The Collective Experience of Empathic Data Systems

ICT-258749

Deliverable 5.1

Specifications and Architecture

Authors	Alberto Sanfeliu Cortés, Andreu Corominas Murtra, Edmundo Guerra Paradas, Alex Goldhoorn, Joan Perez Ibarz, Pedro Omedas, Alberto Betella
Version	f1.0
Date	30.09.2011
Classification	Restricted
Contract Start Date	01.09.2010
Duration	48 months
Project Co-ordinator	Goldsmiths, University of London
File Name	CEEDs_D5.1_f1.0_UPC

Project funded by the
European Community under
the Seventh Framework Programme

Consisting of:

No	PARTICIPANT NAME	S.N.	COUNTRY
1	Goldsmiths, University of London	GOLD	UK
2	Universitat Pompeu Fabra	UPF	ES
3	University of Sussex	UOS	UK
4	Informatics and Telematics Institute	ITI	GR
5	Eberhard Karls Universitaet Tuebingen	EKUT	DE
6	Universität Augsburg	UAU	DE
7	University of Teesside	TEESSIDE	UK
8	Università degli Studi di Padova	UNIPD	IT
9	Max Planck Gesellschaft zur Foerderung der Wissenschaften E.V.	MPG	DE
10	Ecole Normale Superieure	ENS Paris	FR
11	Budapesti Muszaki Es Gazdasagtudomanyi Egyetem	BME	HU
12	Universitat Politecnica de Catalunya	UPC	ES
13	Università di Pisa	UDP	IT
15	Electrolux Italia SpA	ELECTROLUX	IT
16	Leiden University	UL	NL
18	Helsingin Yliopisto	UH	FI

Responsible of the document: UPC

Defined Contributors to the document: Alberto Sanfeliu Cortés, Andreu Corominas Murtra, Edmundo Guerra Paradas, Alex Goldhoorn, Joan Perez Ibarz, Pedro Omedas, Alberto Betella

Document History

VERSION	ISSUE DATE	BY	CONTENT AND CHANGES
V1.0	15.09.2011	UPC	First draft of the document
V2.0	21.09.2011	UPC	Second draft of the document
V3.0	23.09.2011	UPC	Third draft of the document
V4.0	27.09.2011	UPC	Fourth draft of the document
f1.0	30.09.2011	UPC	Final document

Executive Summary

The CEEDs project requires the integration of devices (WP2), a synthetic reality platform (CXIM 2.0, WP4), state of the art of science concepts (WP1) and a set of four applications designed to show the potential of the project (WP6). All this elements are investigated and developed by a set of a heterogeneous group of partners.

At a conceptual level, this integration is defined by the CEEDs architecture and CEEDs Core Features, defined by WP3 and WP8 respectively. At a technological level, integration is defined in this document. Therefore, this document will in section 1 first describe the general integration methodology chosen by the consortium. In section 2 the specifications of the computer system in charge of running the applications are listed. Section 3 is devoted to explaining the integration guidelines following the case of one of the CEEDs application, the Neuroscience application. Due to the dynamics of the project this document will be updated over time, especially the part of the specifications.

The document also provides five annexes: Annex I and II explain the YARP installation, Annexes III and IV show pseudo code implementations of inter process interaction modes and Annex V a describes testbed for the integration of the CEEDS applications called "Empathic Human Robot Interaction in the Hide & Seek Game". This testbed is used by the partner leading the technical integration of the CEEDs project (UPC) to test and validate integration issues.

Table of Contents

List of Figures

1 Integration Methodology

This section will discuss the integration methodology of WP5 in the CEEDs project, starting with a general introduction to the general policy. Next the main concept of defining processes and their interaction through messages are discussed. Thereafter several middleware systems are compared, and YARP the chosen middleware is explained shortly. The section ends with the port naming conventions.

1.1 General Policy

The CEEDs project will implement several applications of different nature, involving in each case a number of partners from the consortium that are providing and developing state of the art ideas in the field. Due to this context, a technical decision has been taken to develop software in an incremental way, taking as a base the CXIM 2.0 architecture, since all applications have to integrate with it.

The proposed approach is to break down each application into a set of processes. This division can be done iteratively and incrementally when new concepts and ideas become clear and need to be implemented in terms of processes. Hence, a process is the key software entity that can be developed and tested independently by a single partner. Once a process is ready, integration will be made by pairs of processes, then triplets and so on.

Therefore, at technological integration level a common software library is proposed, called YARP, to communicate between processes in real-time and finally build a given application but keeping processes decoupled, a requirement due to the multi-partner context of the project.

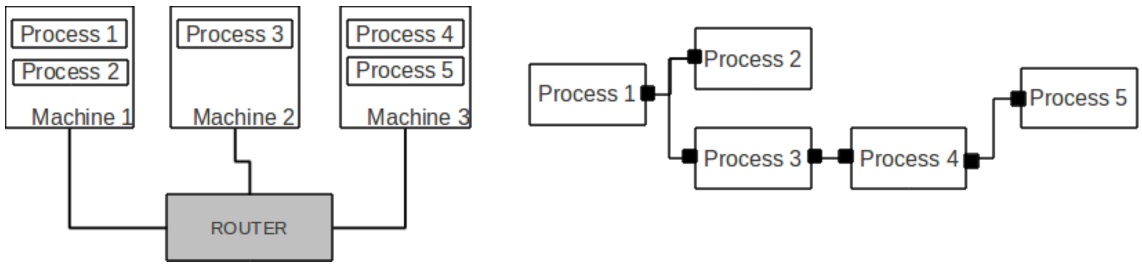
1.2 Processes and Messages

This section provides some integration guidelines in order to maximize the decoupling between partners during the development phase, and to take benefit of modularity of software blocks, especially those that will be used in more than one CEEDs application such as sensor acquisition processes and the rendering engine.

The main concept is to think and implement applications as a set of connected processes that are sending messages between them during the run time, that is a component based software architecture, where each component is a process.

Basic Concepts

In component based software architecture, an **application** is defined as a set of *N* **processes** running concurrently on a set of *M* **machines** that are connected through some **network infrastructure** such as the Internet or a given local area network (LAN/WLAN), see Figure 1a. The abstraction of the processes and how data connections are established between them is the so called **process graph** (Figure 1b) where each **node** is a process, and **links** are set when two nodes interact by exchanging data, through sending and receiving **messages**. Input and output of these messages is done by each process through virtual inputs and outputs, called **ports**. The **process interaction**, made specifically between two different processes through two ports, can be of two types: **publisher / subscriber** or **client / server**. Therefore, for each publisher/subscriber or client/server pair, a message definition should be specified, indicating the data type structure and order that all message contents will fit.



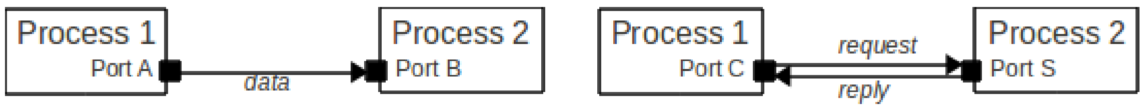
(a) **(b)**

Figure 1: (a) An example of the hardware infrastructure of three machines running five processes. Machines are connected through a LAN router. (b) The process graph abstraction for an arbitrary example application. Small black squares represent ports.

1.3 Interaction Modes between processes

In the publisher / subscriber interaction mode, a process is **continuously outputting** data through a port *A*. Then, each node of the whole application that requires that data being published through port *A*, should declare an own port, let call it port *B*, and subscribe it to port *A*. This will result in a **continuous reception** of the target data in the subscriber processes. A publisher port can have more than one subscriber (see Figure 2a).

The second interaction mode is the client / server situation (see Figure 2b). In this case, a process named as server offers some service upon **requests** arriving to a given port *S*. Other processes requiring this service will be named as client processes and should send a request through an own port connected to port *S*. When a request is received, the server process will compute the request and after that it will **reply** the resulting answer to the client. A service port can receive requests from multiple clients.



(a) **(b)**

Figure 2: (a) The publisher/subscriber interaction mode between port *A* (publisher) and *B* (subscriber) of processes 1 and 2. (b) Client / server interaction mode between ports *C* (client) and *S* (server). First the client sends a request, and once the requested computations have been performed on the server it sends a reply back to the client.

In Annex III you can find example code for YARP in the publisher / subscriber case and in Annex IV for the client / server case.

1.4 Analysis of existing middleware

Nowadays, there exist a huge number of software engineering middleware systems, each of them with its advantages and drawbacks. This section first intends to identify and analyse the existing needs for such software at the CEEDs project. With that knowledge, the most important existing middleware software platforms are analysed in terms of those needs, trying to identify the most suited ones.

1.4.1 Requirements

There are four high priority features that any middleware has to fulfil to be considered as candidate for being deployed within the context of CEES project:

- **Multi-process and multi-computer communications**: Use of middleware to allow several applications (drivers, control programs, etc.) to communicate between them through different processes and computers without having to deal with the low level communication issues. This simplifies the creation of complex networks of computers with several actuators and/or sensors exchanging information to complete a task. Another important aspect of multi-process and multi-computer communications are which kind of communications are supported (data streaming, request/reply, etc.), which communication protocols are supported (TCP, UDP, Bluetooth, etc.), and also how the data sending and receiving is handled (single or multi-threaded, callback functions, etc.). Platform interoperability is also a requirement, as the partners at CEEDs develop in different environments.

- **Generic interfaces**: Use of generic interfaces to allow access to a given feature of a device. That is, the interface for a sensor and actuator included into a hardware platform and the interface for an equivalent independent sensor should look externally the same; however, their implementations can be completely different. These generic interfaces may be provided by the middleware used and/or developed by WP2 and WP5. The interfaces shall be used to standardize the information exchange in the multi-process and multi-computer network using the middleware. Finally, the generation of such interfaces, as well as the necessary messages to be transmitted and received, and software to generate and parse them should be easy to implement and well documented.

- **Robustness**: The middleware used must be robust in several ways:
    - It must be mature enough to avoid bugs present in most new releases.
    - There should be good support from developers, as within this context it is not a rare instance requiring technical assistance from people who know the software at programming-code level.
    - The presence of a strong and participative community of users worldwide is desirable. This usually helps to enrich the software with continuous feedback from users, and deep testing, improving middleware quality.

