AsTeRICS Deliverable D2.1

System Specification and Architecture

UCY

SEVENTH FRAMEWORK PROGRAMME

European Commission
Information Society and Media

Document Information

Issue Date	**30 June 2010**
Deliverable Number	**D2.1**
WP Number	**WP2 System Specification and Architecture**
Status	**Final**
Dissemination Level	**RE** **PU Public** **PP Restricted to other programme participants (including the Commission Services)** **RE Restricted to a group specified by the consortium (including the Commission Services)** **CO Confidential, only for members of the consortium (including the Commission Services)**

AsTeRICS – Assistive Technology Rapid Integration & Construction Set
Grant Agreement No.247730
ICT-2009.7.2 - Accessible and Assistive ICT
Small or medium-scale focused research project

Version History

Version	Date	Changed	Author(s)
0.1	12 Feb. 10	First draft	Konstantinos Kakousis (UCY)
0.2	19 Feb. 10	First overview of the architecture provided for discussion STARLAB's Changes accepted	Nearchos Paspallis, Konstantinos Kakousis (UCY)
0.3	04. Mar. 10	Integrated contributions of partners, updated document structure	Nearchos Paspallis, Konstantions Kakousis (UCY)
0.4	06. Mar. 10	Integrated requirements, added ASAPI description, glossary, updated document structure	Chris Veigl (FHTW)
0.5	07. Mar. 10	Corrected template (heading 2), updaded references	Gerhard Nussbaum (KI-I)
0.6	08. Mar. 10	Added requirements and specifications	Chris Veigl (FHTW)
0.7	09. Mar. 10	UCY Reviewed the new version and accepted it with minor comments	Nearchos Paspallis, Konstantions Kakousis (UCY)
0.8	10. Mar. 10	Latest template	Konstantions Kakousis (UCY)
0.9	11.Mar.10	STARLAB sent contribution on Section 5	Javier Acedo (STARLAB)
0.10	31. Mar. 10	UPMC update to Smart Vision Module specifications + Glossary	Edwige Pissaloux, Francis Martinez (UPMC)
0.11	13.Jun.10	UCY Updated Section 4 with ARE and ASAPI details	Nearchos Paspallis (UCY)
0.12	14.Jun.10	KI-I Updated Section 4 with ASAPI details	Roland Ossmann (KI-I)
0.13	15.Jun.10	FHTW reworked large parts of the requirements and changed the HW platform specifications. Also integration of ASAPI definition from UCY	Chris Veigl (FHTW)
0.14	16.Jun.10	Integrated contribution from CEDO in Section 3	Tomas Drajsajtl (CEDO)
0.15	16.Jun.10	KI-I contribution in the gripper section and Conclusions	Roland Ossmann (KI-I)

0.16	16.Jun.10	STARLAB Updates in Sections 2, 4, 3 and 5	Javier Acedo, Aureli Soria Frisch (STARLAB)
0.17	16.Jun.10	CEDO update in Section 3	Tomas Drajsajtl (CEDO)
0.18	16.Jun.10	UPMC Contribution in Sections 2.1.3 and 3.4	Edwige Pissaloux, Francis Martinez (UPMC)
0.19	16 Jun. 10	Updated Native ASAPI section	Karol Pecyna (HARPO) Paul Blenkhorn (SENSORY)
0.20	21 Jun. 2010	Review	Jarek Urbański (HARPO)
0.21	27 Jun. 2010	Review	Zdenek Barton (CEDO)
1.0	02. Jul. 2010	Final	UCY, FHTW, KI-I

Glossary and Declaration of Terms

1 Terms Specific to AsTeRICS

ACS	**AsTeRICS Configuration Suite** A graphical software application running on the host PC for AsTeRICS model configuration and monitoring of the runtime system
ARE	**AsTeRICS Runtime Environment** The configured system model (which consists of pluggable components and their interconnections) which has been deployed to the execution environment (usually the AsTeRICS embedded computing platform)
ASAPI	**AsTeRICS Application Programming Interface** Provides methods to setup and interact with the ARE via a TCP/IP connection. The ASAPI builds the functional framework used by the ACS to generate and deploy the ARE system model and to configure ARE components. Furthermore, the ASAPI can be used to send and retrieve live data to or from the ARE, and to get logging/status from the ARE. The ASAPI can be used by third-party software applications integrate the ARE.
CIM	**Communication Interface Module** Hardware interface and dedicated driver software to connect a sensor / actuator module or to use a standardized communication medium/protocol.
EP	**Embedded Platform (aka AsTeRICS Personal Platform)** A customized hardware / PCB with high performance low power CPU and dedicated Communication Interface Modules to support connection of sensors and actuators.
PCOM	**Pluggable Component Module** Software components of the ARE, which can be grouped into **Signal Sources** (featuring output ports), **Signal Processors** (featuring input and output ports) and **Signal Sinks** (featuring input ports
SVM	**Smart Vision Module** A computer vision sensor with dedicated computing module for feature extraction. Transfers images features to the ARE.

2 Other Relevant Technical Terms

BCI	**Brain Computer Interface**
BNCI	**Brain or Neural Computer Interface**
FFT	**Fast Fourier Transformation**
IR	**Infrared**
JNI	**Java Native Interface**
OS	**Operating System**
OSGi	**Open Service Gateway initiative**
CPU	**Central Processing Unit**
DSP	**Digital Signal Processor**
IMU	**Inertial Measurement Unit**
PCB	**Printed Circuit Board**
SRAM	**Static Random Access Memory**
USB	**Universal Serial Bus**
VPIF	**Video Port Interface**
LCD	**Liquid Crystal Display**
UART	**Universal Asynchronous Receiver/Transmitter**
DAC	**Digital to Analog Converter**
HID	**Human Interface Device**
PCA	**Principal Component Analysis**
CSP	**Common Spatial Pattern**
OVR	**One Versus the Rest**
CVT	**Canonical Variates Transformation**
LDA	**Linear Discriminant Analysis**
ADC	**Analog to Digital Converter**
GPIO	**General Purpose Input / Output**
EMG	**Electromyography**

EEG	**Electroencephalography**
EOG	**Electrooculography**
SVEM	**Support Vector Machine**
FPR	**False Positive Rate**
TPR	**True Positive Rate**
VOG	**Video-Oculography**
SDK	**Software Development Kit**
HMI	**Human Machine Interface**
FIR	**Finite Impulse Response**
JVM	**Java Virtual Machine**
FIFO	**First In First Out**
MVC	**Model View Controller**
MVVM	**Model View Viewmodel**
WPF	**Windows Presentation Foundation**
COTS	**Commercial Off-The-Shelf**
AT	**Assistive Technology**

Table of Content

1 Introduction

The purpose of Work Package 2: “System Architecture and Specification” is to define in detail the requirements and characteristics of the Assistive Technology Rapid Integration and Construction Set (AsTeRICS) project. AsTeRICS’ objective is to enable easy-to-use Assistive Technology, based on user requirements and needs. This document aims to describe the system specification and architecture for the overall project. In particular, we identify the system requirements for hardware and software and present in detail the specification of the hardware and software architecture.

1.1 AsTeRICS System Architecture Overview

The system architecture of AsTeRICS consists of hardware and software components, which provide the flexible Assistive Technology Rapid Integration- and Construction Set.

Hardware Components:

- AsTeRICS Personal Platform – an embedded high performance computing system
- Communication Interface Modules (CIMs), allow attachment of sensors and actuators
- Sensors and actuators, which include:
    - modules designed and manufactured by consortium members as major objectives of the AsTeRICs project (like the Smart Vision Module)
    - extensions of existing hardware of consortium members (like the Starlab Enobio device)
    - integration of commercial off-the-shelf AT devices (like special joysticks, switches, IR-gateway etc.)

