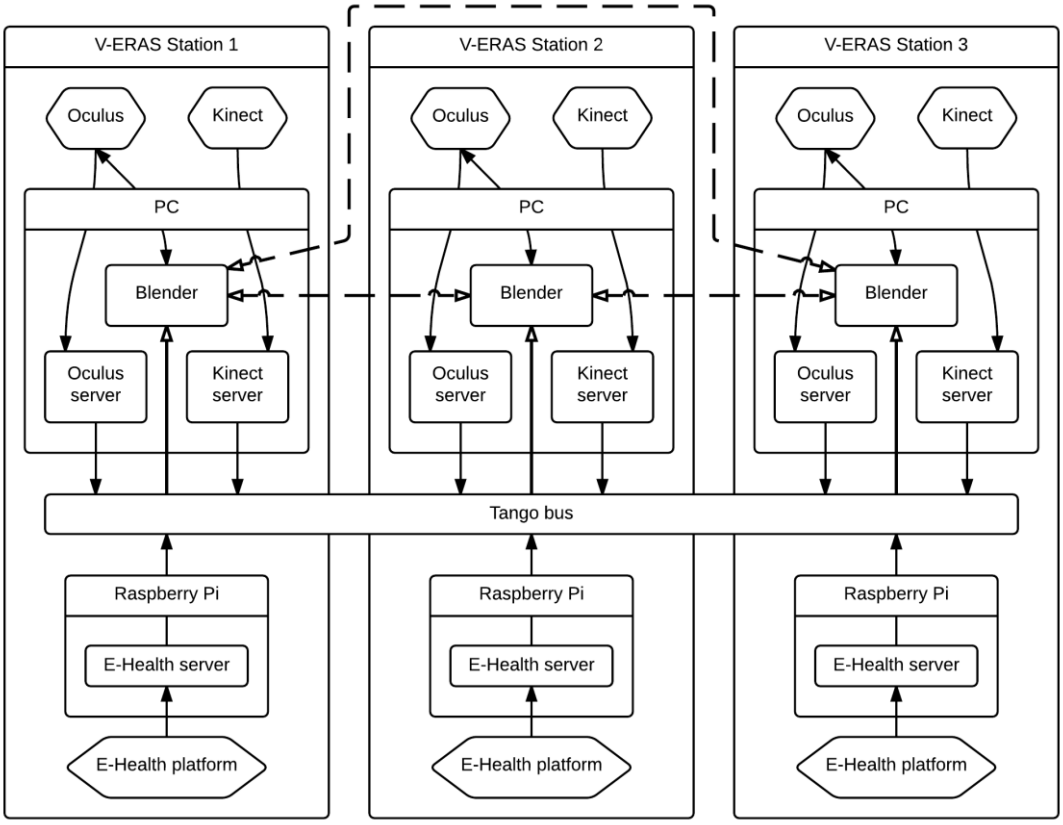


Figure 8. Implementation diagram of a peer-to-peer architecture. All the Blender instances read data from the Tango bus and communicate with each other using sockets.

Here all the Blender instances read data directly from the Tango bus, and use sockets (indicated with dashed lines) to check for consistency.

3.2.2 Tango

Tango provides higher-level facilities, and is a better choice for several reasons:

- It is already deeply integrated in V-ERAS.
- It already provides many facilities.
- It does not require writing/using a separate system.

In Figure 9 it is possible to see how Tango is the only communication channel between the stations. Here all the hardware devices share their data on the Tango bus, and the Blender instances access them directly, thus creating a peer-to-peer network.

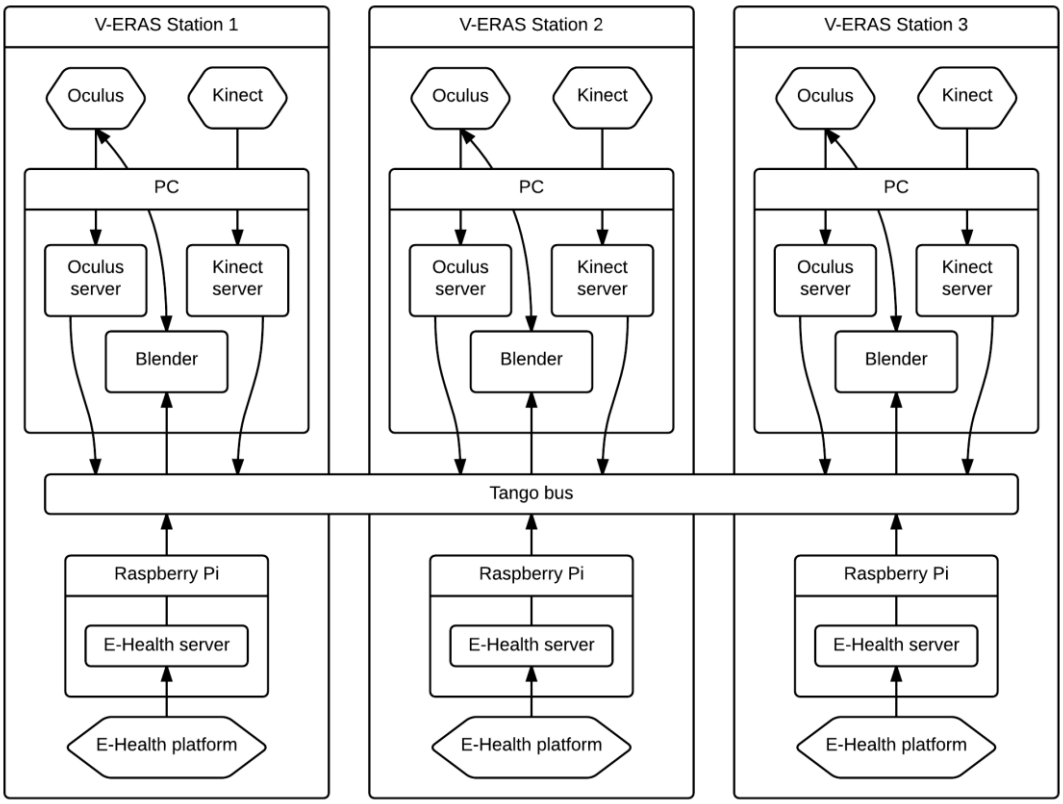


Figure 9. Implementation diagram of a peer-to-peer architecture where the Tango bus is the only communication channel.

Tango provides two communication methods: polling and events.

3.2.2.1 Polling

When polling is used, the server continuously reads the data from the device at a fixed interval (e.g., every 100 ms), and updates the device attributes on the Tango bus accordingly. The clients also access the data from the Tango bus in a similar fashion.

This means that if data are written on the Tango bus faster than the clients are requesting them or if a client is busy and delays the reading, some data might go missing as they are overwritten by most recent data. If instead the client is requesting data faster than the server is writing them, the same value might be read more than once by the clients. Due to hardware limitations, the polling interval is also inaccurate, so for an interval of 100 ms there is no guarantee that the requests will be executed exactly every 100 ms — a difference of a few milliseconds might occur.

This leads to two main issues:

1. If the data are relative (e.g., they are offsets that should be based on the previous value), missing values will cause errors and the simulation will start diverging.
2. If the data are absolute (e.g., they are the absolute position of an object in the simulation) the simulation will maintain coherence. However, the animations might be less fluid when data go missing or they are read more than once.