  Since the middleware creates a network of computers, with actuators and sensors working together, it also must be robust to component failure. Thus, if some of the devices connected to the network fail, the network should be able to detect it, continue working, and if possible, without having to completely shut it down. Instead, the device that has failed should be able to restart again and reconnect to the whole network without problems.

Along this line, it would be desirable the presence of a monitoring process which checks the correct operation of each module and perform notification tasks in case of malfunction. Also, there should be tools to set up and down an arbitrary complex network of processes.

- **Easy to build in**: The middleware should also be easy to extend to match the needs of CEEDS partners to develop the applications. Thus, the creation of new middleware-compliant modules should be intuitive and easy.

1.4.2 Evaluation of existing middleware

As with any tool to choice for a big project, it is important to know the options and alternatives available, in order to select the most suited one. A state-of-the-art study for robot middleware systems was performed. This brief review of robotics middleware shows some of the alternatives analysed. The study was focused on middleware for robotics because of the similarities of the CEEDS proposed architecture and CXIM 2.0 architecture with a robotics system; where many input/output processes must run parallel, with multiple devices such as sensors, actuators and displays being controlled simultaneously, and heavy presence of distributed processing of great amounts of data:

- **YARP**: Yet Another Robotics Platform. This middleware supports building a robot control system as a collection of programs communicating in a peer-to-peer way, with a family of connection types (TCP, UDP, multicast, local, MPI, ...) that can be swapped in and out to match a given network. It also supports similarly flexible interfacing with hardware devices. It is mainly focused on humanoid robots and its main goal is to increase the longevity of robot software projects. YARP is not an operating system for a robot (http://eris.liralab.it/yarp/).
- **Player**: provides a network interface to a variety of actuator and sensor hardware. Player's client/server model allows robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. Player supports multiple concurrent client connections to devices. It is mainly focused in mobile robots and their accessories (http://playerstage.sourceforge.net/).
- **ROS**: Robotics Operating System. ROS is an open-source, meta-operating system for a robot. It provides services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. The ROS runtime "graph" is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication (http://www.ros.org/wiki/).
- **ORCA**: Orca is an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks (http://orca-robotics.sourceforge.net/).
- **Urbi**: The goal of Urbi is to help making robots compatible, and simplify the process of writing programs and behaviours for those robots. Urbi has been successfully used in generic complex systems, where parallel and event-driven orchestration on multiple agents is required (http://www.urbiforge.org/).

Accounting for both the specific features and characteristics of each middleware, and the previous knowledge and expertise of several groups in two of these middleware, reduced the options to ROS and YARP.

1.4.3 Comparison between ROS and YARP

We are going to compare only two of these middleware, YARP and ROS, because they are well established in the robotics community where the key issue is the interaction among distributed systems that manage sensors, visualization devices, database and software components. YARP is a well-known middleware, though technically it is only a set of C++ libraries, as in his long trajectory the developers have kept an emphasis on maximizing interoperability, by not adding components and features that are not strictly necessary and can be packaged as libraries. ROS instead offers a complete framework to work within, with several tools to support and develop complex integration projects. Though this reduces ROS interoperability with other systems, in fact nothing keeps from using YARP technology within the ROS framework, as YARP is packaged as a library within the ROS package management system. A small overview of the comparison between ROS and YARP is shown in Table 1.

Table 1: Comparison between YARP and ROS.

Feature	YARP	ROS
Multi-process & computer Communications	• Yes	• Yes
Multi-platform:		
• **Linux**	• Yes	• Yes
• **Windows**	• Yes	• Partial
• **Mac**	• Yes	• Partial
Communication types:		
• **TCP/UDP**	• Yes	• Yes
• **Multicast**	• Yes	• Yes
• **Shared Memory**	• Yes	• No
Communication protocols:		
• **Publisher/Subscriber**	• Yes	• Yes
• **Server/Client**	• No	• Yes
• **Actions**	• No	• Yes
Communication tools:	• Port / Buffered Port	• Node Handle
Communication messages:	• Bottles	• Services and Messages
Communication robustness:		
• **Execution Order**	• No	• No
• **Disconnection notification**	• Reporter	• Core
• **System restore**	• No	• Partial
Interfaces:	• netWrappers	• SRV and Messages
Community:	• Relatively Small	• Big, growing

From a strictly technical point of view, ROS is a much more powerful development tool, with higher specifications and capabilities, being a complete development framework. This is because of two main factors: YARP's original release dates 2002, while ROS' public release was done by 2008, thus being YARP more than half a decade older than ROS; and while YARP is maintained by an assorted team of developers and collaborators, ROS has a bigger development community, and its developer and user base is growing at a good pace.

Even as ROS poses as a newer, more powerful, and with higher specifications and features middleware, some problems and issues were addressed to assure its usefulness within the CEEDs context. These were mainly two issues: the not easily accessible learning curve and the only partial support into non-Linux based computers.

The first handicap to use ROS within the context of CEEDs is because of the high level of features and tools provided. The experience of the IRI Robotic Lab at the UPC shows that once a learning phase is completed, in which some non-trivial competences on ROS are acquired, most of the developers increase both the productivity and quality of development, as ROS enforces best practice paradigms into its normal utilisation. But this learning phase requires a period of even months, that most of the developers on CEEDs partners cannot allow themselves to take. Thus, several scripts and tools have been developed to simplify enormously the effort and knowledge of the ROS framework needed to start working with publisher, subscribers, clients and servers, and the most usual features. These scripts allows for the work done by and experienced ROS user in some hours configuring features and performing time-consuming tasks, to be done by a relative novel user of ROS in less than an hour.

The other great issue of ROS is the non-native support to systems based upon non-Linux operating systems. This problem can be dealt with in several ways, most of them include copying libraries from Linux based machines, employing specific software to link these libraries and performing some environment configurations that normally are performed automatically by ROS. Though it requires some advanced user knowledge, working with ROS in a Windows based computer proved during the tests being more uncomfortable than difficult. Even though the test proved the feasibility of working ROS on non-Linux based system, it still kept a specially dampening feature. A ROS communication network presents a core node which allows for all the processes and communications to be established and reached. This core node must always run on an Ubuntu based machine, thus imposing a hard restriction on interoperability. Thus, even integration test deploying and using ROS as middleware in Windows & Mac based machines were performed as proof of concept with success; the requirement to deploy an Ubuntu based machine cannot be avoided.

In the end, thought important efforts were done to allow ROS being a desirable option for middleware into the CEEDS project, solving a great deal of its problem (such as automating some of the more harder and time consuming task while developing and integrating code within it), the fact that it lacks enough interoperability to work seamlessly (even after some efforts) on environments without any Linux machine excluded it as an option.

1.5 YARP tool

YARP stands for Yet Another Robot Platform (Fitzpatrick et al. 2008, http://eris.liralab.it/yarp), and is a communication tool between processes. Unlike several other alternatives, it is neither a control system nor OS or a meta OS. It could be labelled as a middleware system, but unlike many of them does not impose constrains of architectures and technologies deployed. In general terms, it is a set of libraries, protocols, and tools to keep modules and devices cleanly decoupled while enabling communication, independently of OS's, protocols and technologies employed. This suits CEEDS, which includes modules from different groups using several OS's and technologies, each one with distinct features and needs in terms of OS and communications.

YARP organizes communication between sensors, processors, and actuators with loose coupling, encouraging gradual system evolution. Its communication model is transport-neutral, so that data flow is decoupled from the details of the underlying networks and protocols in use (allowing several to be used simultaneously). YARP's methodology for interfacing with devices and modules also encourages loose coupling making changes in devices less disruptive.

Development of YARP is done by and for researchers, who find themselves with multiple complex hardware devices to control with an equally complex stack of software. This makes YARP entirely free of licensing costs as it is an open source project. Nowadays, running decent visual, auditory, and tactile perception while performing elaborate response and control in real-time requires high performance computation, being scalability a usual requisite for integration in long-term projects.

Also in CEEDs there is a need for communication between processes, which can be either from one or different groups. To have a good communication an Integration Deliverable Document should incorporate a diagram of each Application Process Graph, which should be detailed explaining the input and output.

The concept of YARP is to send data, a *bottle* from a *port* of *process 1* to a *port* of *process 2.* The process which expects to receive a message will have to *read* the port; the process therefore will have to wait until it receives something. The sending process will send a message through one port to another port. You will have to specify exactly which bottle to send through which port to which other port. The conventions of naming are detailed in the next subsection.

For more details about the installation of YARP see Annex I in the case of installation on Windows and Annex II in the case of installation on Linux based systems. Specific code examples of using YARP are shown in Annexes III and IV. More detail information about YARP can be found in http://eris.liralab.it/yarpdoc/index.html.

1.6 Port Naming Convention

In order reuse as much as possible the software between applications, especially the sensor acquisition and the CXIM 2.0 modules the following port naming convention is *recommended*:

For **publishers**, ports should be named as:

```
/[process_name]/[data_name]/out
```

Example: `/sensorized_tshirt/heart_beat/out`

For **subscribers**:

```
/[process_name]/[data_name]/in
```

Example: `/xim_visualization/3d_pose/in)`

For **servers**, request ports should follow:

```
/[process_name]/request/in
```

Example: `/electrolux_db/request/in)`

The **server** will reply through the port:

```
/[process_name]/reply/out
```

Example:: `/electrolux_db/reply/out`

A **client** process within the network, request a service by using the port:

```
/[process_name]/[server_process_name]/request/out
```

Example: `/compose_visualization/electrolux_db/request/out`

And it will receive the reply to the port:

```
/[process_name]/[server_process_name]/reply/in
```

Example: `/compose_visualization/electrolux_db/reply/in`

In case that a server implements more than one service, the YARP bottle containing request data should indicate which service targets the request by means of a string or a service id. For instance, the first field of a request to “electrolux_db” server could be the string “oven_params”, indicating that the request is targeted to the service related to get some oven parameters from that commercial data base.

2 Computer System Specifications

This section shows the specifications for the computer network and the required software libraries. Secondly it shortly discusses the CXIM 2.0 system.