Software Components:

The software components consist of individual packages for host computer and for the runtime environment. The software packages dedicated to the host computer are:

- the AsTeRICS Configuration Suite (ACS)
- the AsTeRICS Application Programming Interface (ASAPI)
- applications which are developed or extended in course of the AsTeRICS project
- the BNCI Evaluation Suite by Starlab
- AT-software solutions by SENSORY like the OSKA on-screen keyboard

The software components dedicated to the AsTeRICS Runtime Environment include:

- the middleware architecture for the ARE
- the Pluggable Component Modules (PCOMs):
- Signal Sources
- Signal Processors
- Signal sinks

The architectural framework as a whole provides the functionality of the AsTeRICS system as defined in the Description of Work document [1], approved by the EC on 22-10-2009, which can be summarised as follows:

- Making sensor data from the user available for processing on the platform (interfacing with sensor module-drivers which are part of the operating system),
- Routing the sensor data to and among signal processing and signal shaping elements, as defined by the interconnection of these elements, for the purpose of extraction of relevant information about voluntary user commands. Furthermore, routing this information to actuator elements,
- Controlling actuators for AT purposes (interfacing with actuator module-drivers provided by the operating system),
- Enabling remote access to the embedded platform via TCP/IP (for download and upload of actual configurations, error reporting and display of live data).

1.2 Relationship to other AsTeRICS Deliverables

The deliverable is related to the following AsTeRICS deliverables:

- D1.1 Report on Users, Preferences and Needs: AsTeRICS follows a user-centric approach and the user needs and requirements are the main input for elicitation of the system requirements.
- D1.3 “Technical Specifications”: This document describes the transformation of user requirements into technical requirements and outlines basic use cases for the AsTeRICS system. It thereby defines the capabilities which have to be met by the AsTeRICS architecture.
- D2.4 “Report on the State of the Art” [3]: This document outlines the technological basis for the planning of the AsTeRICS architecture and describes relevant hardware and software components available on today’s market.
- D2.3 “Report on API specification for sensors to be integrated into the AsTeRICS Personal Platform Prototype 1” [2]: This document shows the setup of software plugins and the adjustment of plugin parameters using the AsTeRICS Application Programming Interface
- D4.7 “Report on feasibility of OSGi porting to the AsTeRICS Personal Platform” [4]: This document contains documentation of performance tests and porting efforts which influenced the decision for the AsTeRICS Personal Platform hardware

1.3 Relationship to Description of Work

According to the Description of Work [1], AsTeRICS will provide a flexible and affordable construction set for user driven Assistive Technologies or assistive functionalities. The AsTeRICS architecture is derived from the user’s requirements and needs and combines the flexibility and modularity of the Java programming language and OSGi compositional framework with state-of-the-art software and hardware sensors and actuators.

1.4 Structure of This Document

The deliverable is structured as follows. Section 1 first provides an overall view of the system architecture and briefly describes the AsTeRICS approach in system architecture and specification. Section 2 identifies the system requirements which are classified in hardware requirements (Section 2.1), software requirements (Section 0) and requirements for the BNCI evaluation suite (Section 2.3). Section 3 details the hardware specification of the embedded platform and other custom hardware modules. In particular, Section 3.1 defines the specification of the AsTeRICS Embedded Platform while Section 3.2 describes extra communication modules needed. Section 3.3 defines the custom sensor- and actuator modules we plan to develop while Section 3.4 defines the specifications of the AsTeRICS Smart Vision Module. Section 4 details the software specification and architecture of the configuration and runtime components and reveals the system model and middleware platform components. In particular, Section 4.1 presents the AsTeRICS system model while sections 4.2 and 4.3 describe the AsTeRICS runtime environment and configuration suite, respectively. Finally Section 4.4 presents the AsTeRICS application programming interface. Section 5 presents the architecture and specifications of the BNCI evaluation suite while Section 6 concludes this document with an overview of the main contributions of this deliverable to WP2 and to the AsTeRICS project in general.

2 AsTeRICS Requirement Analysis

In the following sections, a comprehensive list of requirements for the overall AsTeRICS system will be presented, clustered into the following subsections:

Hardware requirements

- Hardware requirements for the AsTeRICS embedded platform (Section 2.1.1)
- Requirements for AsTeRICS pluggable components (sensor and actuator hardware modules) (Section 2.1.2)
- Hardware-requirements for the Smart Vision Module (Section 2.1.3)

Software requirements

- AsTeRICS Runtime Environment requirements (Section 2.2.2)
- AsTeRICS Configuration Suite requirements (Section 2.2.3)
- AsTeRICS Application Programming Interface requirements (Section 2.2.4)

A particular requirement will be identified by a prefix for the group and a unique number. The requirement description includes the priority of a particular implementation and the expected time to be delivered (i.e., either in the first or second prototype). Requirements identified as high priority requirements should be provided as they reflect basic functionalities expected to be delivered by the project as defined in the AsTeRICS Description of Work [1]. Requirements with medium or low priority are either "nice to have" features or extra functionality not directly affecting the project’s goals.

2.1 Hardware Requirements

2.1.1 Hardware Requirements for the AsTeRICS Embedded Platform (EP)

The table below lists the hardware requirements for the AsTeRICS embedded platform as they were elicited from the user needs identified in AsTeRICS deliverables D1.1 – “Report on Users, Preferences and Needs”, D1.3 –“Technical Requirements” and D2.4 – “Review on State of the Art”.

Nr.	**Requirement**	**Description**	**Priority**	**Prototype**
HW1	Size and weight	The Embedded Platform hardware consists of small modules which can be combined as needed. A display with touchscreen (7 to 10 inches) can be added as a separate module.	H	2
HW2	Low power consumption	The hardware components have to operate with low power requirements and/or offer power management functions for acceptable battery lifetime.	M	1, 2
HW3	Temperature Range	The desktop components shall have 0 to 60 ˚C temperature range but the portable components which are intended to be used also outside should have extended range at least from -20 to 85 ˚C.	H	1, 2
HW4	Performance	The Embedded Platform has to be powerful enough to support online image processing functionalities, e.g. for the Smart Vision Module or a webcam.	H	2

HW5	Hot-Pluggability	A connection or disconnection of a component should not affect or interrupt the functionality of the rest of the system. An automatic detection of a newly connected or disconnected peripheral is desired.	L	1, 2
HW6	Affordable price	The whole system has to be affordable at acceptable cost, compared to other available AT products with similar or less functionalities.	H	2
HW7	Portability	The system can be battery powered and is usable without a connection to the mains power supply for at least 5 hours.	H	2
HW8	Platform Interfaces	The EP supports the hardware interfaces which are necessary for connecting sensors and actuators in sufficient amount (Bluetooth, 6 x USB, ZigBee, WiFi, Ethernet).	H	1, 2
HW9	UART interface	The EP supports integration with UART interface.	H	1, 2
HW10	LCD interface	The EP supports integration with an LCD display with touchscreen.	H	2
HW11	Sufficient Memory	A large non volatile memory is essential to hold operating system, ARE components and system model configurations (at least 4GB). A large RAM is needed for the runtime system to work efficiently. (at least 512MB, depending on the Operating System).	H	1, 2
HW12	Operating System	For low latency processing of parallel sensor values (multiprocessing), TCP/IP stack, memory management, JAVA Virtual Machine, Windows or Linux based OS solution should be supported.	M	2

Table 1: Hardware requirements for the AsTeRICS embedded platform

2.1.2 Requirements for AsTeRICS Pluggable Hardware Components

The table below lists the hardware requirements for pluggable components, such as sensors, processors and actuators that the AsTeRICS system is expected to support by default.