Figure 10 illustrates these issues:

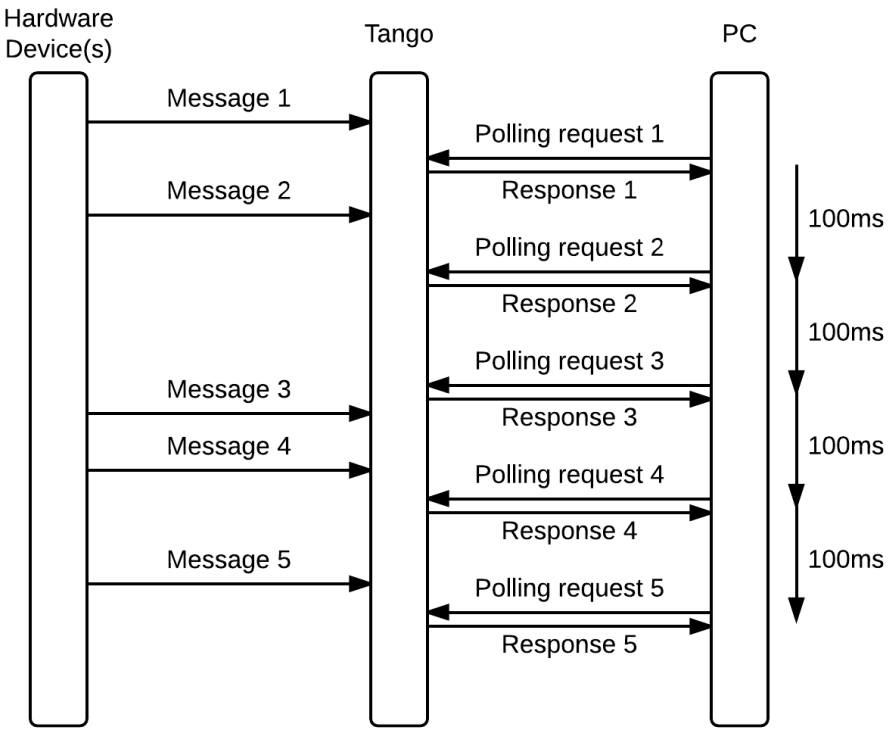


Figure 10. An event interaction diagram of the communication between and hardware device and a PC that uses polling to retrieve data from the Tango bus.

On the left, we have one (or more) hardware device writing data on the Tango bus at irregular intervals (e.g. whenever the user moves). On the right, we have one of the PCs, requesting data from Tango with a fixed interval of 100 ms. This is what happens:

1. The hardware device sends Message 1 to the Tango bus.
2. The PC sends the Polling Request 1 to the Tango bus, and Tango replies with the value received in Message 1.
3. The hardware device sends Message 2 to the Tango bus.
4. After 100 ms, the PC sends the Polling Request 2 to the Tango bus, and Tango replies with the value received in Message 2.

5. After 100 ms, the PC sends the Polling Request 3 to the Tango bus, and Tango replies again with the value received in Message 2, since no new values have been received from the hardware device.
6. The hardware device sends Message 3 to the Tango bus.
7. The hardware device sends Message 4 to the Tango bus.
8. After 100 ms, the PC sends the Polling Request 4 to the Tango bus, and Tango replies with the value received in Message 4. Here the value of Message 3 has been replaced by the one from Message 4 before the PC could request it, thus getting lost.
9. The hardware device sends Message 5 to the Tango bus.
10. After 100 ms, the PC sends the Polling Request 5 to the Tango bus, and Tango replies with the value received in Message 5.

The first problem happens at step 5, where the same message is received twice. If the message contains relative data such as “move forward one meter”, then it might result in the avatar moving forward two meters if the PC does not check for duplicate messages. Similarly, at step 8 one of the two values gets lost, and this might result in the avatar moving forward one meter instead of two. As mentioned above, both of these issues might be solved by using absolute values from the hardware device.

A possible advantage of using polling is that it might be easier to implement than the event-driven alternative. Using polling might also be more effective when the hardware devices write large amounts of data, since it limits the maximum number of requests and avoids bandwidth saturation.

3.2.2.2 Events

Unlike polling, where the clients have to request data from the Tango server, with events the Tango server broadcasts data to all the clients as soon as it reads them from the device. This ensures that all the data are sent to the clients, and also reduces the delay and avoids missing and duplicating data, thus making it a preferable approach.

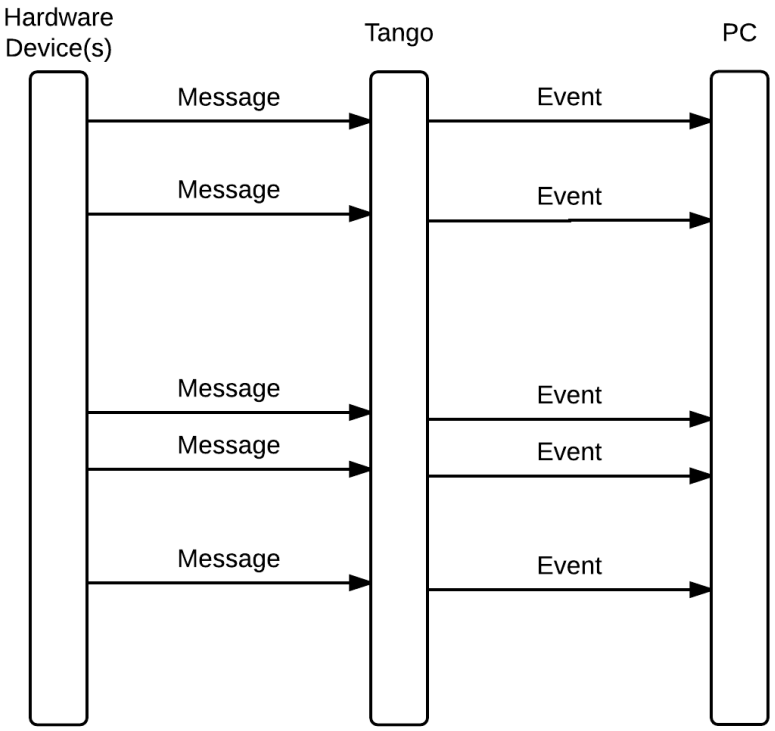


Figure 11. An event interaction diagram of the communication between and hardware device and a PC that uses events to retrieve data from the Tango bus.

In Figure 11 we can see that, unlike the polling approach, the events are sent every time a message is received without any delay.

Unfortunately the documentation for the Tango event system is very limited, and the whole event system has been recently rewritten to use ZeroMQ instead of the CORBA Notification Service. This resulted in issues (TANGO Control System, 2014a) (TANGO Control System, 2014b) and subsequent compatibility problems between the Tango version used for the project and the default ZeroMQ implementation provided by the Linux distribution.

Another issue with this approach is that an event-driven approach might be more difficult to implement and integrate with the other frameworks and software.

Despite this, using Tango and events seems to be a promising approach, especially once all the issues have been ironed out.

3.2.3 Blender plugins

There are several different plugins and scripts for Blender that can be used to implement a networking sub-system. However there does not seem to be any de facto standard.

Most of these plugins seem to use sockets to enable communication among the different Blender instances, thus using an architecture similar to the ones shown in the Sockets section, but different architectures are also available.

Given the complexity of the project, it is likely that these plugins would need to be adapted and integrated with the data read by the Tango bus. Depending on architecture of the plugin, this might eventually turn out to be more time-consuming than developing an ad hoc system.

Due to these reasons it was decided that, for the time being, it was not worth spending further time investigating Blender-based solutions, even though some of them might potentially be useful.