2.1 Computer Network and Software libraries specifications

A networked computer system will be in charge of implementing the different applications proposed by the CEEDs project. The required features for the computer system and network are listed below:

- Set of N_S sensors: placed on the CEEDs user or in the CEEDs environment
- Set of N_P computers to do sensor acquisition, cue processing and data discovery
- Set of N_X computers to do the visualization and actuation
- Set of actuators and displays (CXIM 2.0 installation)
- LAN/WLAN router
- OS: Linux (Debian, Ubuntu 10.10+), MAC OSX, Windows XP/7
- Common software and libraries: ACE 5.6+ and YARP 2.3+
- C++ builder to compile and build ACE (Borland, MS VC++, gcc or others)
- Cross platform build system: CMAKE 2.8+

The layout of the elements is shown in Figure 3.

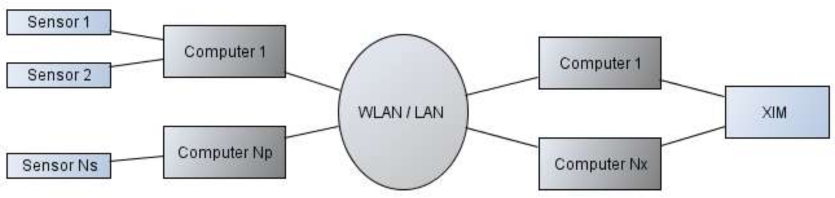


Figure 3: Connection layout of the computer systems in the CEEDS environment.

Optionally for the portable CEEDs sytem, if some application requires Internet access, an extra computer can be connected to the CEEDs LAN to play the role of a proxy, so that the required data can be requested through this proxy from the Internet. This situation however can introduce large delays and requires the implementation of an extra software process that translates and sends messages over the Internet.

2.2 CXIM 2.0

XIM stands for eXperience Induction Machine and contains several servers, sensors, effectors and a network. The second version, as used in the CEEDs project is the CXIM 2.0, the CEEDs eXperience Induction Machine 2.0, an upgraded version of the XIM. In order to control and operate all the sensors and displays of the CXIM 2.0 environment (see Deliverable D4.1), a network of multiple servers is deployed, connected through YARP as can be seen in Figure 4. This network previously worked on a basis of a machine per server, but thanks to updates performed on CXIM 2.0 within the scope of CEEDs project, most of the servers can run on a reduced number of machines (using Virtual machines).

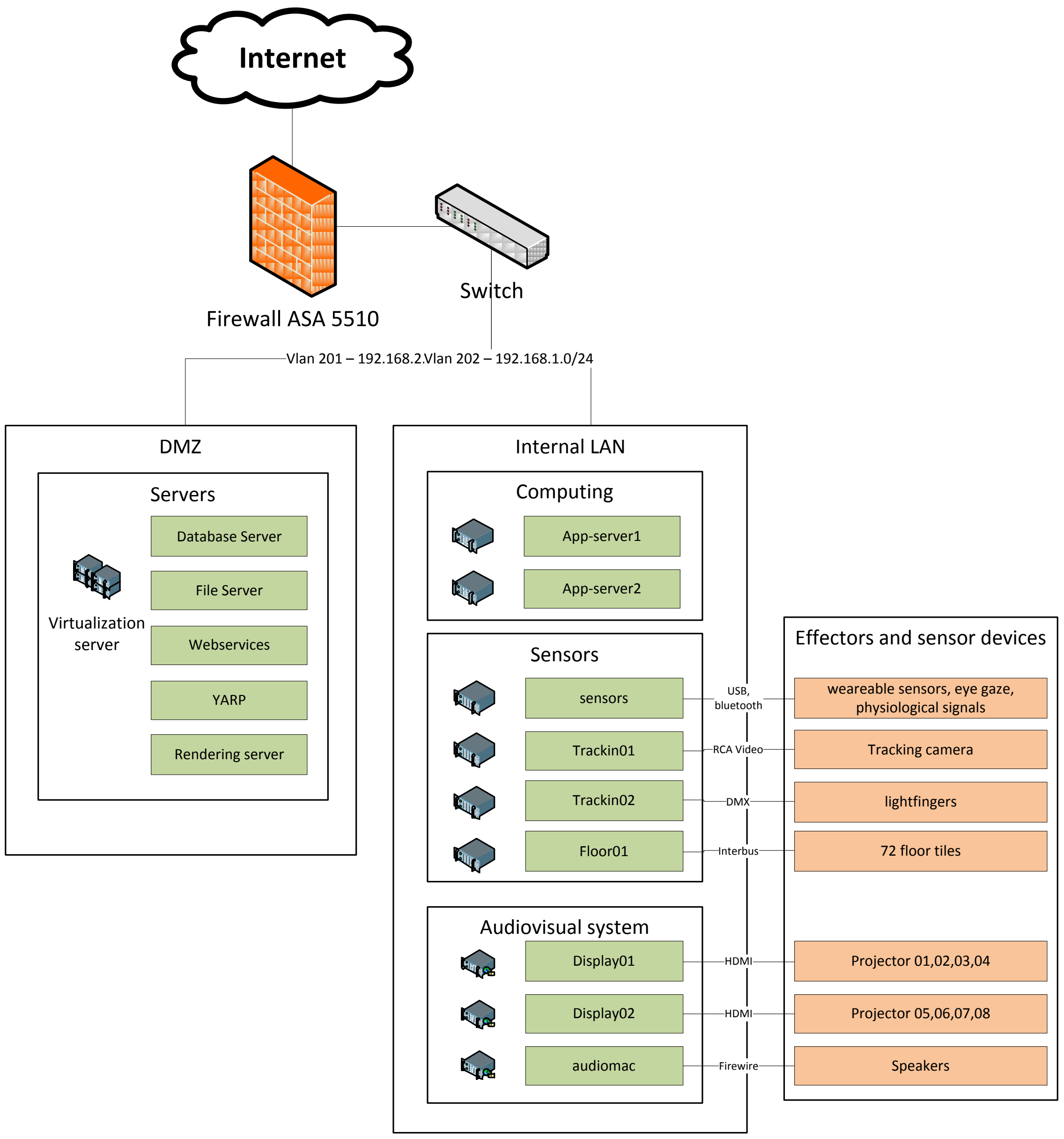


Figure 4: The CXIM 2.0 network architecture.

3 Integration Guidelines through the Neuroscience Application

This section explains the integration process of a CEEDs application and the way of preparing it to use with processes of other CEEDs partners. In this section we use the Neuroscience application as example.

3.1 Application Overview

The main goal of the Neuroscience application is to allow a user to visualize a neuronal model and interact with it. It is based on "iqr" and on the CXIM 2.0 infrastructure. iqr is a multi-level neuronal simulation environment which allows designing complex neuronal models graphically, and to visualize and analyse their properties on-line. It has a modular expandable multi-platform architecture, and it is publicly accessible under the GNU General Public License (GPL). iqr provides a 2D graphical interface to design and manipulate the neuronal model. While the simulation is running, the user can visualize internal states and change the parameters of system elements (see deliverable D6.1 for full description of the application).

3.2 Process Diagram

Like explained before in sections 1.2 and 1.3 it is important to define the process diagram where you show the different processes and their interaction. In Figure 5 the process diagram for the neuroscience application is shown.

The schema shows three groups of processes and the user interacting with them:

1 *User Sensors*: These are the systems which monitor the user for any implicit or explicit cues. Each hardware element has an acquiring process which receives the signal from the hardware and makes it available for other processes in the network (for detailed specification see D2.1). The explicit commands are given through the WiiMode. Furthermore there are some processes (such as *Tracking*) that process data from several hardware sensors.

2 *Presentation*: Unity is used to do the 3D model visualization and iqr is used to run the simulation of the model, i.e. the interaction of the neurons. The hardware parts are the screens (or beamers which project onto the screen), the floor with lights in the tiles and the sound speakers.

3 *CEEDs Engine*: This is where the sentient agent should use the implicit and explicit cues of the user to form a model of the user and to learn to use these cues to improve the discovery in the data.

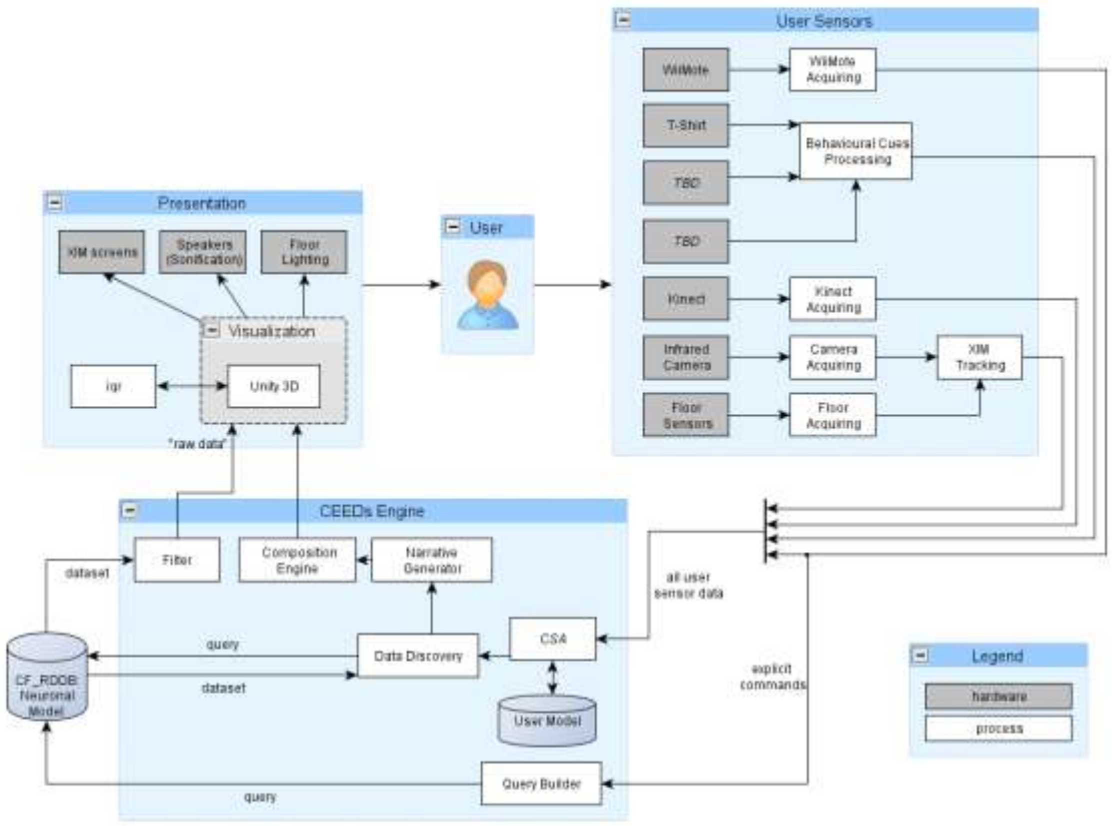


Figure 5: the processes and hardware within the Neuroscience application.