Nr.	**Requirement**	**Description**	**Priority**	**Prototype**
Hardware Requirements for pluggable sensors				
HSEN1	Generic switches (GPIO CIM)	The AsTeRICS system supports connectivity to at least 5 generic switches via a dedicated module (GPIO-CIM, 3,5mm jacks, digital input).	H	1,2
HSEN2	Sweety!	The AsTeRICS system should support Sweety! Switches via Bluetooth connectivity.	L	2
HSEN3	USB HID devices	The AsTeRICS system supports USB mice, keyboards and joysticks via USB connectivity.	H	1, 2
HSEN4	9 DOF Razor IMU	The AsTeRICS system supports the 9 DOF Razor Inertial Measurement Unit.	H	1, 2
HSEN5	Strain Gauge	The AsTeRICS system supports Strain Gauge connectivity (via ADC-CIM).	H	1, 2
HSEN6	Touchscreen	The AsTeRICS system supports Touchscreen connectivity (e.g. via USB).	H	2
HSEN7	Enobio	The AsTeRICS system supports connectivity with the Enobio System (wired or via ZigBee).	H	1, 2
Hardware Requirements for pluggable actuators				
HACT1	Mobile Phone Interface	The AsTeRICS system supports integration with Mobile Phone Interface via Bluetooth wireless link	H	2

HACT2	LC-Display Interface	The AsTeRICS system should support integration with an LCD interface (e.g a USB-pluggable LCD with touchscreen)	H	1, 2
HACT3	IR Connectivity	The AsTeRICS system should support infrared connectivity by interfacing to a suitable IR gateway	H	2
HACT4	Digital to Analog Converter (DAC-CIM)	The AsTeRICS system provides a Digital-to-Anolog Converter module (DAC-CIM) with 5 channels (minimum) between 0 and to 25 Volt. At least 2 channels provide high power up to 5 Watts	M	2
HACT5	HID Mouse Actuator	The AsTeRICS system supports mouse emulation for a PC via the USB HID device class	H	1, 2
HACT6	HID Keyboard Actuator	The AsTeRICS system supports keyboard emulation for a PC via the USB HID device class	M	2
HACT7	HID Joystick Actuator	The AsTeRICS system supports Joystick emulation for a PC via the USB HID device class	H	2
HACT8	KNX-easy	The AsTeRICS System supports integration with KNX-easy	H	2
Hardware Requirements for Non-Classical PC-User Interfaces				
HNCPUI1	3D accelerometers	The AsTeRICS system supports connectivity with 3D accelerometers	H	2
HNCPUI2	Webcam as Face Mouse	The AsTeRICS system should support connectivity with webcams that serve as face mice	L	2
HNCPUI3	Webcam for other input modalities (e.g. colour tracking)	The AsTeRICS system should support connectivity with webcams that serve as other input modalities	L	2
HNCPUI4	External Touchpad/keypad (used in novel ways, e.g. in “Joystick” mode)	The AsTeRICS system supports connectivity with Touchpad / Keypad that can be used in novel ways for PC-User interfacing	H	2
HNCPUI5	GamePads/ Joysticks	The AsTeRICS system supports connectivity with GamePads/Joysticks	M	2
HNCPUI6	Mobile phone with touch screen	The AsTeRICS system supports connectivity with Mobile Phones with touch screens	M	2

Table 2: Hardware requirements for the AsTeRICS pluggable components

2.1.3 Hardware Requirements for the AsTeRICS Smart Vision Module

The table below lists the requirements expected to be provided by the Smart Vision Module which is going to be developed during the project.

Nr.	Requirement	Description	Priority	Prototype
SVM1	Eye camera	The Smart Vision Module should acquire images from an eye camera and process them	H	1,2
SVM2	Scene camera	The Smart Vision Module should acquire images from a scene camera and process them	H	1,2
SVM3	Inertial Measurement Unit	The Smart Vision Module should acquire data from an IMU and process them	M	2
SVM4	Synchronization interface	The Smart Vision Module should synchronize the data of the eye and scene cameras and the IMU	H	1,2

SVM5	Head-mounted support	The head-mounted system of the Smart Vision Module should be lightweight and intrusiveness should be minimized	H	1,2
SVM6	Ewe detection	An eye detection algorithm should be proposed/modified and implented	H	1,2
SVM7	Gaze estimation	A gaze estimation algorithm should be proposed/modified and implemented	H	1,2

Table 3: Hardware requirements for the AsTeRICS Smart Vision Module

2.2 Software Requirements

2.2.1 Software Requirements for Pluggable Component Modules

The table below lists the software requirements for pluggable components (software plugins), such as sensor-plugins, processor-plugins and actuator-plugins that the AsTeRICS system is expected to support by default. Each hardware requirement in Section 2.1.2 should have a counterpart software requirement in the table below. In addition requirements for purely software components are listed below.

Nr.	Requirement	Description	Priority	Prototype
Software Requirements for sensor plugins				
SSEN1	GPIO/ Generic switches plugin	A software plugin exists which makes the state of the external generic switches available on its output ports.	H	1,2
SSEN2	Sweety! plugin	A software plugin exists which makes the state of the Sweety! – buttons (connected via Bluetooth) available on its output ports.	L	2
SSEN3	USB HID class support	The operating system of the AsTeRICS Embedded Platform supports USB devices classes and HID host functionality to connect mice, keyboards and joysticks.	H	1, 2
SSEN4	9 DOF Razor IMU plugin	A software plugin exists which provides data from the 9 DOF Razor Inertial Measurement Unit.	H	1, 2
SSEN5	ADC/Strain Gauge plugin	A software plugin exists which provides ADC data from the ADC CIM (provide e.g. Strain Gauge data).	H	1, 2
SSEN6	Touch-screen	The AsTeRICS system and/or the operating system of the EP supports touchscreen connectivity.	H	2
SSEN7	Enobio	A software plugin exists, which supports channel data readout of the Enobio system.	H	1, 2
SSEN8	FTDI chip driver	In case a USB2.0 connection to the Enobio system is used, the FTDI driver has to be available in the operating system.	H	1,2
SSEN9	IEEE 802.15.4 antenna driver	In case a built-in wireless receiver is used, its software driver shall be available for the Enobio software control.	M	2
SSEN10	Serial port driver	The operating system allows access to the serial port / virtual COM port to provide wired connections to UART/RS232 devices.	M	1,2
Software Requirements for actuator plugins				

SACT1	Mobile Phone Interface plugin	A software plugin exists which supports integration of one or more mobile phones (send SMS, call a number).	H	2
SACT2	LC-Display Interface support	The AsTeRICS system and/or the operating system of the EP supports integration of a portable LCD module with touchscreen.	H	1, 2
SACT3	IR Connectivity plugin	A software plugin exists which supports interfacing with a suitable infrared gateway.	H	2
SACT4	Digital to Analog Converter plugin	A software plugin exists which can control the Digital-to-Analogue converter module (DAC CIM).	M	2
SACT5	HID Mouse Actuator plugin	A software plugin exists which supports mouse emulation for the PC via the HID mouse actuator.	H	1, 2
SACT6	HID Keyboard Actuator plugin	A software plugin exists which supports keyboard emulation for the PC via the HID keyboard actuator.	M	2
SACT7	HID Joystick Actuator plugin	A software plugin exists which supports joystick emulation for the PC via the HID joystick actuator.	H	2
SACT8	KNX-easy	The AsTeRICS System supports integration with KNX-easy.	H	1, 2
Software Requirements for signal processing plugins				
SPROC1	Filtering EMG signal: 8-500 Hz	Frequency filtering to be applied to ENOBIO channels shall be implemented.	H	1
SPROC2	Filtering Alpha band and Mu Rhythm: 8-12Hz	Frequency filtering to be applied to ENOBIO channels shall be implemented.	H	1
SPROC3	Filtering Beta band: 12-30 Hz	Frequency filtering to be applied to ENOBIO channels shall be implemented.	H	1
SPROC4	High pass filter with frequency cut at 1 Hz	Frequency filtering to be applied to ENOBIO channels shall be implemented.	H	1
SPROC5	Laplacian filter analysis	The feasibility and usefulness of Spatial Laplacian filter for the reduced number of ENOBIO channels shall be analyzed.	L	2
SPROC6	FFT computation	Computation of Power Spectrum Density through FFT.	H	1
SPROC7	Epoch cutting	Epoch cutting.	H	1
SPROC8	Epoch Averaging	Epoch Averaging.	H	1
SPROC9	Linear Transformation	Linear Transformation (matrix product).	H	1
		Transformation matrix will be defined offline (this will allow online ICA, PCA, CSP, linear inverse solution, weighted mapping, CAR referencing, simple Laplacian, etc…).	M	2
SPROC10	Threshold	Application of a threshold to transform continuous output into a binary output.	H	1
SPROC11	Derivative	Derivative of some selected channels.	H	1
SPROC12	Decimation	Decimation of some selected channels.	H	1
SPROC13	Dissimilarity	Dissimilarity.	H	1
SPROC14	PCA projection	PCA projection (projection matrix can be pre-computed).	H	2
SPROC15	CSP and OVR	CSP and OVR.	L	2
SPROC16	CVT	CVT.	L	2
SPROC17	SVEM	SVEM (Support Vector Machine).	M	2