3.2.4 MORSE multi-node simulation

MORSE includes support for multi-node simulations, using a server-client architecture. In addition to the MORSE nodes (the clients), there is a separate synchronization server whose task is to synchronize the events happening in the client nodes, as shown in Figure 12:

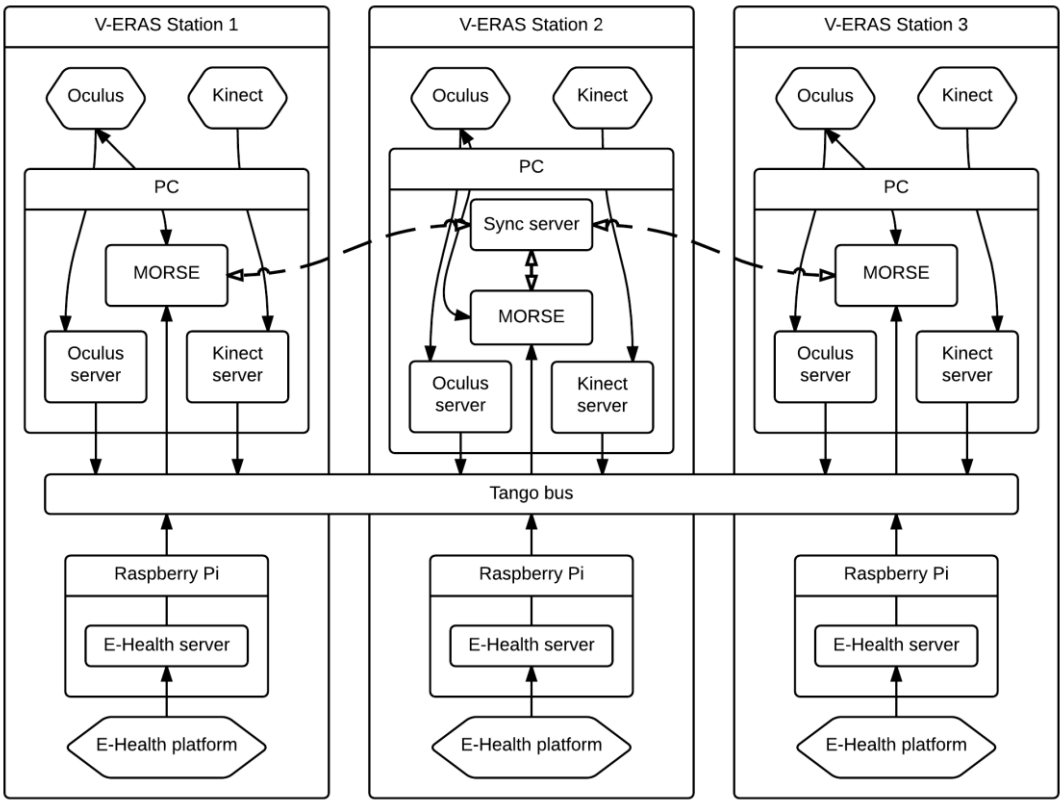


Figure 12. Implementation diagram of a server-client architecture with a MORSE synchronization server running on the PC of the V-ERAS station 2 and all the MORSE instances connected to it.

From the documentation (Anon., 2014c), it appears that running the server on the same machine of one of the clients (as depicted in Figure 12) is possible, so an additional machine might not be required — this could, however, cause overhead on the machine running the server. The multi-node component of MORSE can also use either sockets (Anon., 2014d) or HLA (High-Level Architecture) (Anon., 2014e).

Using the multi-node component of MORSE would avoid the need for rewriting a synchronization system from scratch, but it also requires MORSE as a dependency. At the time of writing, it is not yet clear if MORSE is a suitable framework for the simulation sub-system, even though it appears to be featureful and preliminary tests seem promising. If MORSE is adopted, then the multi-node component might be used, but the other approaches could be considered as well.

Whether this is a feasible approach or not will be determined by future tests, along with further tests that will have to determine if the server can indeed be run on one of the existing machines (where one of the clients is) and if this affects performances in any way.

3.3 Ensuring coherence

This problem can be addressed in two ways:

1. Pre-emptive;
2. corrective;

3.3.1 Pre-emptive approach

A pre-emptive approach aims at ensuring that no incoherences are introduced in the simulation. This can be done by ensuring that all the Blender instances start from the same state and that they all receive exactly the same inputs. While this could in theory be done, it is not possible to ensure that the same inputs are received at the same time, due to external factors (e.g., network latency, machine performance). The simulation could, however, be designed so that delays in the inputs do not cause inconsistencies.

3.3.2 Corrective approach

A corrective approach aims at correcting, rather than preventing, incoherences. A way to correct incoherences is to broadcast at regular intervals the absolute positions of the objects, so that the clients can update their positions in case they are not correct.

4 IMPLEMENTATIONS TESTING

In order to verify the correct functioning of the selected approaches, it was decided to create an extensive and automated test suite. This is achieved by setting up a test environment with one or more instances of the simulation, executing different kind of commands, and checking if the observed results match the expected ones.

This chapter provides a detailed explanation of the tests that have been conducted and why this was deemed necessary.

4.1 Reasons for using automated tests

There are several reasons that led to the decision of using automated tests:

1. Automated tests are a fundamental part of every non-trivial piece of software, as they provide a way to identify problems and bugs early in the development.
2. The V-ERAS code, the code of the frameworks and libraries we are using, and even the operating system keep evolving and changing, and tests allow us to make sure that changing or updating any of these components does not introduce failures.
3. The project requires setting up different components (such as Tango and its database), the availability of certain packages with specific versions, and specific permissions. The developers of the team use different hardware and run different operating systems and different software versions. Tests allow us to quickly verify the correct functioning of the project while installing it on a new or different machine, and to easily identify the missing components in case of failures.
4. Different hardware components are used as part of the project (e.g., the Oculus Rift and the Kinect) and requiring every member to have them available, configured, and connected would be expensive and impractical. Tests allow us to use recorded or fabricated data that eliminate the requirement of the hardware.

5. Using fabricated data enables us to include a wide range of different scenarios and situations, including cases that are very difficult or even impossible to reproduce manually (e.g., sending three conflicting commands from three different stations at the exact same time).
6. Whenever a bug is found, a regression test test is written, not only for aiding the debugging and ensuring that the fix works properly, but also for verifying that the bug does not present itself again in the future or on other machines.

4.2 Tests organization

Different kinds of files have been created to test the behavior and functionality of the single components (unit tests) and their interaction (integration tests).

All the automated tests are designed to be run on a single machine, even though they run different clients and simulate the behavior that would occur in a multi-machine environment. Additional tests have been executed manually on multiple machines to ensure that they indeed work on a multi-machine environment.

4.2.1 Patch scripts

In order to work around bugs in the software and libraries we are using, it was necessary to patch some of them. To simplify the task, additional scripts have been created.

In the repository (Melotti, 2014), there are three scripts used to patch PyTango, Blender, and ZeroMQ. These scripts are meant to be executed once and only if necessary (i.e., only if the installed versions require patching):

- patch_pytango3.py: used to patch PyTango and make it work with Python 3.
- patch_blender.py: used to patch the Blender plugin used to export the Blender standalone runtime.

- patch_zmq.py: used to install a version of ZeroMQ that is compatible with Tango.

4.2.2 Blender files

The directory contains a sample Blender file used by the tests:

- utest.blend: a simple Blender file used by test_blender.py that contains an empty scene with a single cube.
- utest.py: Python script used by utest.blend to move the cube and retrieve its position.

4.2.3 MORSE files

The directory also contains two files used by test_morse.py:

- morsetest.py: a simulation scenario script used to create a test scene within MORSE. This scenario includes an empty environment and a single robot that will be remotely controlled by the test.
- morse_notifier.py: a middleware script necessary to work around a shortcoming of pymorse. Even though it is possible to run several MORSE instances on the same machine without conflicts, pymorse can only handle a single connection with one of them. This problem is solved by running several morse_notifier scripts, each with its own separate pymorse connection. This script also accepts several parameters in order to support several different input and output methods (e.g., pipes, queues, Tango).

4.2.4 Utils scripts

Two utils scripts have been created to automate some tasks and can be invoked either manually or imported and used directly from the tests:

- create_blender_runtime.py: used to create the standalone runtime used by test_blender.py. This should be used whenever utest.blend is changed to create an updated runtime.
- register_test_server.py: this adds the testtango server to Jive. The tests already use it automatically to add/remove the test Tango server, but it can also be used on its own to register the test server while running manual tests.

4.2.5 Test files

There are four test files:

- testtango: a test Tango server, used by test_tango.py, test_blender.py, and test_morse.py.
- test_tango.py: tests for Tango.
- test_blender.py: tests for Blender and Tango-Blender integration.
- test_morse.py: tests for MORSE and Tango-MORSE integration.