3.3 Process Specifications

This section provides, up to the current definition, the specifications of some processes involved in the Neuroscience Application as shown in Figure 5. These specifications help to (1) identify what is missing in terms of process definition, (2) support application management and debugging tasks, and (3) to notify to partners of other applications which processes are being developed in order to maximize software reuse.

The following of the section is a list of "process cards". These cards are intended to be a working material, so during the development of the project, partners will update them as new specifications or ideas will be fixed in terms of process implementation. At the current point, when a feature is not defined or specified, it will be labelled with "TBD" standing for "To Be Defined".

Process Name:	KINECT ACQUIRING
Partner in Charge:	UPF
Goal:	Real-time data acquiring and publication of the Kinect data.
Input:	Kinect (HW)
Output:	YARP Bottle: Set of 3D body skeleton points
Test Methodology:	Qualitative Evaluation through Single User Movement Tests.

Machine:	CXIM - sensor
OS:	Windows
Required Libraries:	TBD

Process Name:	XIM TRACKING
Partner in Charge:	UPF
Goal:	Real-time tracking of the 2D position of the user within the XIM frame.
Input:	XIM - IR Camera image data. XIM-Floor Pressure data.
Output:	YARP Bottle, Vector of "number of users" size. For each user, (x,y) position with respect to the XIM frame and Id integer.
Test Methodology:	Qualitative Evaluation through Single User Movement Tests
Machine:	CXIM - Tracking01
OS:	Linux
Required Libraries:	TBD

Process Name:	WiiMOTE ACQUIRING
Partner in Charge:	UPF
Goal:	Real-time acquiring and publication of WiiMote data.
Input:	WiiMote (HW).
Output:	YARP Bottle: TBD.
Test Methodology:	Qualitative Evaluation through Single User Movement Tests
Machine:	CXIM - sensor
OS:	Linux
Required Libraries:	TBD

Process Name:	BEHAVIORAL CUES PROCESSING
Partner in Charge:	UNIPD, UAU
Goal:	Real-time estimation of a set of behavioural cues (implicit cues).
Input:	Sensorized T-shirt (breath and heart beat) + Other Sensory data TBD
Output:	YARP Bottle TBD: Desired would be: Arousal, cognitive payload, Eye tracking, Gaze tracking.
Test Methodology:	Qualitative Evaluation through Single User Movement Tests
Machine:	CXIM - sensor
OS:	Linux
Required Libraries:	TBD

Process Name:	CEEDs SENTIENT AGENT (CSA)

Partner in Charge:	UPF
Goal:	Real-time implementation of the CEEDs Agent
Input:	User Model (File TBD), User Tracking, Kinect 3D body points, behavioral cues (TBD).
Output:	TBD
Test Methodology:	TBD
Machine:	CXIM - App-server
OS:	TBD
Required Libraries:	TBD

Process Name:	DATA DISCOVERY
Partner in Charge:	MPG
Goal:	Real-time support for data discovery in the Raw Database.
Input:	RDDB (XML File), CSA (TBD)
Output:	Salient Data (TBD)
Test Methodology:	TBD
Machine:	CXIM - App-server
OS:	TBD
Required Libraries:	TBD

Process Name:	NARRATIVE GENERATOR
Partner in Charge:	TEESSIDE
Goal:	Real-time generation of a narrative route/script to support data displaying
Input:	Salient Data
Output:	Narrative Script, TBD
Test Methodology:	TBD
Machine:	CXIM - App-server
OS:	TBD
Required Libraries:	TBD

Process Name:	COMPOSITION ENGINE
Partner in Charge:	UPF
Goal:	Real-time decisions about how and which data has to be displayed (XIM-screens, XIM-speakers, XIM-floor lights)
Input:	narrative script
Output:	TBD
Test Methodology:	TBD
Machine:	CXIM - App-server
OS:	TBD

Required Libraries:	TBD

Process Name:	VISUALIZATION
Partner in Charge:	TBD
Goal:	Real-time renderings and sound generation. Interface with XIM HW
Input:	TBD
Output:	TBD
Test Methodology:	Qualitative evaluation of rendering/sonorization/illumination of static data
Machine:	XIM-display0X
OS:	Windows
Required Libraries:	TBD

Process Name:	iqr
Partner in Charge:	UPF
Goal:	Real-time simulation of a neural model
Input:	Neuro Model (XML File). User Configs (2D GUI)
Output:	Dynamics of the Neuro Model
Test Methodology:	TBD
Machine:	XIM - iqr-control
OS:	Linux
Required Libraries:	TBD

Each of the process specifications give not only the details of the goal of the processes as shown in Figure 5, but also the data sent between them as shown with lines in the Figure 5, and in the input / output fields in the "process cards".

The next step is to detail the input and output: the types, size and the rate. For an input the required rate should be given and for the output a minimum and maximum rate of sending data. These details are important to specify such that can be seen in advance if the requirements of an input match the physical possible range of the output of the other process.

We can distinguish on YARP connections and other connections. An 'other' connection can be for example the *Acquiring* processes which are connected to the hardware. If we take the *WiiMote Acquiring* process, this will run on a machine which is physically connected to the WiiMote (or in this case wireless). The communication of the process to the WiiMote is through a specific driver. These type of connections are mostly within a (work) package and therefore should be well documented, but are not at most importance for other work packages.

More important for other work packages are the definitions of the input / output of processes that respectively receive / send, from / to other processes. As an example we can take the *XIM Tracking* process that receives input from the *Camera Acquiring* process and the *Floor Acquiring* process. As output it should send a list of positions of all the tracked persons in the CXIM. The specification of the YARP bottle (see section 1.5 for more details about YARP bottles) is:

- Main Bottle, which contains *n* sub bottles, where *n* is the number of tracked persons. The sub bottle contains:
  - *x*: float [4 bytes]
  - y: float [4 bytes]
  - id: int [4 bytes]

The size of 1 bottle is thus *n(4+4+4)* bytes = *12 n* bytes. The following step is to look at the send and receive rate, which depend on the type of inter process communication: client / server or publisher / subscriber, see for more information sections 1.2, 1.3 and annexes III and IV. In the case of publisher / subscriber the publisher will decide the rate of sending data, in the case of server / client the client will decide this rate. Obviously in all these cases the rate is limited by the maximum capacity of the hardware (processor, memory, etc.), software (OS, middleware, etc.) and network. The middleware, YARP in our case, also handles the problem of multiple processes requesting and/or sending data.

Annex I: YARP Installation Guide for Windows

This reference guide will briefly detail installation, building and configuration of the needed libraries to use YARP on a MS Windows based environment. Though it is recommended the utilisation of the latest stable releases of the presented software, we list the older versions tested.

Required software

The following software is required:

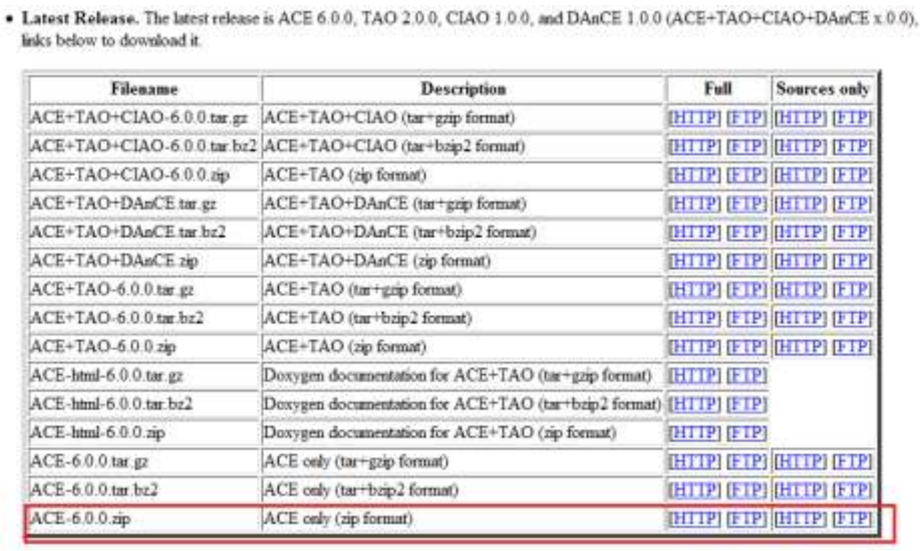
- MS Windows XP or later (Windows 7 recommended)
- Cmake 2.8 or higher version
- ACE 5.6 or higher version
- YARP 2.3 or higher version
- A suitable builder for Cmake*
- A suitable builder to build ACE*

*For this guide, MS Visual C++ 9 2008 will be considered, as it licensed for free through the MS Visual Studio 2008 Express.

ACE installation and building with MS Visual Studio 9

While CMake liberates your project from the particular development environment you are using, the code itself may still have operating-system dependencies that will make cross-platform work difficult. ACE is a free and open source library that provides a portable interface to a vast array of operating systems. You can download it at:

http://download.dre.vanderbilt.edu/



- **Latest Release.** The latest release is ACE 6.0.0, TAO 2.0.0, CIAO 1.0.0, and DAnCE 1.0.0 (ACE+TAO+CIAO+DAnCE x.0.0), links below to download it.