SPROC18	LDA	LDA.	M	2
SPROC19	Fuzzy C-means	Fuzzy C-means.	M	2
SPROC20	Cross Correlation	Correlation function for template matching.	H	1
Software Requirements for BNCI signal processing plugins				
SBPROC1	Teeth grinding detection	The output will be a numerical value within a range.	H	2
SBPROC2	Frown detection	Frown detection to be analyzed. The output will be a numerical value within a range.	M	2
SBPROC3	Blink detection	Blink detection. The output will be binary.	H	1
SBPROC4	Double blink detection	Double blink detection to be analyzed.	H	1
SBPROC5	Horizontal eye movement's detection	Horizontal eye movement's detection. The output will be single events (right/left).	M	2
SBPROC6	Winks detection	Winks detection to be analyzed.	M	2
SBPROC7	Mouth movement	Mouth movement detection to be analyzed.	M	2
SBPROC8	Emotional content	Emotional content mentioned by Damasio 2003 [5] to be analyzed.	L	2
SBPROC9	Motor imagery	Motor imagery to be analyzed for binary.	M	2
SBPROC10	Detection of language thinking	Detection of language thinking to be analyzed for binary.	L	2

Table 4: Software requirements for the AsTeRICS pluggable components

The Requirements from SPROC1 to SPROC20 are extracted from the SoA review on EEG, EOG and EMG based assistive technologies and Report on Pattern Recognition Technologies for BNCI. The purpose of the signal processing functionality set to be integrated in the ARE is online process of different physiological signals to build assistive technologies.

With regard to the Software Requirements for Signal Processing (SPROC1-20) and BNCI Signal Processing (SBPROC1-9), only the one of medium or high priority will be implemented either in prototype 1 or 2, leaving the implementation of low priority ones for the case that enough resources are available. For the prototype 1, functionalities of high feasibility attached to this prototype will be implemented, since its feasibility seems to be ensured. In case of medium level of feasibility, a feasibility study of the corresponding algorithms will be undertaken during the prototype 1 implementation phase in order to decide on its implementation for prototype 2.

2.2.2 Software Requirements for the AsTeRICS Runtime Environment

The AsTeRICS Runtime Environment provides the execution framework for the pluggable components and supports the ASAPI communications. For more details on the ARE specification please see Section 4.2. The table below lists the identified requirements for the ARE.

Nr.	**Requirement**	**Description**	**Priority**	**Prototype**
ARE1	Support sensor plugins	The ARE supports Sensing modules which provide information via output ports.	H	1,2

		For low level sensors that need OS/driver communication we should use JNI for communicating data to ARE.		
ARE2	Support processor plugins	The ARE supports processing elements which transform information from input ports and provide results on output ports.	H	1, 2
ARE3	Support actuator plugins	The ARE supports actuator elements, using information from input ports that might be connected to the output port of a sensor or processor plugin. May interface to operating system / driver layer via JNI.	H	1, 2
ARE4	Dynamic deploy, activation, deactivation and removal of components	ARE supports dynamic deploy, activation, deactivation and removal of elements. ARE should be build upon OSGi / Java to enable this feature.	H	1, 2
ARE5	Remote access to the platform	The ARE supports remote access to the platform via a communication network.	H	1, 2
ARE6	Feedback and error reporting	The ARE supports querying deployed components status and error reporting.	M	2
ARE7	Load / store configuration on platform	The ARE supports loading an existing configuration system model and storing a new or changed configuration system model from/to a hard disk.	M	2
ARE8	Computing load can be monitored	The ARE should be capable of monitoring the system computing load.	L	2
ARE9	Direct manipulation of pluggable components	The ARE allows setting and manipulating element properties directly (not through model submission) through a communication network or other direct interaction with the ARE (buttons, touch screen, etc).	M	2
Requirements for signal connections and data flow within the ARE				
ARE10	Data types on Input / Output ports	Each Input and Output port attached on a component have a specific data type. Supported data types include boolean, int, longint, real (float), int-vector, float-vector, string, state; while new data types for new elements should be possible.	H	1, 2
ARE11	Restricted connections	Connections (channels) can only be established from one output port to one or many input ports of the same type. Only matching connections can be established, also it is not possible to connect more than one signal to an input port.	H	1, 2
ARE12	Metadata on ports	Each port can be connected with some type of metadata about its capabilities, data type, signal bandwidth, block size, timestamp, etc.	H	1, 2
ARE13	Data flow	Data can be transferred via channels (from output to input ports) as single values or in blocks (data chunks).	H	1, 2
ARE14	Channel update rate	The update rate of a channel is defined by the metadata of the output port. This rate can be fixed or changed according to the output port activity.	M	2
ARE15	Synchronous and Asynchronous events reaction	Components are able to listen to Synchronous or Asynchronous events (through ARE) and react accordingly.	H	1, 2

Table 5: Software requirements for the AsTeRICS Runtime Environment (ARE)

2.2.3 Requirements for the AsTeRICS Configuration Suite

The AsTeRICS Configuration Suite (ACS) is mainly used to graphically design the layout of the system, as a network of interconnected components. ACS should also support the deployment of additional components, such as an oscilloscope for monitoring the ARE behaviour. A full specification of the ACS is given in Section 4.3

Nr.	Requirement	Description	Priority	Prototype
ACS1	Graphically design system model	ACS enables graphical design of the system model on a host PC as a network of components. Arrangement of the graphical elements as well as properties configurations should be enabled via mouse / keyboard actions.	H	1, 2
ACS2	Graphically interconnect elements	The ACS GUI allows for connecting two components through their output/input ports. Drawing of connections (channels) from output to input port defines data flow, channel and port parameters should be adjustable.	H	1, 2
ACS3	User-friendly graphical user interface	The Configuration Suite provides a user friendly and accessible graphical user interface. It is important that usage of the ACS is as easy and accessible as possible.	H	2
ACS4	Support connection with the ARE	The ACS is able to connect to the ARE the ASAPI. The connection should be used to download or upload the system model, changing plugin parameters and logging reports.	H	1, 2
ACS5	Components can be grouped and ungrouped in the graphical display	To handle complexity of larger designs, components can be grouped and ungrouped in the graphical display.	L	2
ACS6	Edit properties	Display and edit properties and metadata information on components and ports.	H	1

Table 6: Requirements of the AsTeRICS Configuration Suite

2.2.4 Requirements for the AsTeRICS Application Programming Interface (ASAPI)

The AsTeRICS Application Programming Interface (ASAPI) provides a well defined method for software applications like the AsTeRICS Configurations Suite (ACS) or third party applications to configure, monitor and control the AsTeRICS Runtime Environment (ARE). Section 4.4 provides detailed specification of ASAPI. The table below collects the requirements for the AsTeRICS application programming interface (ASAPI).

Nr.	Requirement	Description	Priority	Prototype
ASAPI1	Detection of ARE	ASAPI provides methods for detecting an instance of ARE server.	H	1, 2
ASAPI2	Connection with ARE	ASAPI provides methods for establishing a connection with a detected ARE server.	H	1, 2
ASAPI3	Handle communication errors	ASAPI provides methods for handling ARE communication errors and breakdowns.	M	1, 2
ASAPI4	Query installed plugins	ASAPI provides methods for querying plugins installed in ARE directly (i.e, without retrieving the whole system	H	1, 2

		model). Queries should return plugins' parameters and properties.		
ASAPI5	Install plugin	ASAPI provides methods to install plugins to the ARE.	H	1, 2
ASAPI6	Create plugin instance	ASAPI provides methods for creating new instance of installed plugin.	M	1, 2
ASAPI7	Ports connection	ASAPI provides methods for interconnecting installed plugins.	H	1, 2
ASAPI8	Run/Stop plugins	ASAPI provides methods for starting or stopping individual plugins.	M	1, 2
ASAPI9	Retrieve system model	ASAPI provides methods for retrieving the system configuration model in a serializable format.	H	1, 2
ASAPI10	Deploy system model	ASAPI provides methods for deploying a new configuration system model.	H	1, 2
ASAPI11	Run/Stop deployed model	ASAPI provides methods for initiating or terminating the execution of deployed model.	H	1, 2
ASAPI12	Retrieve logging information	ASAPI provides functions for retrieving logging. and status information from the runtime environment.	M	1, 2
ASAPI13	Exchange live data	ASAPI provides methods for exchanging live data with connected ARE.	H	2
ASAPI14	Native Interface	Certain functions are provided natively (in C#) for PC AT developers via the Native ASAPI (e.g. for making mobile phones or special sensors available).	M	1,2

Table 7: AsTeRICS Application Programming Interface (ASAPI) requirements

2.3 BNCI Evaluation Suite Requirements

The table below lists the requirements for the BNCI Evaluation Suite. The requirements table is divided into *functional* and *other* requirements. They present priorities and feasibility of the requirements as established.