4.2.5.1 testtango

testtango is an executable script that implements a mock Tango server. The server defines and exports two attributes:

- **`loop0to9`**: a read-only and polled integer attribute. Every 200 ms its value is updated with the consecutive value in range 0 to 9, and restarts from 0 once 9 is reached.
- **`writablecmd`**: a read-write and non-polled string attribute. When the server is started its value is `'INITIAL VALUE'`. The value will then be updated by the clients connected to the server.

4.2.5.2 test_tango.py

The main goal of this test file is to ensure that Tango and its main features work properly.

All the tests have a similar and simple structure: they spawn testtango in a sub-process and use different techniques (direct attribute access, polling, and events) to communicate with it:

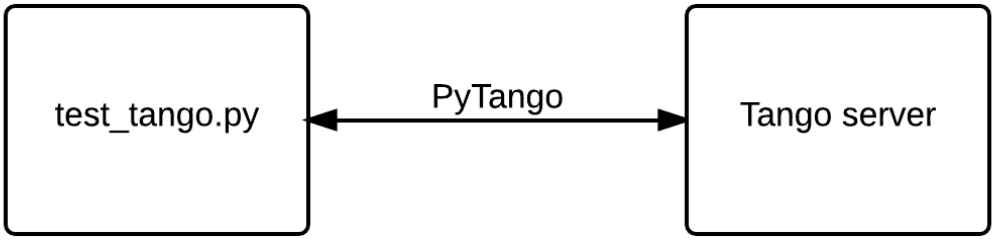


Figure 13. The test_tango.py process starts the Tango server and uses PyTango to communicate with it.

The test file includes seven different tests:

- test_tango_server: a simple smoke test that checks if starting and stopping the testtango server works.
- test_read_attribute: checks if reading from the loop0to9 attribute works.
- test_read_write_attribute: checks if it is possible to write on the writablecmd attribute and if subsequent readings return the updated value.
- test_read_polled_attribute_with_events_stateful: this test is similar to test_read_attribute, but instead of reading the attribute directly, it registers a callback that is invoked whenever the value of the attribute changes. Since this test reads from the loop0to9 attribute, the events are generated automatically every 200 ms, and the test terminates when all the numbers in range 0 to 9 are received.
- test_read_polled_attribute_with_events_stateless: as above, but using stateless event subscription.
- test_unpolled_attribute_with_events_stateful: this is similar to test_read_polled_attribute_with_events_stateful, but instead of accessing a polled attribute (loop0to9), it access an unpolled attribute (writablecmd). In this test the value of the attribute is updated, and it is

verified that an event is triggered for every update. The updated value of the attribute received in the event is also verified.

- test_unpolled_attribute_with_events_stateless: as above, but using stateless event subscription.

After several attempts, all the tests passed, proving that using Tango (either with or without events) is a viable solution. While testing the reading of polled attributes with events, it was observed that some of the values were missing, thus making this approach less preferable.

4.2.5.3 test_blender.py

The main goals of this test file are to:

- Test that the standalone Blender runtime can be started and stopped.
- Test that it is possible to get and set the positions of the objects.
- Test the integration with Tango.

It comprises five tests:

- test_objects_dump_one_instance: a simple test that starts the Blender standalone runtime and instructs it to serialize the position of the objects in the scene in a JSON file. The test then reads the content of the JSON file and checks that the serialization was successful.

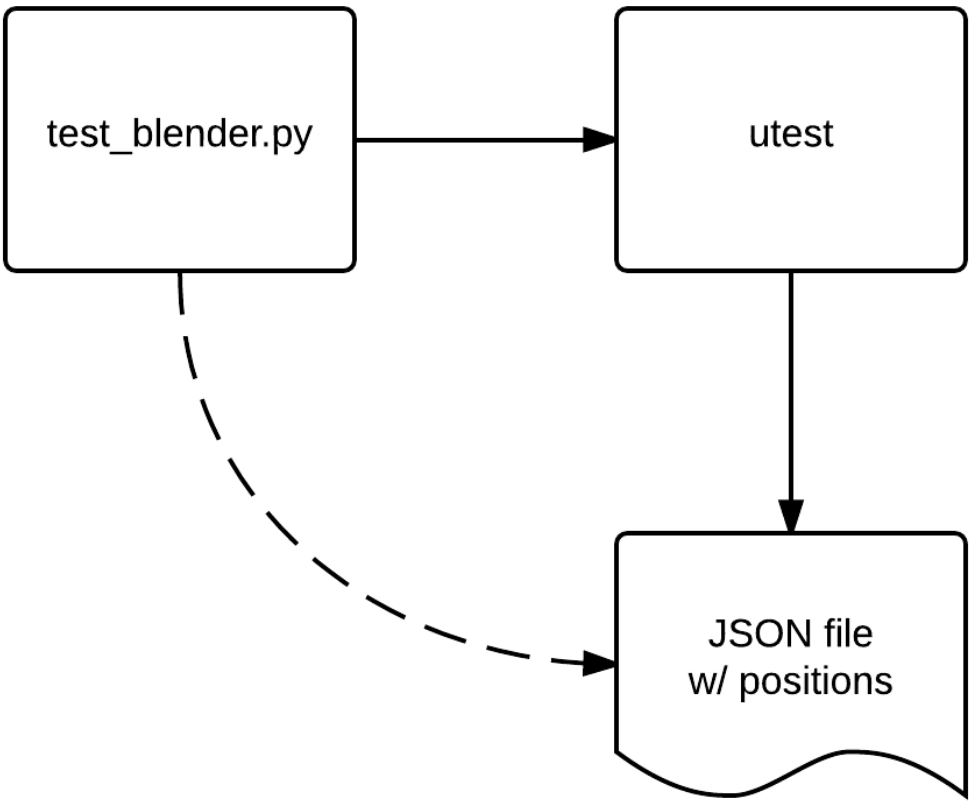


Figure 14. The test_blender.py process starts the utest process that creates a JSON file with the positions of the objects. test_blender.py then reads the content of the file and checks the positions.

- test_objects_dump_multiple_instances: as above, but it runs three separate instances of the Blender standalone runtime. The instances generate three JSON files that are read by the test in order to verify that all of them can serialize and dump the objects positions without conflicts.

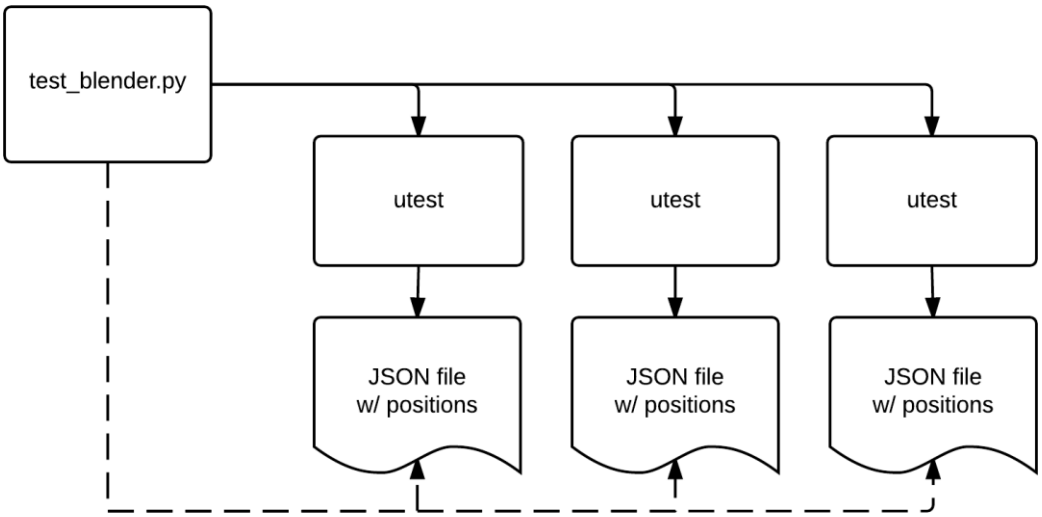


Figure 15. The test_blender.py process starts three utest processes that create three JSON files with the positions of the objects. test_blender.py then reads the content of the files and checks the positions.