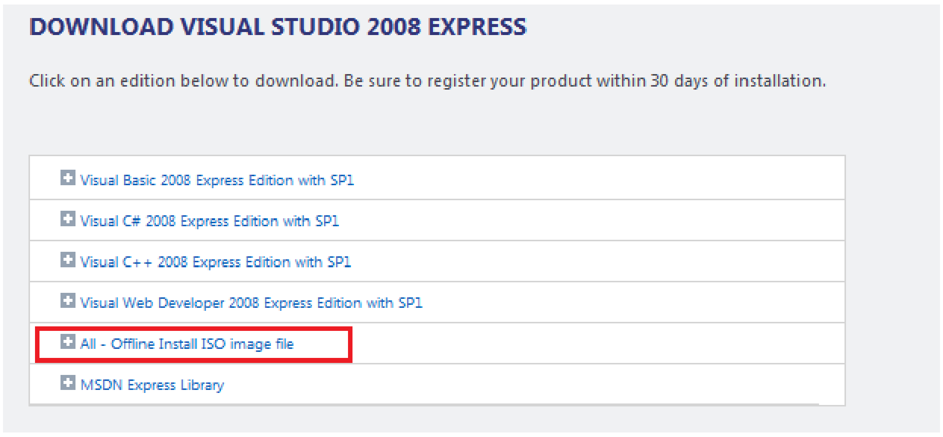
Filename	Description	Full	Sources only
ACE+TAO+CIAO-6.0.0.tar.gz	ACE+TAO+CIAO (tar+gzip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO+CIAO-6.0.0.tar.bz2	ACE+TAO+CIAO (tar+bzip2 format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO+CIAO-6.0.0.zip	ACE+TAO (zip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO+DAnCE.tar.gz	ACE+TAO+DAnCE (tar+gzip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO+DAnCE.tar.bz2	ACE+TAO+DAnCE (tar+bzip2 format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO+DAnCE.zip	ACE+TAO+DAnCE (zip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO-6.0.0.tar.gz	ACE+TAO (tar+gzip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO-6.0.0.tar.bz2	ACE+TAO (tar+bzip2 format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE+TAO-6.0.0.zip	ACE+TAO (zip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE-html-6.0.0.tar.gz	Doxygen documentation for ACE+TAO (tar+gzip format)	[HTTP] [FTP]	
ACE-html-6.0.0.tar.bz2	Doxygen documentation for ACE+TAO (tar+bzip2 format)	[HTTP] [FTP]	
ACE-html-6.0.0.zip	Doxygen documentation for ACE+TAO (zip format)	[HTTP] [FTP]	
ACE-6.0.0.tar.gz	ACE only (tar+gzip format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE-6.0.0.tar.bz2	ACE only (tar+bzip2 format)	[HTTP] [FTP]	[HTTP] [FTP]
ACE-6.0.0.zip	ACE only (zip format)	[HTTP] [FTP]	[HTTP] [FTP]

For this guide we will suppose that MS Visual Studio 9 (or newer) is installed and used to build it, thought you can obtain instruction to several other builder, through the following links:

- Microsoft Visual Studio
- Embarcadero C++Builder
- MinGW
- Cygwin

If you do not have MS Visual Studio 7.1+ installed, you can obtain MS Visual Studio 9 freely at:

http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express:

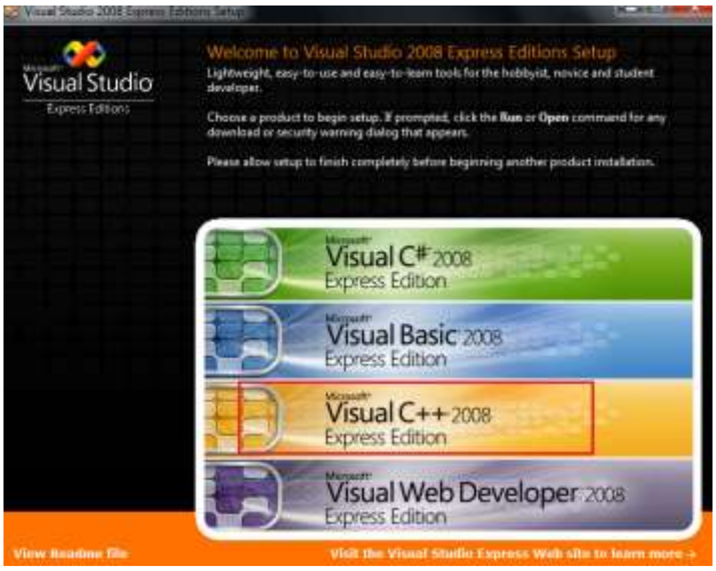


DOWNLOAD VISUAL STUDIO 2008 EXPRESS

Click on an edition below to download. Be sure to register your product within 30 days of installation.

- Visual Basic 2008 Express Edition with SP1
- Visual C# 2008 Express Edition with SP1
- Visual C++ 2008 Express Edition with SP1
- Visual Web Developer 2008 Express Edition with SP1
- All - Offline Install ISO image file
- MSDN Express Library

where you can obtain a full version image to burn in to a CD (recommended option).



Visual Studio 2008 Express Editions Setup

Welcome to Visual Studio 2008 Express Editions Setup

Lightweight, easy-to-use and easy-to-learn tools for the hobbyist, novice and student developer.

Choose a product to begin setup. If prompted, click the **Run** or **Open** command for any download or security warning dialog that appears.

Please allow setup to finish completely before beginning another product installation.

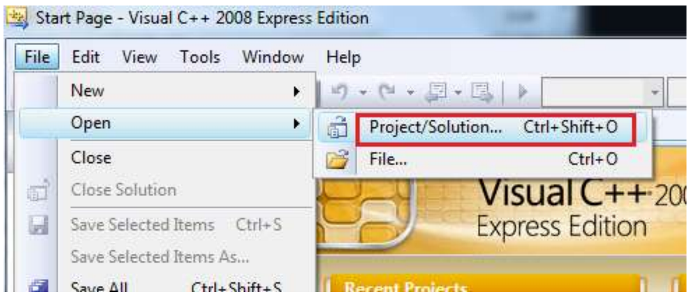
Microsoft Visual Studio Express Editions
Microsoft Visual C# 2008 Express Edition
Microsoft Visual Basic 2008 Express Edition
Microsoft Visual C++ 2008 Express Edition
Microsoft Visual Web Developer 2008 Express Edition
View Readme file
Visit the Visual Studio Express Web site to learn more →

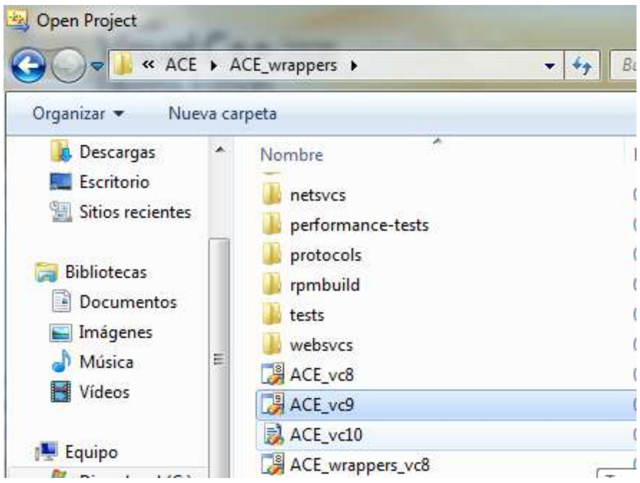
Once installation of Visual C++ is completed, ACE must be uncompressed into a directory, where it will create a ACE_wrappers directory containing the distribution. The ACE_wrappers directory will be referred to as ACE_ROOT in the following steps. We assume here that you have installed the free Visual Studio 9, other versions we have not tried.

After uncompressing you should create a file called `config.h` in the ACE_ROOT\ace directory that contains the line:

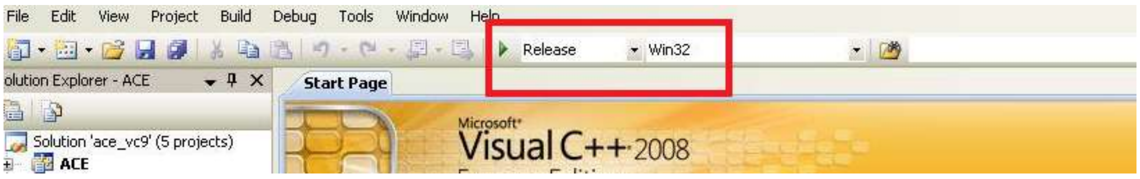
```
#include "ace/config-win32.h"
```

Then load the solution file for ACE (ACE_ROOT/ACE_vc9.sln for our case):





Once loaded, build both release and debug configurations. All the different configurations are provided for your convenience. You can either adopt the scheme to build your applications with different configurations, or use `ace/config.h` to tweak with the default settings on NT. At any case it is advised that you build both cited configurations.



In the end, you should have some files in the directory ACE_wrappers/lib, including ACE.dll, ACE.lib, ACEd.dll, ACEd.lib (if you compiled both release and debug versions).

You must make sure you include the path to "ACE_wrappers\lib" in your PATH environment variable whenever you run programs that use ACE (such as YARP). It is also a good idea to set an environment variable ACE_ROOT to hold the path to ACE_wrappers. This will make it easier to find the libraries and header files. You will find a small how-to set environment variables later on in this annex.

Obtaining YARP

Just get the latest stable YARP release at:

http://eris.liralab.it/yarpdoc/download.html

Your Yet Another Robot Platform download will start in 0 seconds...

Problems with the download? Please use this direct link, or try another mirror.

And get Cmake to build YARP.