Nr.	Description	Feasibility	Priority	Prototype
BNCI Evaluation Suite Functional Requirements				
BNCI1	BNCI Evaluation Suite shall not work in real-time.	H	H	1, 2
BNCI2	Recalling parameters for real-time on-line processing might be allowed.	M	M	2
BNCI3	The input file data formats will be the standard ones used by the acquisition hardware.	H	H	1
BNCI4	Data formats used by BIOSIG toolbox [33] shall be accessed from the Evaluation Suite.	H	H	1
BNCI5	Starlab data cube (SDC) format shall be used.	H	H	1, 2
BNCI6	Classification functionalities of the BIOSIG toolbox shall be interfaced from the Evaluation Suite.	H	H	1
BNCI7	Temporal windowing of trial sequences shall be implemented.	H	H	1
BNCI8	Temporal 8-order Chebyshev Type I band-pass filtering of trial sequences shall be implemented.	H	H	1

BNCI9	Different projection techniques for dimensionality reduction shall be implemented.	H	H	1
BNCI10	A decision tree methodology might be implemented.	M	M	2
BNCI11	Re-sampling training functionalities might be implemented.	H	M	2
BNCI12	Data fusion operators shall be implemented for integration in classifier ensemble methodologies.	H	H	1
BNCI13	Genetic algorithms might be implemented for system optimization.	H	M	1
BNCI14	A fuzzy control algorithm shall be implemented.	H	M	2
BNCI15	Feature extraction based on Bereitschaft potential shall be implemented	H	H	2
BNCI16	PSD estimation based on multitaper approaches shall be implemented.	H	H	2
BNCI17	An approach based on mutual information ranking shall be implemented for feature selection.	M	H	2
BNCI18	SVEM shall be implemented.	H	H	1
BNCI19	Linear proximal SVEM shall be implemented.	M	H	1
BNCI20	A logistic regression algorithm shall be implemented.	M	L	1
BNCI21	Linear discriminant analysis shall be implemented.	H	H	1
BNCI22	Temporal decimation of trial sequences shall be implemented.	H	H	1
BNCI23	Temporal averaging of trial sequences shall be implemented.	H	H	1
BNCI24	Performance evaluation shall be implemented based on TPR and CA measures.	H	H	1
BNCI25	Wavelet transformation of a temporal sequence shall be implemented.	M	H	1
BNCI26	A procedure for analysis of variance (ANOVA) shall be implemented.	H	M	1
BNCI27	At least one performance measures for BCI shall be implemented, e.g. kappa, TPR/FPR, ITR.	H	H	1
BNCI28	Recall parameters of a particular framework might be saved on disk for posterior on-line processing.	L	M	2
BNCI29	An application based on P300 for image browsing shall be implemented.	M	H	1
BNCI30	P300 classical approach shall be implemented.	H	H	1
BNCI31	Classical BCI based on motor imagery might be implemented.	H	L	2
BNCI32	An approach for single-trial source reconstruction for BCI might be implemented.	M	M	2

BNCI33	StarEEGlab [34] user interface shall apply.	H	H	1
BNCI Evaluation Suite Other Requirements				
BNCI34	BNCI Evaluation Suite shall be implemented in two phases. Prototype 1 (PT1) to be delivered till 30/04/2011.	H	H	1
BNCI35	BNCI Evaluation Suite shall be implemented in two phases. Final Prorotype (FP) to be delivered till 31/05/2012.	H	H	2
BNCI36	Starlab shall integrate the Evaluation Suite functionalities within the StarEEGlab toolkit.	M	H	2

Table 8: BNCI Evaluation Suite requirements

Requirements with High feasibility and High priority that are expected to be delivered in Prototype 1 will be attained first. Then requirements of Medium feasibility and High priority for Prototype 1 might be realized. For this, a feasibility study that confirms the feasibility should be undertaken earlier.

3 Hardware Specification and Architecture

The usual hardware configuration of the main system consists of the Embedded Computing Platform and connected sensors and actuators via standard integrated interfaces or via special Communication Interface Modules (CIMs). Thanks to the high modularity, the system can be adapted to the user's abilities and needs and mounted as desktop or portable system. The key components of the HW platform will be designed to support both desktop and portable variants as much as possible.

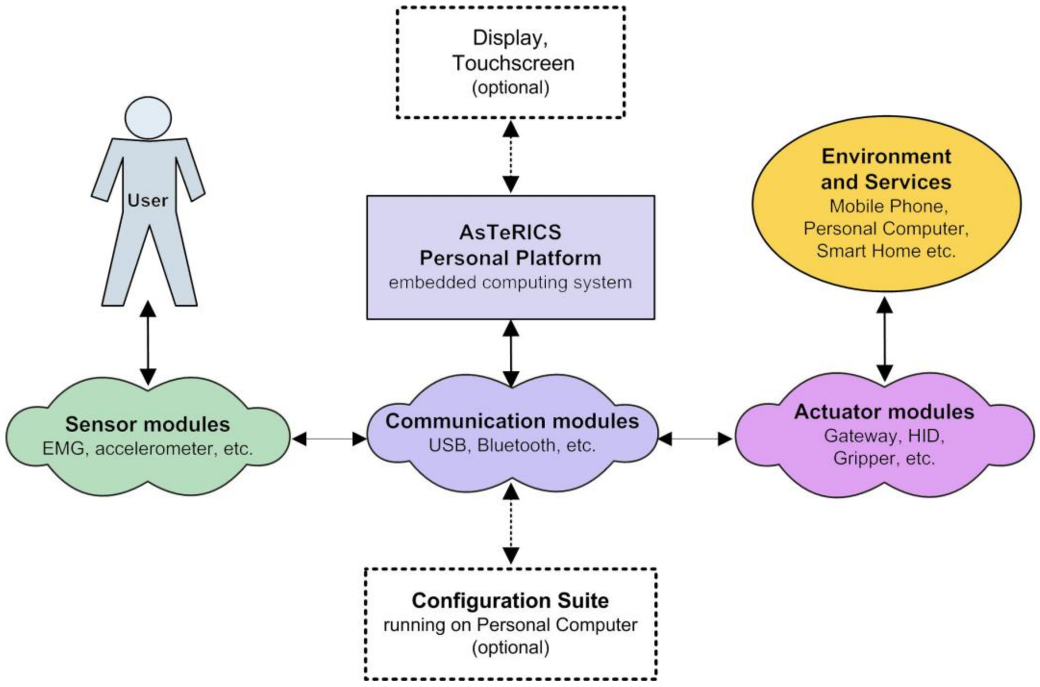


Figure 1: Concept of the modular Assistive Technology system

The architecture of the system is described in Figure 1. The personal platform contains core computer described in chapter 3.1 and the core expansion module described in 3.2.1. When the set of interfaces provided on the personal platform is not sufficient, more communication modules specified in chapter 3.2 can be added and connected via USB interface to the system. Custom sensor and actuator modules being developed in the project are then described in the rest of chapter 3.

3.1 Specification of the AsTeRICS Embedded Platform

For the first AsTeRICS system prototype, the State-of-the-Art analysis revealed some interesting candidates for the Embedded Platform hardware (see D2.4, [3]). In course of the system evaluation and performance tests performed in course of D4.7 – "Porting of OSGi to the Personal Platform", the Kontron 1,6Ghz Atom Single Board Computer (SBC) with pico-ITX form factor [15] proved to be the best option due to the following reasons:

- The Atom platform features a low-power chipset and provides best-in-class performance at a reasonable power dissipation
- The Kontron board features a robust design and rich connectivity
- All interesting operating systems are supported, including recent Windows and Linux distributions

The Kontron pico-ITX SBC fulfills all hardware requirements for the AsTeRICS computing platform (HW1-HW12) defined in section 2.1.1, except the interfaces for Bluetooth, ZigBee and WiFi (HW8 and HW9), which will be provided by the Core Expansion Module (see section 3.2.1).