- test_tango_server: a smoke test identical to the one in test_tango.py to ensure that Tango works properly before proceeding with the Tango-Blender integration tests.
- test_tango_blender_one_instance: this tests starts the Tango server and the Blender standalone runtime and waits until they are both operational. During this test, Blender is instructed to subscribe to Tango events for the **`writablecmd`** attribute. The test then writes on the Tango server triggering events that are received by Blender. The events contain messages that instruct Blender to move a cube in different position and to terminate the simulation. Before terminating the simulation, Blender serializes the objects positions on a JSON file so that the test can verify that the final position of the cube is correct.

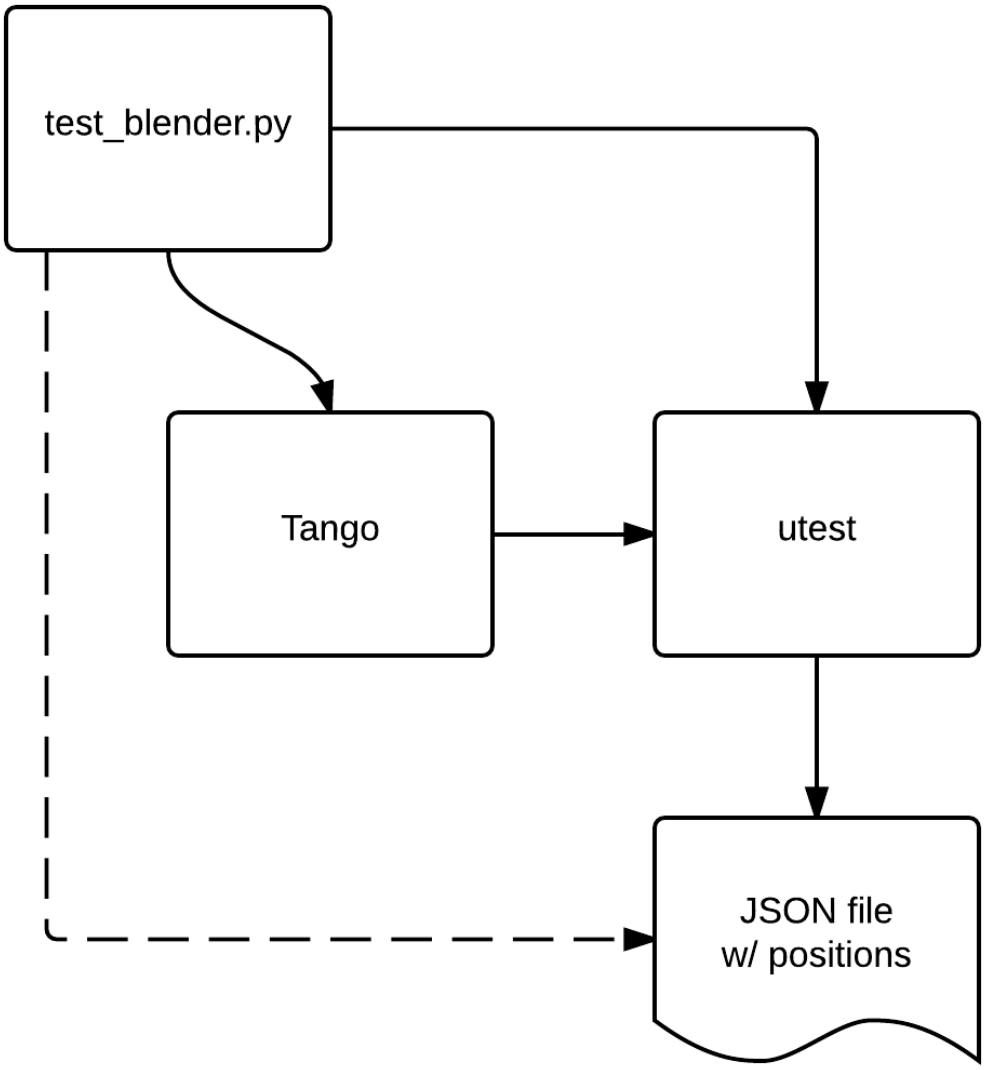


Figure 16. The test_blender.py process starts the utest process and the Tango server, and then instructs Tango to move the objects in the utest scene. The final positions of the objects are then written in a JSON file that gets read and checked by test_blender.py.

- test_tango_blender_multiple_instances: this is similar to the previous test but it runs three Blender instances instead of one. All the instances subscribe to Tango events for the **`writablecmd`** attribute, so that every time its value is updated, the event is triggered and sent to all the instances. Eventually the instances serialize the objects positions and

quit, and the test verifies the all the positions are consistent and that they match the expected results.

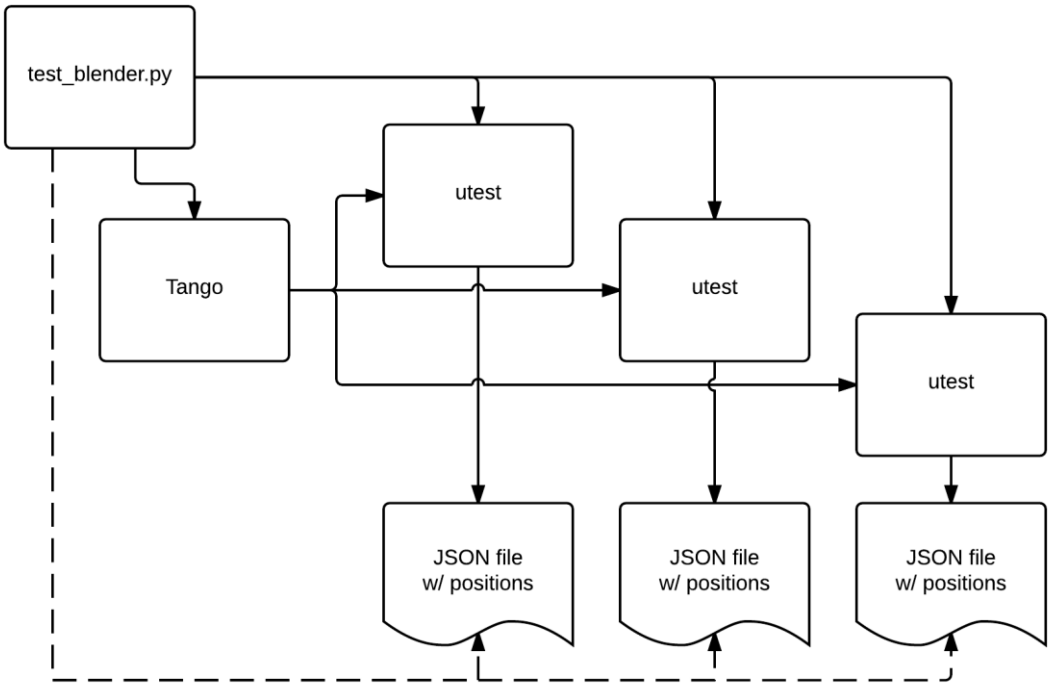


Figure 17. The test_blender.py process starts three utest processes and the Tango server, and then instructs Tango to move the objects in the utest scenes. The final positions of the objects are then written in three JSON files that gets read and checked by test_blender.py.

Using the Blender standalone runtime and subscribing to events generated by a Tango server also proved to be a viable solution. Even though the focus of these tests was on the use of events, it has been verified by other members of the team that direct access of polled attributes is possible. However, events might be preferable for the reasons listed in the Events section.

4.2.5.4 test_morse.py

This test file is similar to test_blender.py but checks the functioning of MORSE. Its main goals are to:

- Test that the MORSE simulation can be started and stopped.
- Test that it is possible to get and set the positions of the robot.
- Test how pymorse interacts with MORSE.
- Test the integration with Tango.