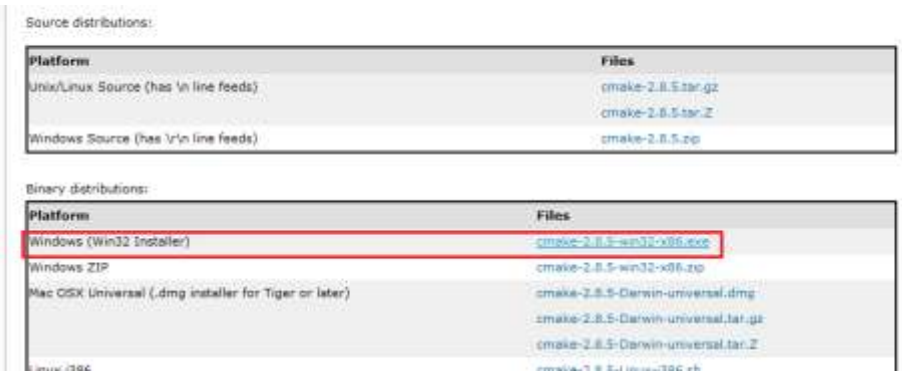
Cmake Installation and building YARP

CMake is a family of tools designed to build, test and package software. CMake can be used with the compiler environment of your choice, making cross platform working easier, thanks to the use of platform independent makefiles and workspaces.

To install Cmake download the desired version, in our case the binary Installer, at

http://www.cmake.org/HTML/Download.html

And execute installer following the instructions.



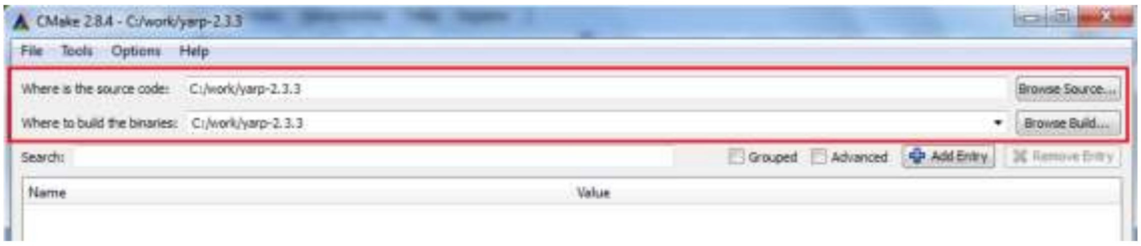
Source distributions:

Platform	Files
Unix/Linux Source (has \n line feeds)	cmake-2.8.5.tar.gz
	cmake-2.8.5.tar.Z
Windows Source (has \r\n line feeds)	cmake-2.8.5.zip

Binary distributions:

Platform	Files
Windows (Win32 Installer)	cmake-2.8.5-win32-x86.exe
Windows ZIP	cmake-2.8.5-win32-x86.zip
Mac OSX Universal (.dmg installer for Tiger or later)	cmake-2.8.5-Darwin-universal.dmg
	cmake-2.8.5-Darwin-universal.tar.gz
	cmake-2.8.5-Darwin-universal.tar.Z

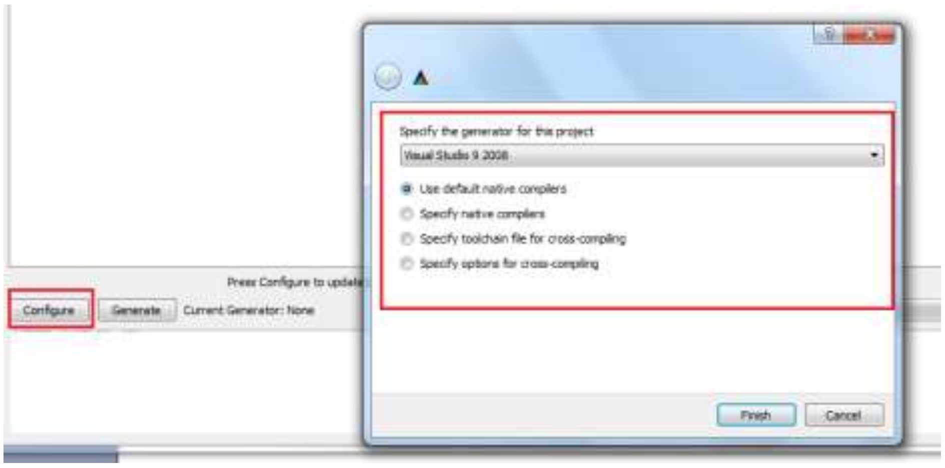
Windows' versions of CMake dispose of a user friendly GUI. After installing, you should have an icon for CMake in your START menu. Click that, and then fill in the path to the code to work with, in this case to the YARP version downloaded, and the path where CMake shall build in YARP.



Where is the source code: C:/work/yarp-2.3.3
Where to build the binaries: C:/work/yarp-2.3.3
Search:

Name	Value

Click "configure". For this case we will work with the same generator used for ACE, MS Visual Studio 9, but you can choose any option you feel more suitable for your needs. When the "OK" button becomes clickable, then CMake has enough information to set up your project. Click "OK" and you're done. Project files of the type you specified should exist in the build path you gave.

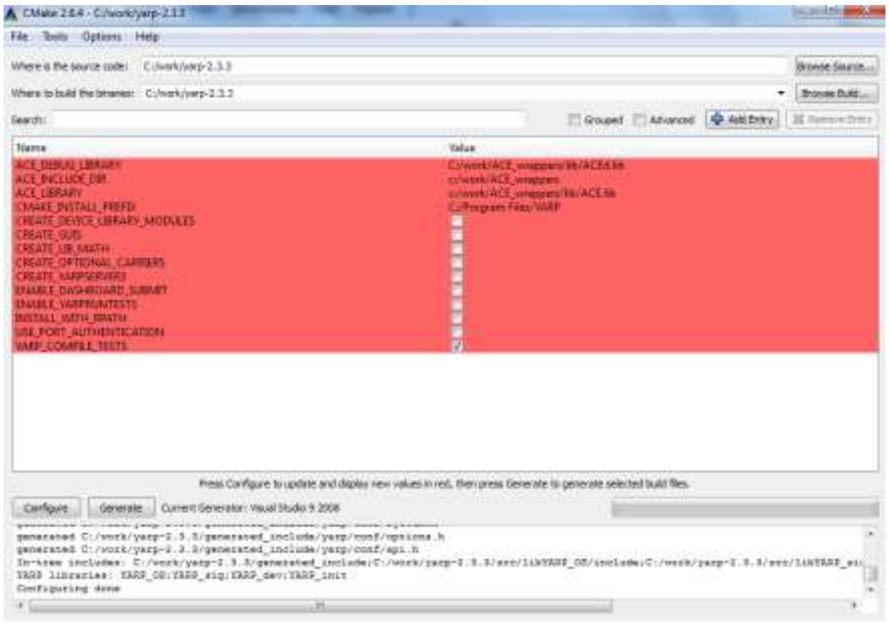


Press Configure to update

Current Generator: None

Specify the generator for this project
Visual Studio 9 2008
[✓ Use default native compilers]
Specify native compilers
Specify toolchain file for cross-compiling
Specify options for cross-compiling

After the configuration is done, you should have a list to choose which properties you want to set up:



Where is the source code: C:/work/yarp-2.3.3
Where to build the binaries: C:/work/yarp-2.3.3
Search:

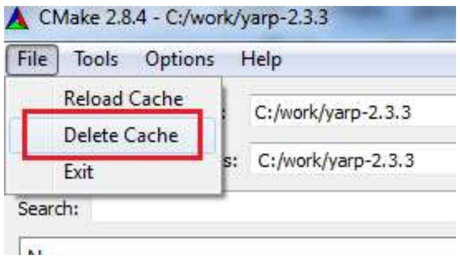
Name	Value
ACE_DEBUG_LIBRARY	C:/work/ACE_wrappers/lib/ACEd.lib
ACE_INCLUDE_DIR	c:/work/ACE_wrappers
ACE_LIBRARY	c:/work/ACE_wrappers/lib/ACE.lib
CMAKE_INSTALL_PREFIX	C:/Program Files/YARP
CREATE_DEVICE_LIBRARY_MODULES	☐
CREATE_GUIS	☐
CREATE_LIB_MATH	☐
CREATE_OPTIONAL_CARRIERS	☐
CREATE_YARPSERVER3	☐
ENABLE_DASHBOARD_SUBMIT	☐
ENABLE_YARPRUNTESTS	☐
INSTALL_WITH_RPATH	☐
USE_PORT_AUTHENTICATION	☐
YARP_COMPILE_TESTS	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
generated C:/work/yarp-2.3.3/generated_include/yarp/conf/options.h
generated C:/work/yarp-2.3.3/generated_include/yarp/conf/api.h
In-tree includes: C:/work/yarp-2.3.3/generated_include;C:/work/yarp-2.3.3/src/libYARP_OS/include;C:/work/yarp-2.3.3/src/libYARP_si
YARP libraries: YARP_OS;YARP_sig;YARP_dev;YARP_init
Configuring done
```

After checking the desired options, press Generate Button. So, YARP is ready to work on your Windows based machine. For last point it is work noting that shall you want to start over from the beginning, you can delete all done through the Delete Cache Button:



Now you should open the generated Visual Studio project file: ALL_BUILD.vcxproj

Which has been generated in the YARP directory, see 0.

Compile and test YARP

Now we should compile YARP with the generated project file (in case of Visual Studio). Go to the directory where you unzipped YARP and open the project file generated with CMake: ALL_BUILD.vcxproj.

Build the project in Release and Debug, this generates the executable files for the YARP system in <YARP>\bin\RELEASE and the library files in <YARP>\lib\RELEASE where <YARP> is the directory of YARP.

To be able to run YARP in all directories you can the previously mentioned it to the path. Now we have YARP installed and compiled we can test it. Open a command prompt (run: `cmd`) and type:

```
yarpserver
```

The first time Windows might show a firewall warning, just accept it. After this open a new command window and type:

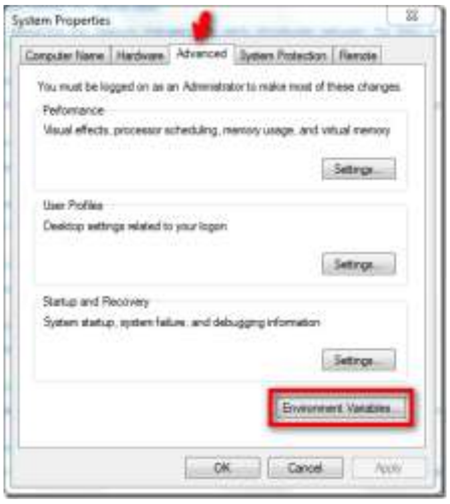
```
yarp check
```

This also might show a warning from the Windows firewall, accept it and then if everything is ok it should show in the end:

```
yarp: *** YARP seems okay
```

Setting Environment Variables in Windows

Find the environment variables window. You should go though Control Panel => System => Advanced tab and push Environment Variables.


System Properties

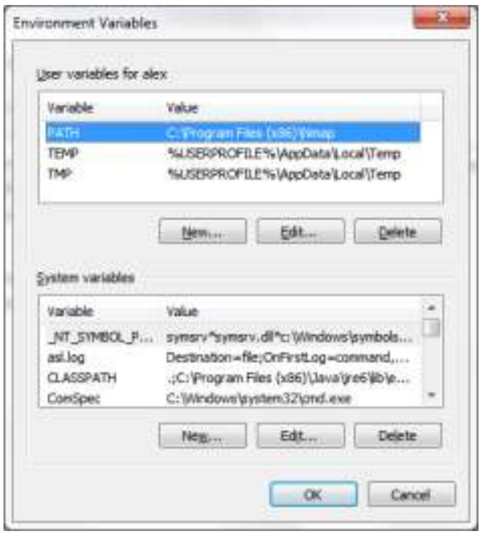
You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

There, you can add or edit the values of variables. Remember that when you add a path to an environment variable you must separate it from the previous with a semicolon.


Environment Variables

User variables for alex

Variable	Value
PATH	C:\Program Files (x86)\Nmap
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
_NT_SYMBOL_P...	symsrv*symsrv.dll*c:\Windows\symbols...
asl.log	Destination=file;OnFirstLog=command,...
CLASSPATH	.;C:\Program Files (x86)\Java\jre6\lib\e...
ComSpec	C:\Windows\system32\cmd.exe

Change YARP server

When you want to specify the YARP server to use, you can edit the YARP configuration file. You can find this file with the following command:

```
yarp conf
```

The first line of this file is the IP and port of the YARP server to use, for example:

```
192.168.100.220 10000
```

Annex II: YARP Installation guide for Linux

This guide contains two sections dealing with installation of YARP libraries and required components on GNU/Linux based systems, though it can be extrapolated to similar architectures. At any case, for dealing with troubles and special cases some references to sources are introduced. The first section will introduce a really quick installation on 5 steps guide for Ubuntu and similar distributions. The later will explain the more general installation for other Linux distributions.

YARP installation into Ubuntu

Just introduce into your command line:

- `sudo apt-get install cmake libace-dev subversion`
- `svn co https://yarp0.svn.sourceforge.net/svnroot/yarp0/trunk/yarp2 yarp2`
- `cd yarp2; mkdir build; cd build; cmake ..`
- `make`
- `sudo make install`

And then to check:

- `yarp server`

And in a separate window:

- `yarp check`

let yarp perform an automatic test. It should end telling something like:

- `yarp: *** YARP seems okay`

These two later steps just test that your YARP installation is able to run the server and establish communication. If you have any problem you can read more about the YARP installation for Ubuntu at:

http://eris.liralab.it/yarpdoc/install.html

To change the YARP server see the last section of Annex II.

Non Debian based Linux

Before you can install YARP you need to install CMake and ACE as is discussed below.

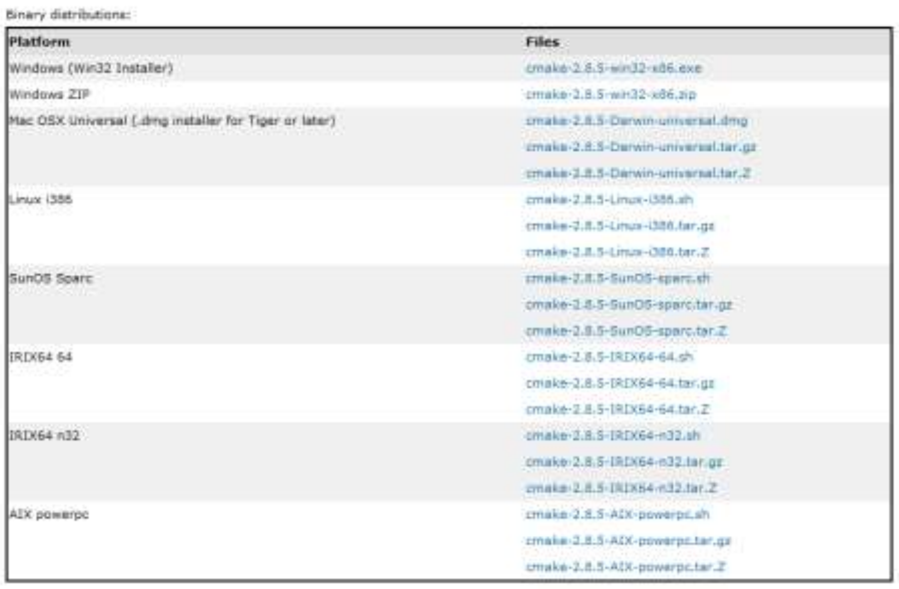
CMake Installation

For Debian and Debian-based distributions, the best approach is command-line:

- `apt-get install Cmake`

For SUSE Linux distributions you can add the GURU YAST repository and use YAST to install Cmake or get it from the GURU website.

Additionally, you can find pre-compiled binaries for Linux and other UNIX based systems at http://www.cmake.org/cmake/resources/software.html.



Binary distributions:

Platform	Files
Windows (Win32 Installer)	cmake-2.8.5-win32-x86.exe
Windows ZIP	cmake-2.8.5-win32-x86.zip
Mac OSX Universal (.dmg installer for Tiger or later)	cmake-2.8.5-Darwin-universal.dmg
	cmake-2.8.5-Darwin-universal.tar.gz
	cmake-2.8.5-Darwin-universal.tar.Z
Linux i386	cmake-2.8.5-Linux-i386.sh
	cmake-2.8.5-Linux-i386.tar.gz
	cmake-2.8.5-Linux-i386.tar.Z
SunOS Sparc	cmake-2.8.5-SunOS-sparc.sh
	cmake-2.8.5-SunOS-sparc.tar.gz
	cmake-2.8.5-SunOS-sparc.tar.Z
IRIX64 64	cmake-2.8.5-IRIX64-64.sh
	cmake-2.8.5-IRIX64-64.tar.gz
	cmake-2.8.5-IRIX64-64.tar.Z
IRIX64 n32	cmake-2.8.5-IRIX64-n32.sh
	cmake-2.8.5-IRIX64-n32.tar.gz
	cmake-2.8.5-IRIX64-n32.tar.Z
AIX powerpc	cmake-2.8.5-AIX-powerpc.sh
	cmake-2.8.5-AIX-powerpc.tar.gz
	cmake-2.8.5-AIX-powerpc.tar.Z

Or you can also download the source files to compile and build them, following instructions and guidance found at http://www.cmake.org/cmake/help/install.html.

ACE installation

If your system distribution has an ACE package you shall get it through the package manager (this option is enable for Debian-based distributions, Redhat, etc…). Look for libace-dev or libace-devel.

In other cases you can obtain source files from http://download.dre.vanderbilt.edu/, and find instructions and guidance at http://www.dre.vanderbilt.edu/~schmidt/DOC_ROOT/ACE/ACE-INSTALL.html#unix_traditional

Though this installation will require some more effort.

Installing YARP

Once Cmake and ACE are both installed into your system, you can obtain YARP at http://yarp0.sourceforge.net/

Your Yet Another Robot Platform download will start in 4 seconds...

Problems with the download? Please use this direct link, or try another mirror.

or installed from SVN by following the instructions at:

http://sourceforge.net/svn/?group_id=62418

with module name "yarp2" to get the code.

To check the YARP installation see the previous section.

Annex III: YARP Publisher / Subscriber Pseudo-code

A simple publisher is in charge of opening and naming ports. Thereafter, it enters its main processing loop, and once its job is finished, it creates a data container, called Bottle in YARP, it sets the container with the data of interest to be published and finally it sends the container through the port. Out of the loop, before finishing, the process should close the port. A pseudo code for a simple publisher is as follows:

```
initYARP();
BufferedPort<Bottle> myPort;
myPort.open("/publisher/output");
while (running)
{
       myData = myProcess();
       Bottle & myBottle = myPort.prepare();
       myBottle = fillTheBottle( myData );
       myPort.write();
       sleep(); //relax output rate if necessary
}
myPort.close();
```

At the receiver side, a simple **subscriber**, synchronized to data reception, follows the pseudo code presented below:

```
initYARP();
BufferedPort<Bottle> myPort;
myPort.open("/subscriber/input");
Network::connect("/publisher/output", myPort.getName(), "tcp");
while (running)
{
       Bottle * myBottle = myPort.blockingRead();
       myData = getFromBottle(myBottle);
       //do something with myData

}
myPort.close();
```

The subscriber also creates and names a port and it connects it to another existing port that provides the input data. Inside the main loop, it is blocked, waiting for data coming from the input port. When data arrives, the process unblocks and it can retrieve the data from the input container, again a YARP Bottle. Once the data is retrieved, it can do any kind of job with the received data. Out of the loop, before finishing, the process should close the port.

Annex IV: YARP Client / Server Pseudo-code

For the client / server interaction mode, the server opens and names two ports, one to receive requests, and the other to send replies. Inside the main loop, it is blocked waiting for a request. Once a request arrives, it gets the request data, does the processing, and creates a new bottle in the reply port. It finally fills the reply bottle with the reply data, and sends that bottle through the reply port. Out of the main loop, before finishing, the server should close all the opened ports.