In the following sections, basic specifications, connectivity and functional features of the Kontron pico-ITX SBC will be described, including a measurement of the power consumption of the board under various system loads.

3.1.1 Kontron pITX SBC - dimension and operating conditions

The following table comprises general specifications of the Kontron 1.6 Ghz pico-ITX Single Board Computer (SBC), as dimensions and operating conditions [15]:

Product	Kontron pITX SBC
Dimensions (H x W)	100 x 72 mm (Pico-ITX)
Temperature Operating	0 °C – 60 °C (32 °F ~140 °F)
Temperature range	0 – 60 °C ambient temperature, active and passive cooling solutions available
Power Supply / Consumption	5 V DC (5 W typical)

Table 9: Kontron pITX SBC dimension and operating conditions

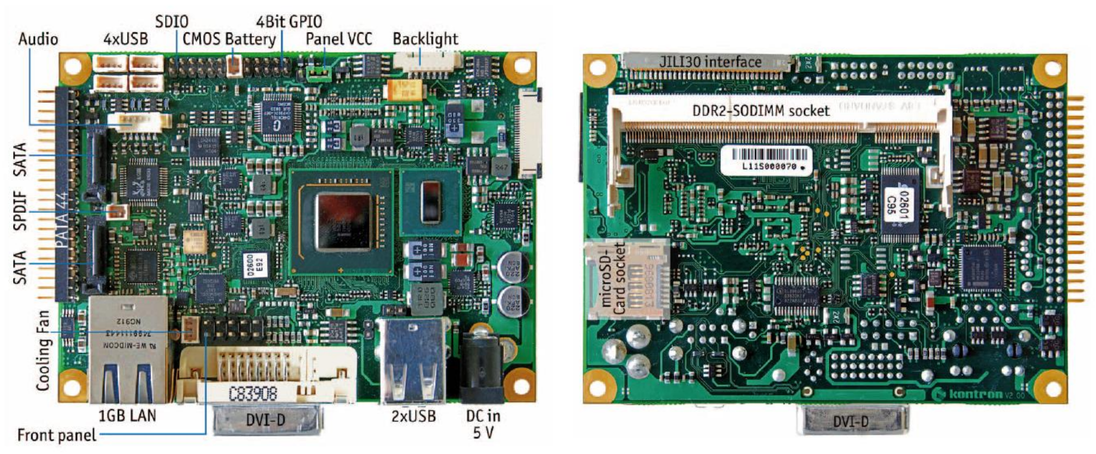


Figure 2: Kontron pITX SP Single Board Computer, top and bottom view

3.1.2 Kontron pITX SBC - Functional Specifications

The following table lists the functional specifications of the Kontron pico-ITX Single Board Computer [15]:

Processor	Intel Atom Z510 (1.1 GHz) or Z530 (1.6 GHz) with 24 kB data and 32 kB instruction L1 cache and 256/512 kB L2 cache
Chipset	Intel US15W (Poulsbo) - 400/533 MHz Front Side Bus (FSB), One DDR2-400 / DDR2-533 unbuffered DDR-SDRAM (SODIMM form factor) up to 2 G
Graphic controller	Integrated Intel GMA500 graphic controller with dual independent display support, supports Ultra DMA (UDMA5), onchip Video Graphics Array (VGA), hardware acceleration of following video decode standards: H.264, MPEG2, MPEG4, VC1 and WMV9
Audio controller	Integrated Intel® High Definition audio controller (HD audio) with line in/out, mic in and digital audio output
USB support	Onchip Universal Serial Bus: Six ports are capable to handle USB 1.1 (UHCI) and USB 2.0 (EHCI), one port alternatively supports USB client functionality as a peripheral mass storage volume or RNDIS device
Ethernet-support	Gigabit LAN (PCI Express): Intel 82574L, full duplex operation at 10/100/1000 Mbps
Mass storage support	Serial-ATA (PCI Express): JMicron JMB362 Two Secure Digital I/O / MultiMedia Card (SDIO/MMC) controllers, onchip Secure Digital I/O / Multimedia Card (SDIO/MMC), fully compliant with SDIO revision 1.1 and MMC revision 4.0
Other Interfaces	Two PCI Express ports (x1 lanes) Low Voltage Differential Signaling (LVDS) flatpanel interface supports single clock Digital I/O (CPLD): four inputs and four outputs, +3.3V signal level
Temperature monitoring	One onchip thermal sensor and one remote temperature sensor (CPU), SMBus Winbond W83L771W
BIOS	AMI Bios, 1 MB Flash BIOS
Realtime Clock	Supported, requires external battery

Table 10: Kontron pITX SBC functional specifications

3.1.3 Block Diagram

The following figure shows a block diagram of the functional elements of the Kontron pico-ITX Single Board Computer (SBC) and depicts the onboard connectors for connectivity of peripheral units [15]:

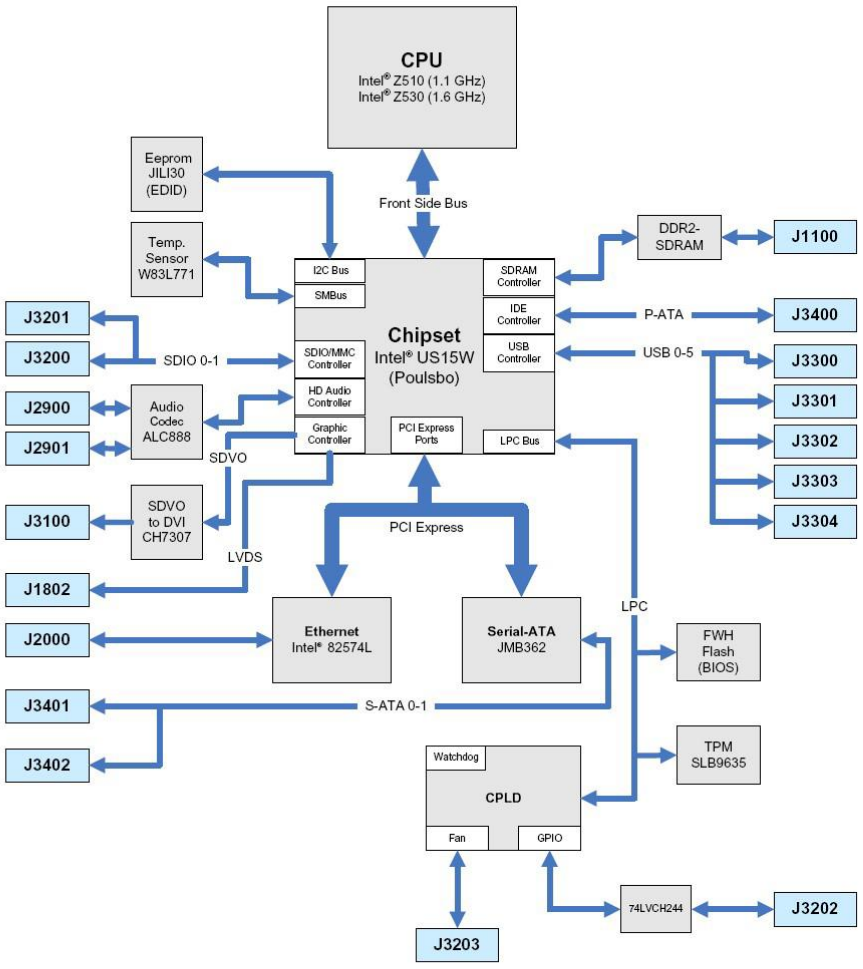


Figure 3: Kontron pITX SP embedded computing platform, block diagram

3.1.4 System setup and power requirements

For the evaluation of the hardware architecture and the feasibility tests performed in D4.7, the following system setup for the Kontron pico-ITX SBC has been used:

Main board	Kontron pITX SBC, 1,GHz, Atom CPU
RAM	1 GB SO-DIMM module, DDR2 RAM
Mass storage	64 GB Kingston “SSD-now” series Solid State Disk drive, 2.5”, connected via S-ATA
Peripherals	USB-Keyboard, USB-Mouse, optional MIMO USB display with touchscreen
Network connection	LAN, 1 GB Ethernet
Power supply	External 12V DC adapter (60W), internal micro ATX DC converters (12V, 5V, 3,3V)

Table 11: System setup for the Embedded Platform power requirement evaluation

Under normal load conditions, an average power consumption of about 10 Watts could be measured using the setup shown in Table 11 . For details about power requirements and measurements performed in different states and load conditions please refer to D4.7.