It comprises six tests:

- test_morse: a smoke test that ensures that MORSE can be started in a sub-process and stopped without errors.
- test_morse_one_instance: a simple test that loads the MORSE test scene in a sub-process, and uses pymorse to directly move the robot and verify that its final position is correct.

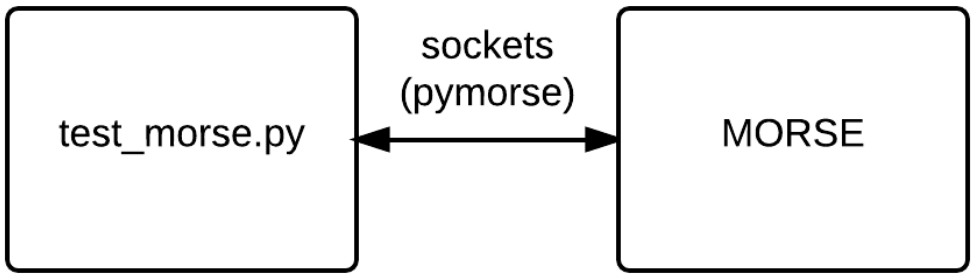


Figure 18. The test_morse.py process starts the MORSE process and uses pymorse to communicate with it.

- test_morse_one_instance_with_morse_notifier: this test is similar to the previous one, but instead of using pymorse directly, it launches MORSE and the morse_notifier in two sub-processes and uses the latter to control the MORSE simulation. The morse_notifier moves the robot according to a list of directions passed to the sub-process via command

line, and reports back the final position of the robot via stdout. The position is finally checked by the test.

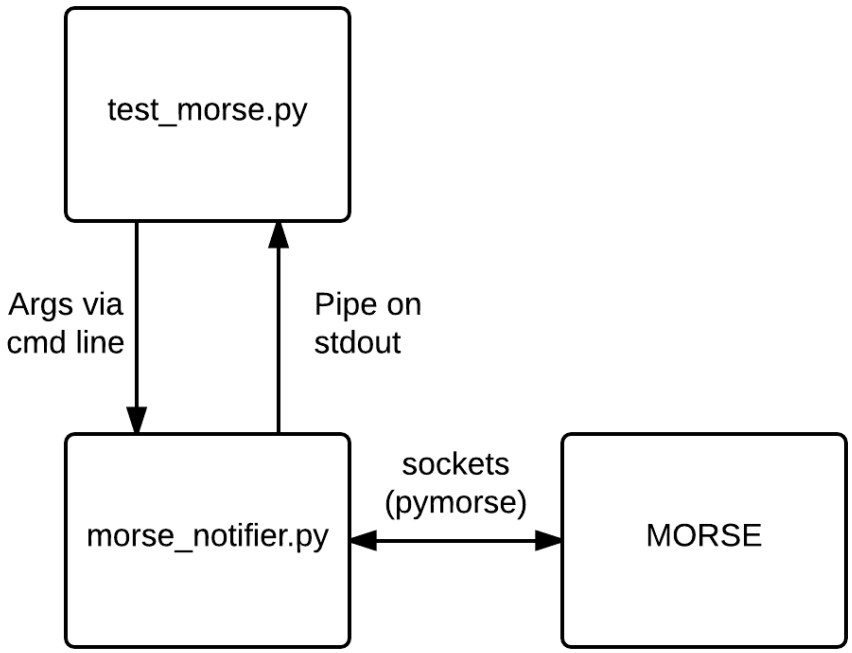


Figure 19. The test_morse.py process starts the morse_notifier.py and the MORSE processes and uses the morse_notifier to communicate with MORSE.

- test_morse_one_instance_with_morse_notifier_and_multiprocessing: this test is also similar to the one above, but instead of using subprocess, passing the directions to it and reading the results from stdout, it uses multiprocessing and communicates with the process using a queue. This approach is to be considered an experiment and it has not been explored further, even though it did not show any major downsides.
- test_tango_server: a smoke test identical to the one in test_tango.py to ensure that Tango works properly before proceeding with the Tango-MORSE integration tests.
- test_morse_one_instance_with_morse_notifier_and_tango: this test builds on top of test_morse_one_instance_with_morse_notifier and adds a third sub-process for Tango. Instead of passing the direction via

command line to the notifier, the test writes them to the Tango bus using PyTango, and the notifier (instructed by the test to subscribe to Tango events) receives and forwards them to MORSE. The final position is then read by the notifier through pymorse and returned to the test via a pipe on stdout.

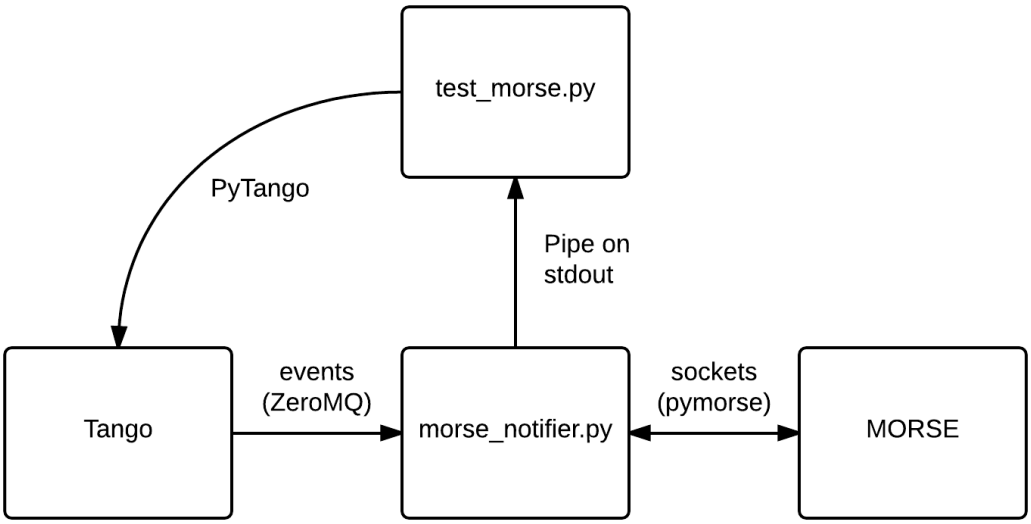


Figure 20. The test_morse.py process starts the morse_notifier.py and the MORSE processes and the Tango server and uses Tango to communicate with MORSE via the morse_notifier.

- test_morse_three_instance_with_morse_notifier_and_tango: the final test is also similar to the previous one, but instead of having only a notifier and a MORSE instance running, it has three for each. All the MORSE instances are controlled by their respective notifiers that, similarly to the previous test, receive commands from a single Tango instance.

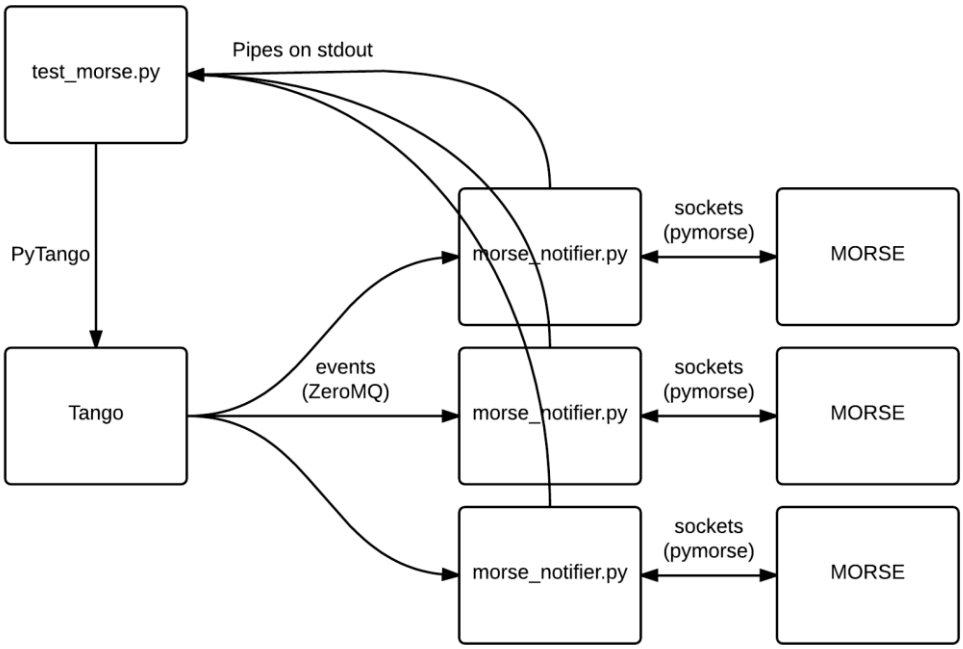


Figure 21. The test_morse.py process starts three morse_notifier.py and three MORSE processes and the Tango server and uses Tango to communicate with all the MORSE instances via their respective morse_notifiers.