```
initYARP();
BufferedPort<Bottle> requestPort, replyPort;
requestPort.open("/server/request");
replyPort.open("/server/reply");

while (running)
{
       Bottle * requestBottle = requestPort.blockingRead();
       requestData = getFromBottle(requestBottle);
       replyData = myProcess(requestData);

       Bottle & replyBottle = replyPort.prepare();
       replyBottle = fillTheBottle(replyData);
       replyPort.write();
}
requestPort.close();
replyPort.close();
```

The **client** side is the counter part of the server, so that the client first creates a request bottle and sends it through a request port, and then it blocks to receive the server reply. Please note that the client is in charge of making the connections with the server. The pseudo code for a simple client follows the outline presented below:

```
initYARP();
BufferedPort<Bottle> requestPort, replyPort;
requestPort.open("/client/request");
replyPort.open("/client/reply");
Network::connect(requestPort.getName()
                          , "/server/request", "tcp");
Network::connect("/server/reply", replyPort.getName(), "tcp");
while (running)
{
       Bottle & requestBottle = requestPort.prepare();
       requestBottle = fillTheBottle(requestData);
       requestPort.write();

       Bottle * replyBottle = replyPort.blockingRead();
       replyData = getFromBottle(replyBottle);
       //do something with replyData
}
requestPort.close();
replyPort.close();
```

Annex V: Empathic Human Robot Interaction in the Hide & Seek Game

The purpose of this experiment is to develop a testbed for the integration of the CEEDS applications. In particular, the experiment will handle the integration of the CXIM 2.0 system through the inter-process communication tool YARP.

In this application not only the CXIM 2.0 hardware will be used, but also different sensory systems of partners to be able to detect implicit and explicit cues of the user. It will serve as a framework and a list of key rules to have the other partner's software and hardware modules.

The test example: Empathic Human Robot Interaction in the Hide & Seek Game

This application has two objectives: 1) to improve the integration of the different applications with the XIM system, and 2) to use not only explicit commands to control and interact with the robot, but also to learn from and use implicit cues of the user while he/she is manipulating the robot.

The application will use the XIM and the available user sensors which are developed by the partners. Furthermore the problem for our application will be more difficult because of a real-time data stream which is required to the XIM as will be explained later on.

The aim of the experiment is to enable a human to learn and discover human-robot interaction strategies through his/her embodiment into a robot using the XIM machine. The human will perceive and react to real-time stimuli present at the robot site.

A hide-and-seek situation will be solved in which other humans hide from the robot at the experimental site, and the user in the XIM must find them. With respect to the CEEDs paradigm, the experiment poses relevant challenges for real-time data representation through a limited communications channel, and for learning and categorization of human behaviour from multi-sensory data patterns.

Description of the test example

The robot will play the hide and seek game against one or more opponents. It will act as the seeker in the game that will take place in an urban environment. The seeker wins the game if it recognizes and catches (or approaches sufficiently close) the hider within the allowed time frame, the hider wins if it reaches the base within the game time, otherwise it is a draw.

A human user will control the robot from the XIM system at a high level, thus he/she will play the seeker through the 'eyes' of the robot, i.e. it will see through the sensors of the robot (camera, laser, model, etc.). Eventually, on later stages of development, not only the explicit user control, but also the implicit reactions of the user (such as heart and breath rate, poses, etc.) will be used to 1) learn robot motion behaviours and redefine them to adapt to human activity; 2) the human will discover rules of engagement for an optimal completion of a task from its immersion into a multisensory experience that otherwise would not be possible

The robot to be used (Figure 6) is the result of the URUS (Ubiquitous networking Robotics in Urban Settings) project (Sanfeliu & Andrade-Cetto 2006; Trulls et al. 2011; Sanfeliu et al. 2010), which had as goal to navigate within an urban environments and to interact with people and to guide them. This project will be the base, because it not only had as result the robot, but also the lower level of control which allows it to autonomously navigate in the urban environment.

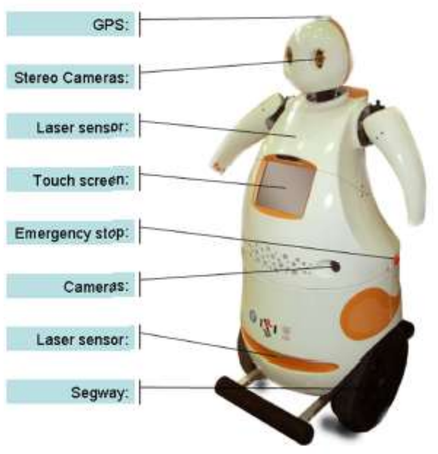


Figure 6: The robot Tibi.

As a first step we will use the XIM system to have the user of the system control the robot remotely in an explicit way as shown in Figure 7. The user can see 'through' the sensors of the robot, i.e. it can see the image of the camera, the laser data, and information about the localization and navigation algorithms. All the sensory and processed information from the robot will have to be sent real-time to the XIM, this will be a technical challenge.

In a second step, a game will be done where the user has to find a person with a bright shirt and a hat (for example yellow) in a urban environment with the presence of other pedestrians. Thus, the environment will contain a great amount of objects and people. In this data the user has to find a specific person through data features which can distinguish the person from the rest of the people. The data containing these features can be described as the salient data. The goal is to find a relation of the (implicit) reactions of the user to the data which is shown to the user in which the person is found. These implicit user responses have to be fed to the CEEDs Sentient Agent; which in its turn should act upon the implicit cues of the user and learn from them.

Finally the game will be converted to a really interactive dynamic hide and seek game, where not only it is important to find the person, but also to approach it sufficiently.

In contrast with other applications as suggested in WP6 (archaeology, neuroscience, commercial appliance) we will not only use a fixed database, but several databases from which one is actually the live-feed of the robot sensors, such as the video stream, laser scanner ranges, people tracking states.

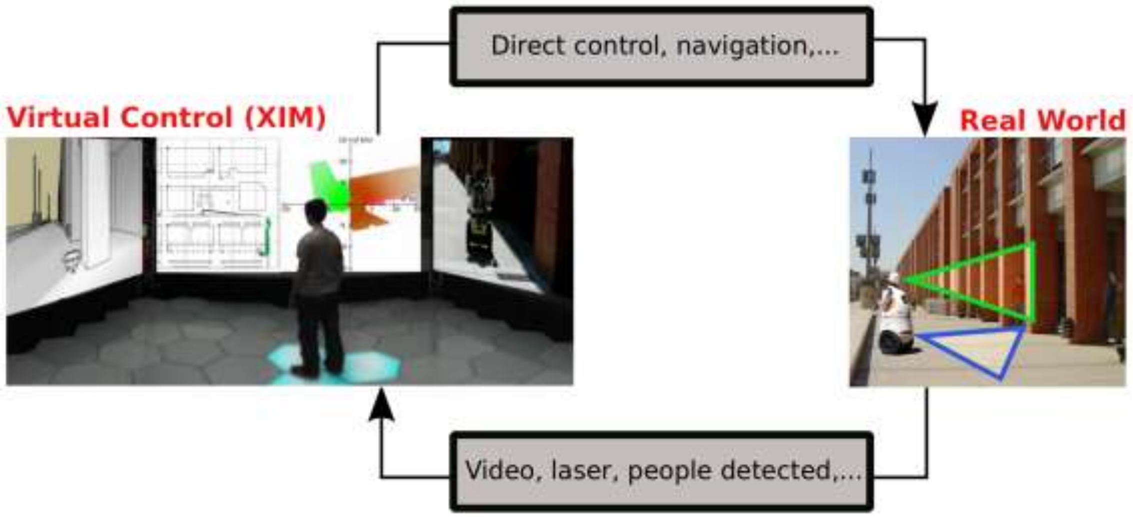

Direct control, navigation,...
Virtual Control (XIM)
Real World
Video, laser, people detected,...

Figure 7: The robot is controlled by the XIM user which sees the sensory data of the robot.

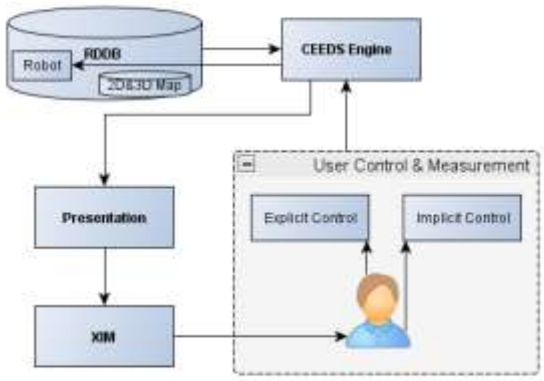


Figure 8: a schematic overview of the robotic hide and seek game.

This application has several CEEDs issues, first of all the amount of data captured by the robot is huge: unprocessed laser distances, and camera images and processed face recognition, localization, path planning, etc. From this data the hiders have to be found, and to play the game intelligently some sort of strategy of the opponent should be extracted and used to predict its steps.

The volume of information that has to be processed in real time will be very high and will have to be used for detecting and tracking a human player (who can be hidden at any time) and for navigation purposes. Obstacles and other non-player people can be present in the game field. Take into account that the person of the XIM perceives the environment through the perception sensors of the robot and there will be some delays in the perceived information and the robot motions.

References

[1] Fitzpatrick, P., Metta, G., & Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1), 29-45.

[2] Omedas, P., Bernardete, U., Betella, A., Tsaousi, E.N., Verschure, P., Leymarie, F.F., Olbrich, E., Pizzi, D., Jacucci, G., Wijekoon, D., Zanella, D. (2011). Deliverable 6.1: Conceptualization and Design of CEEDS system architecture for Applications. The Collective Experience of Empathic Data Systems (ICT-258749). Project deliverable.

[3] Sanfeliu, A., Andrade-Cetto, J., Barbosa, M., Bowden, R., Capitán, J., Corominas, A., Gilbert, A., et al. (2010). Decentralized Sensor Fusion for Ubiquitous Networking Robotics in Urban Areas. *Sensors*, *10*(3), 2274-2314.

[4] Sanfeliu, A., Andrade-Cetto, J. (2006). Ubiquitous networking robotics in urban settings, 2006 IROS Workshop on Network Robot Systems, Beijing, Xina, pp. 1-12, IEEE.

[5] Takacs, B., Omedas, P., Bernardete, U., Verschure, P. (2011). Deliverable 4.1: XIM 2.0 Environment. The Collective Experience of Empathic Data Systems (ICT-258749). Project deliverable.

[6] Trulls, E., Corominas Murtra, A., Pérez-Ibarz, J., Ferrer, G., Vasquez, D., Mirats-Tur, J. M., & Sanfeliu, A. (2011). Autonomous navigation for mobile service robots in urban pedestrian environments. *Journal of Field Robotics*, *28*(3), 329-354.