Optional LC Display with Touchscreen

As defined in the Description of Work [1] and in the hardware requirement section (see 2.1.1), a modular display solution for the AsTeRICS Embedded Platform is desired, because some use cases may require a display and/or a touchscreen for user interaction, but others may not. In the first prototype of the AsTeRICS system, a MIMO 720-S USB pluggable display will be used as optional LC display with touchscreen [16]. The MIMO720-S features a small screen with a dedicated graphics processor and touch functionality. A USB connection is used for data transferring and doe display power supply. The included drivers can be configured to use the MIMO as secondary or main display device in Windows operating systems. The touchscreen driver emulates standard mouse functions.

The price of the MIMO 720-S is affordable (around 120€), and the power consumption is low compared to display solutions of similar size and brightness. The following table shows the specifications of the MIMO 720-S [16].

Figure 4: MIMO 720-S USB pluggable LC display with touchscreen

Display size	7 inch
Display resolution	800 x 480 pixel
Brightness	350 cd/m2
Contrast ratio	400:1
Connections	USB 2.0
Dimensions	7" x 4.5" x 0.8"
Touchscreen	resistive sensor
Monitor Pivot	90 degrees
Weight	< 1 pound
Power requirements	USB powered, < 500mA @ 5V (2,5 W)
additional features	Integrated stand and cover

Table 12: Specifications of the MIMO 720-S USB pluggable LC display with touchscreen

3.2 Communication Interface Modules

Due to the fact that no embedded computing platform exists which provides all the needed physical interfaces to connect the sensors and actuators listed in section 3.1, special Communication Interface Modules (CIMs) will be designed.

Nr.	Name	Main purpose	Provided Interface
CIM 1	Core expansion module	Basic set of interfaces always attached to the emmedded computing platform	More USB interfaces, GPIO, UART, status display, WiFi, BlueTooth, configurable user buttons
CIM 2	Zigbee-CIM	Enobio interfacing, remote switch interfacing	Zigbee CIM
CIM 3	GPIO-CIM	hotkeys and selection buttons of platform user input (digital switches) user output (custom control)	General purpose input and output ports (digital input or output) with ESD, overvoltage and shortcut protection
CIM 4	ADC-CIM	Read analogue values (strain gauge, accelerometer)	Analog / digital conversion (e.g. 10bit resolution @ 256 Hz)
CIM 5	DAC-CIM	Ouput of analog voltages, capability of driving inductive loads (e.g. pneumatic valves for the gripper module)	Digital / analog conversion 0-25V, (optionally via separate supply voltage), 5 ports, two of them up to 5 Watts

Table 13: Hardware interfaces provided by Communication Interface Modules

3.2.1 Core Expansion Module

The chosen computing platform provides only limited basic set of interfaces like 6 USB ports, Ethernet interface and LCD DVI output. There is no system bus for expansion therefore the key component of the expansion module will be an USB hub with sufficient amount of USB ports to integrate the following additional interfaces plus 7 USB interfaces available to connect other CIM, sensor and actuator modules.

Next to the USB hub, the core expansion module will offer:

- **8x GPIO** Eight digital general purpose input and output pins will offer to integrate e.g. few user configurable buttons and/or status LEDs directly in the core of the AsTeRICS platform.
- **2x UART** Two serial ports could be used to directly connect or even integrate e.g. the Zigbee-CIM or another CIM/sensor/actuator via UART interface.
- **WiFi** One of the requirements is to provide at least 802.11b wireless network interface for wireless connection from the AsTeRICS configuration suite,
- **BlueTooth** Some sensors like Sweety! Switches or actuators like mobile phone need BlueTooth connectivity.
- **Status Display** Many AsTeRICS installations will be without any LCD display. It will be nice if a small one or two text lines or narrow graphical status display in the core box will show the current status of the system.

3.2.2 Zigbee-CIM

The IEEE 802.15.4 ZigBee CIM is necessary if user wants to connect the Enobio device wirelessly. Also other functionality could be implemented in future like e.g. a remote light switch. The ZigBee CIM will be connected via USB or UART interface. There is also a chance to integrate this interface directly in the core expansion module.

3.2.3 GPIO-CIM

A basic set of 8 digital general purpose input and output pins will be available directly in the core expansion module but more can be needed. Therefore an expansion module with another 8 (16) input and 8 (16) output pins will be developed. A standard ESD/shortcut/overvoltage protection will be on all I/O pins. The output pins should offer at least open-collector and programmable 5V pullup function. Optionally also relays could be assembled on this board.

3.2.4 ADC-CIM

Some sensors provide analogue voltage, current output or other electrical value like strain gauge or analogue accelerometer. To connect such sensors a universal analogue-to-digital converter module is needed. For the strain gauge also an in-built excitation source will be necessary. The accuracy and maximum frequency needed for these sensors are not so high; therefore a standard 10/12 bit ADC with sampling frequency above 1 kHz should be sufficient. The module will provide 4 channels with basic range 0-5 V adjustable via configurable dividers.

3.2.5 DAC-CIM

To control an actuator with analogue interface a digital-to-analog converter CIM is needed. The developed DAC CIM will offer five 0-25V analog voltage outputs. If external power supply is connected, two of them will provide also programmable current limit up to at least 5 Watts each.

Note: The GPIO, ADC and DAC CIMs will be probably integrated together on one common PCB board. Then two variants will be manufactured – one cheaper with GPIO only, one with full GPIO/ADC/DAC set. This will be defined by the components assembled on board and the internal firmware.

3.3 Custom Sensor and Actuator Modules

3.3.1 Universal HID Actuator

The Universal Human Interface Device (HID) actuator is a microcontroller-based peripheral device developed in course of the AsTeRICS project, which connects to a PC (laptop-, netbook or desktop computer) via a standard USB interface. The Universal HID actuator is capable of emulating different standard user interaction devices (mouse, keyboard and joystick) without additional driver-installation on the PC, given that the operating system supports the standard USB Human Interface Device classes (HID-classes). This is true for every modern operating system including different Windows versions and Linux distributions.

The purpose of the Universal HID actuator in terms of accessibility is that the AsTeRICS Embedded Platform can generate mouse, keyboard or joystick control commands out of various sensor data, and the Universal HID actuator emulates the desired PC input device. Thus, a mapping of (pre-processed) sensor data to different standard Human Computer

Interaction devices can be performed. Furthermore, dedicated Assistive Software products can use AsTeRICS sensor information via a standard and well supported process, e.g. by using API calls of the operating system to get joystick button states or –positions.

In the first AsTeRICS prototype, the Universal HID actuator will be interfaced by a wired connection, where the second AsTeRICS prototype will provide a wireless connection to the HID actuator dongle, which then plugs into the USB port of the PC like a commercial off-the-shelf USB memory stick.

As a hardware platform for the HID emulation, a suitable USB-capable 8-bit microcontroller will be chosen from the available products which have been described in D2.4 [3]. Most probably, the PIC Low-Pincount USB Development Kit or the Atmel USB-Key will be used as suitable development platform. For the second Prototype, a custom PCB including the microcontroller and a wireless solution will be developed.