The test writes the direction of the robot on the Tango bus and this triggers an event that is received by the three notifiers and the command is then forwarded to their respective MORSE instances. The notifiers then access the robot position through pymorse and report it back to the test via pipes on stdout. Finally, the test checks that all the positions are correct and coherent.

These tests proved to be extremely difficult to implement properly, and even if the final implementations can be considered stable, there are still sporadic failures and errors, mostly during the startup and shutdown phases and while trying to establish connections between the sub-processes.

The implementation of pymorse has some limitations and issues, most notably the fact that it cannot handle connections to multiple MORSE instances, thus requiring the use of morse_notifier.py and further increasing the number of processes involved in the tests and consequently the complexity and fragility of the tests.

Another major problem is the lack of an effective way to determine if the MORSE simulation is fully loaded and ready to accept commands; the notifier is also affected by similar problems. Without an adequate way to ascertain the status of the sub-processes the only viable solution is to use timeouts and wait for a fixed amount of time before attempting communication. However this has two downsides. First, if the timeout is too short, the sub-process might still not be ready when the time has elapsed and this might cause errors and failures. Second, if the timeout is too long, the tests will needlessly wait and thus require more time to run, and this might affect the workflow negatively, especially when a large number of tests is required.

Fortunately, these issues (including problems during the process shutdown) should not affect negatively the final simulation, because eventually all the processes will have to be started only once. Since this is done manually, it is possible to wait long enough and even restart the processes in case of failure. Once all the processes are up and running and all the connections have been successfully established, the simulation can start without further problems.

5 CONCLUSION

Thanks to the tests, it was possible to determine the strengths and weaknesses of the different implementations and to decide what would be more suitable solution.

Eventually, Tango proved to be the best solution due to its extensive adoption within the project, stability, and to the availability of all the required features. The decision was taken despite some minor issues, also because the Tango development team has been responsive at responding to the bug reports.

As mentioned above, Tango supports two communication methods: polling and events. Both methods turned out to be useful in different contexts, and since they can both be used together, the more apt can be adopted where needed.

This last chapter describes the final system design and discusses open issues and future plans.

5.1 Final system design

The following image shows in detail the final design of the system:

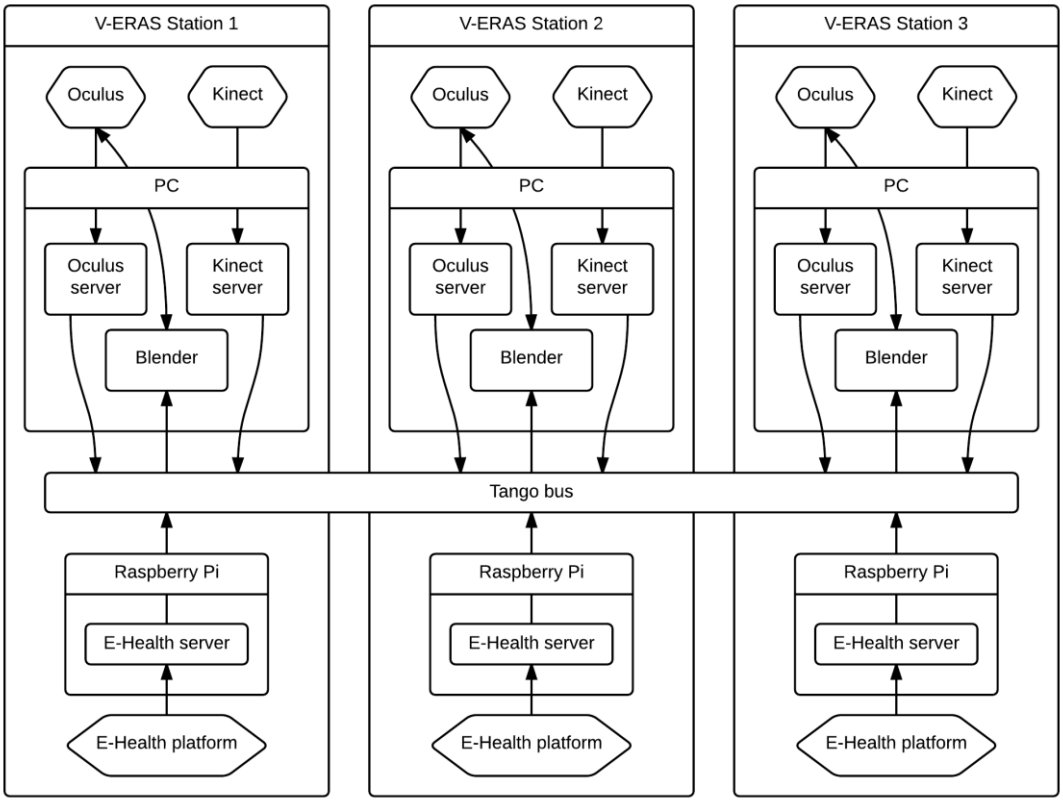


Figure 22. The final design of the networking sub-system, using the Tango bus as the only communication channel.

Every V-ERAS station comprises:

- an Oculus Rift;
- a Kinect;
- a PC;
- a Raspberry Pi;
- an E-Health sensor platform mounted on the Raspberry Pi;
- a Motivity (not depicted in Figure 22, since it's a passive component);

The PC runs Blender, the Oculus Tango server, and the Kinect Tango server, while the Raspberry Pi runs the E-Health Tango server.

The Oculus Tango Server reads sensors data relative to the head position from the Oculus Rift and writes them on the Tango bus, so that the other Blender instances can read them and update the position of the avatar's head accordingly during the simulation.

Similarly, the Kinect Tango Server reads data relative to the body position from the Kinect and writes them on the Tango bus. These data will then be used by the Blender instances to update the positions of the avatars and also to calculate movements relative to the environment.

The Raspberry Pi runs the E-Health Tango Server that gathers data from the mounted E-Health platform and its numerous sensors. Note that this and the PC are the only two components able to run Tango servers.

Blender is responsible for running the simulation and sending the generated images to the Oculus Rift. It also reads the position of the users and their heads from the Tango bus, and uses it to update the avatars in the simulation. In order to minimize the latency of the images sent to the user whenever he moves the head, Blender can also read sensors data directly from the Oculus Rift. In addition, the data received from the E-Health platform can be displayed to the avatars within the simulation both on a virtual HUD or on screen located inside the virtual station.

The V-ERAS project is expected to use up to four V-ERAS stations, even though it is theoretically possible to connect more stations. If necessary, external machines can easily access the Tango bus without affecting the architecture or the functioning of the stations.

5.2 Open issues

While certainly useful, the test suite is not able to easily check how the final system will behave when three or more stations are connected. The final design takes this into account by making the support of alternative and fallback methods easy to implement in case problems with the current design arise.

If necessary, the system design will be updated based on the feedback received by doing manual tests with the complete setup.

5.2.1 Oculus Rift communication

In the final design, the Blender instance reads data directly from the Oculus Rift rather than accessing the data from the Tango bus in an effort to minimize latency. This decision has been taken because higher latency has been linked to motion sickness symptoms in the user, especially after prolonged sessions. Motion sickness is caused by a mismatch between the position perceived by the vestibular system and the one seen by the eyes, so reducing or eliminating the mismatch by lowering the latency is of crucial importance.

The Blender instance could easily read data from the Tango bus, if this proves to have other advantages and if the changes in latency do not cause motion sickness in the users.

5.2.2 Ensuring coherence

The goal of ensuring coherence among the simulation is met by making sure that all the simulations read the same data from the Tango bus and update the simulation accordingly. Events are used to guarantee that all the data are received by the Blender instances, and where this is not essential (as is the case with absolute data) polling is also used.

In theory, this should be enough to guarantee that all the simulations are coherent, but this has only been verified in simplified test scenarios that are run for a limited amount of time. In a long-running simulation and more complex scenes, divergences might still arise.

5.2.3 Polling versus events

Since most of the data written on the Tango bus are absolute, losing some of the values while reading is a lesser concern. This makes both polling and

events viable solutions and thus the best approach can be determined while running a full simulation with three or more V-ERAS stations.

Since some of the Tango servers produce a great amount ofdata (for example, the body tracking server writes on the Tango bus a set of three-dimensional Cartesian coordinates for each tracked joint), using events might turn out to be too expensive for both performances and bandwidth. If this proves to be true, falling back on polling is not difficult, even if this might result in some values being lost and cause slightly less fluid avatar movements.

5.2.4 MORSE adoption

MORSE initially looked promising, and extensive tests have been done with it. However, it turned out that the project is still not mature enough and that integrating it with the rest of the project is not an easy task. The pymorse module also has issues and limitations that might affect the stability of the system.

Despite this, MORSE still provides several useful features, and will probably be reconsidered in the future. Since it is based on Blender, it should not be too difficult to replace Blender with MORSE, and adapt the rest of the system to work with it.

5.3 Further additions

The final design also takes into account the possibility of further additions, such as additional machines or Tango servers. This section outlines some possible additions that have been discussed.

5.3.1 Voice recognition server

The IMS has been active in researching and developing a voice recognition software used to interact with the environment (for example, to open and close doors in the virtual station or to request status information) and with rovers and other devices. An additional Tango server used to listen to the user and convert

the voice inputs in commands, can be run on the Raspberry Pi of each user (that will eventually be integrated in the space suit).

5.3.2 Blender synchronization server

As mentioned in the Open Issues section, the current design might not be able to ensure coherence in case of long-running simulations and complex environments. The use of additional Tango servers used by the Blender instances have been discussed, and they will be implemented if necessary.

Each V-ERAS station will run a Tango server used by the Blender instance to write data on the Tango bus. Each Blender instance will be responsible for one of the avatar, and will periodically broadcast its position to the other instances, so that they can verify if any divergence occurred and correct it by using the position provided.

5.3.3 Mission control display

During the simulation, a mission control crew will also be present. This crew will observe and possibly interact with the users of the simulation, so it will be necessary for them to access the simulation data.

This can be implemented with an additional machine connected to the Tango bus that will retrieve and display detailed information about the status of the users and also show on a screen or projector a third-person view of the simulation. This machine will not participate actively to the simulation and only read data from the bus, so its impact on the system will be minimal.

5.4 Future plans

The Italian Mars Society is planning to keep expanding and improving the project and testing and researching the best technologies available.

The first V-ERAS mission has been scheduled for December 2014, during week 50, at the Dolomites Astronomical Observatory / Carlo Magno Hotel in Madonna di Campiglio, Trento (Italy). An international team of eight members has already

been selected, including three crew members, three mission control members, and an outreach communication team of two members.

This first mission will allow the IMS to perform a real test of V-ERAS and lay the foundation for all the future V-ERAS missions, and eventually the construction of a real ERAS station. Collaborations with other organizations are also being planned.

If these efforts prove to be successful, they will be a big step towards the colonization of the red planet.

6 REFERENCES

Anon., 2014a. *What is MORSE? -- The MORSE Simulator Documentation.* [Online]
Available at: http://www.openrobots.org/morse/doc/latest/what_is_morse.html
[Accessed 23 10 2014].

Anon., 2014b. *Raspberry Pi.* [Online]
Available at: http://www.raspberrypi.org/
[Accessed 23 10 2014].

Anon., 2014c. *Multi-node simulation -- The MORSE Simulator Documentation.* [Online]
Available at: http://www.openrobots.org/morse/doc/1.2/multinode.html
[Accessed 23 10 2014].

Anon., 2014d. *Multi-node Simulation using sockets -- The MORSE Simulator Documentation.* [Online]
Available at:
http://www.openrobots.org/morse/doc/1.2/user/multinode/socket.html
[Accessed 23 10 2014].

Anon., 2014e. *Multi-node Simulation using HLA -- The MORSE Simulator Documentation.* [Online]
Available at: http://www.openrobots.org/morse/doc/1.2/user/multinode/hla.html
[Accessed 23 10 2014].

Blender Foundation, 2014. *About - blender.org - Home of the Blender project - Free and Open 3D Creation Software.* [Online]
Available at: http://www.blender.org/about/
[Accessed 23 10 2014].

Cohen, M. M., 2011. *Is NASA Ready for Deep-Space Human Spaceflight?* [Online]
Available at: http://spirit.as.utexas.edu/~fiso/telecon/Cohen_5-11-11/Cohen_5-

11-11.pdf
[Accessed 23 10 2014].

iMatix Corporation and Contributors, 2014. *Code Connected - zeromq.* [Online]
Available at: http://zeromq.org/
[Accessed 23 10 2014].

Libelium Comunicaciones Distribuidas S.L., 2014. *e-Health Sensor Platform V2.0 for Arduino and Raspberry Pi [Biometric / Medical Applications].* [Online]
Available at: http://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical
[Accessed 23 10 2014].

Melotti, E., 2014. *italianmarssociety / V-ERAS Blender / source / test -- BitBucket.* [Online]
Available at: https://bitbucket.org/italianmarssociety/v-eras-blender/src/default/test/
[Accessed 23 10 2014].

Preiser, W., Rabinowitz, H. & White, E., 1988. *Post-Occupancy Evaluation.* New York: Van Nostrand Reinhold.

TANGO Control System, 2014a. *TANGO Control System / Bugs / #662 ZMQ 4 compatibility.* [Online]
Available at: http://sourceforge.net/p/tango-cs/bugs/662/
[Accessed 23 10 2014].

TANGO Control System, 2014b. *TANGO Control System / Bugs / #689 Problems while getting events on a remote machine.* [Online]
Available at: http://sourceforge.net/p/tango-cs/bugs/689/
[Accessed 23 10 2014].

The Italian Mars Society, 2014a. *The Italian Mars Society.* [Online]
Available at: http://www.marssociety.it/
[Accessed 23 10 2014].

The Italian Mars Society, 2014b. *Home.* [Online]
Available at: http://erasproject.org/
[Accessed 23 10 2014].

The Mars Society, 2014a. *The Mars Society.* [Online]
Available at: http://www.marssociety.org/
[Accessed 23 10 2014].

The Mars Society, 2014b. *Society FAQ - The Mars Society.* [Online]
Available at: http://www.marssociety.org/home/about/faq#TOC-Q:-What-is-the-Mars-Society-doing-to-prepare-for-humans-to-Mars-missions-
[Accessed 23 10 2014].

The Mars Society, 2014c. *Flashline Mars Arctic Research Station.* [Online]
Available at: http://fmars.marssociety.org/
[Accessed 23 10 2014].

The Mars Society, 2014d. *Mars Desert Research Station.* [Online]
Available at: http://mdrs.marssociety.org/
[Accessed 23 10 2014].