The following table shows features of the Universal HID actuator in the different development stages of PT1 and PT2:

HID actuator feature	**Prototype – 1**	**Prototype -2**
Mouse emulation via USB HID class	available	available
Keyboard emulation via USB HID class	-	available
Joystick emulation via USB HID class	available	available
Connection to AsTeRICS Embedded Platform	wired e.g. via RS232/UART	wireless e.g. via Bluetooth or ZigBee
Concurrent use of different HID classes	no	yes

Table 14: Features of the Universal HID actuator module at different design stages

In a particular AsTeRICS configuration, the desired function of the Universal HID actuator is defined in the AsTeRICS Configuration Suite by selecting the according emulator SW-plugin. This selection affects the firmware configuration of the HID emulator dongle when the model is deployed to the runtime system. Figure 5 illustrates this process in a setup where the joystick firmware profile is activated by a deployed joystick actuator plugin:

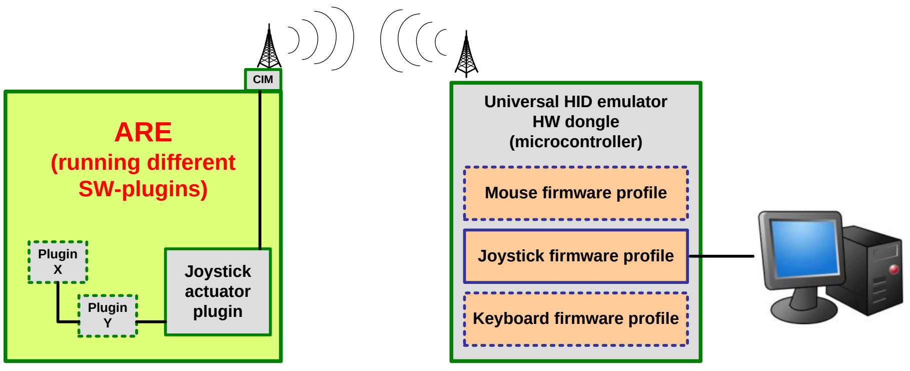

CIM
ARE
(running different SW-plugins)
Plugin X
Plugin Y
Joystick actuator plugin
Universal HID emulator HW dongle (microcontroller)
Mouse firmware profile
Joystick firmware profile
Keyboard firmware profile

Figure 5: Firmware profile selection for desired HID device

The Universal HID actuator module can be used in various user interaction tasks and modalities. The following diagram shows three different use cases including the HID actuator wireless version (Prototype 2):

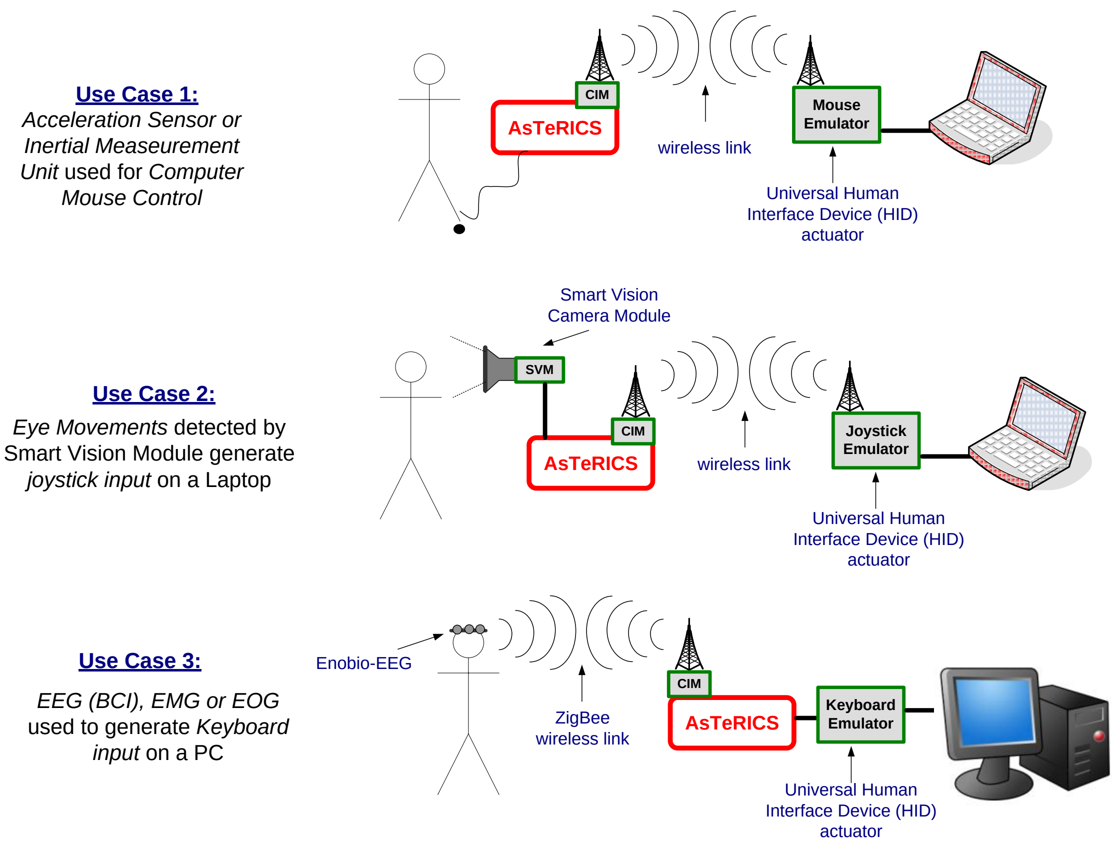

Use Case 1:
Acceleration Sensor or Inertial Measeurement Unit used for Computer Mouse Control
CIM
AsTeRICS
wireless link
Mouse Emulator
Universal Human Interface Device (HID) actuator
Use Case 2:
Eye Movements detected by Smart Vision Module generate joystick input on a Laptop
Smart Vision Camera Module
SVM
CIM
AsTeRICS
wireless link
Joystick Emulator
Universal Human Interface Device (HID) actuator
Use Case 3:
EEG (BCI), EMG or EOG used to generate Keyboard input on a PC
Enobio-EEG
ZigBee wireless link
CIM
AsTeRICS
Keyboard Emulator
Universal Human Interface Device (HID) actuator

Figure 6: Use cases including the Universal HID actuator

3.3.2 Generic Switches

A small set of generic switches shall be integrated with few status LEDs and GPIO CIM into one box. If a custom designed switch is not needed, this can reduce the size of the AsTeRICS system.

3.3.3 Accelerometer

3-axis, ultralow power accelerometer sensor ADXL345 attached to USB port will be provided. The accelerometer has I2C and SPI digital interface but if possible, not only the USB-I2C converter but also joystick/pointing device emulation will be supported in the hardware.

The used ADXL345 sensor provides user selectable measurement ranges up to ±16 g. It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity in all three axes while consuming 23 µA in measurement mode and only 0.1 µA in standby.

3.3.4 Strain Gauge

A strain gauge sensor will be prototyped or integrated with the ADC or ADC/DAC CIM to test the functionality. The pressure or tension translated to the resistivity difference of the sensor will be measured by the CIM module and sent to the platform for further processing. If the accuracy of the ADC used in ADC(/DAC) CIM is not sufficient then it can be improved in PT2.

3.3.5 Pneumatic Gripper actuator

The Pneumatic Gripper actuator provides functionalities for grabbing (tiny) objects. For this a lightweight pneumatic gripper (FIPA GR04.090) is used which can be mounted on a mouthstick (see Figure 7).

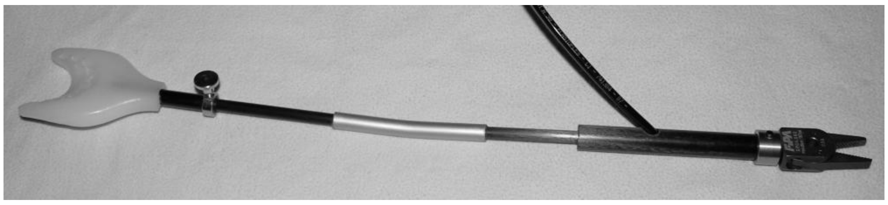


Figure 7: Mouthstick equipped with pneumatic gripper FIPA GR04.090

The gripper itself is controlled by a 3 port solenoid valve (SMC V114 SLOU) which is controlled by the GPIO-CIM or the DAC-CIM.

To support two gripping forces a separate pressure regulator (SMC AR10) and a further 3 port solenoid valve (SMC V114 SLOU) is used. Also this solenoid valve is controlled by the GPIO-CIM or the DAC-CIM.

As source of compressed air an air compressor or scuba diving equipment (compressed air cylinder with single valve and 1st stage regulator) can be used (see Figure 8).

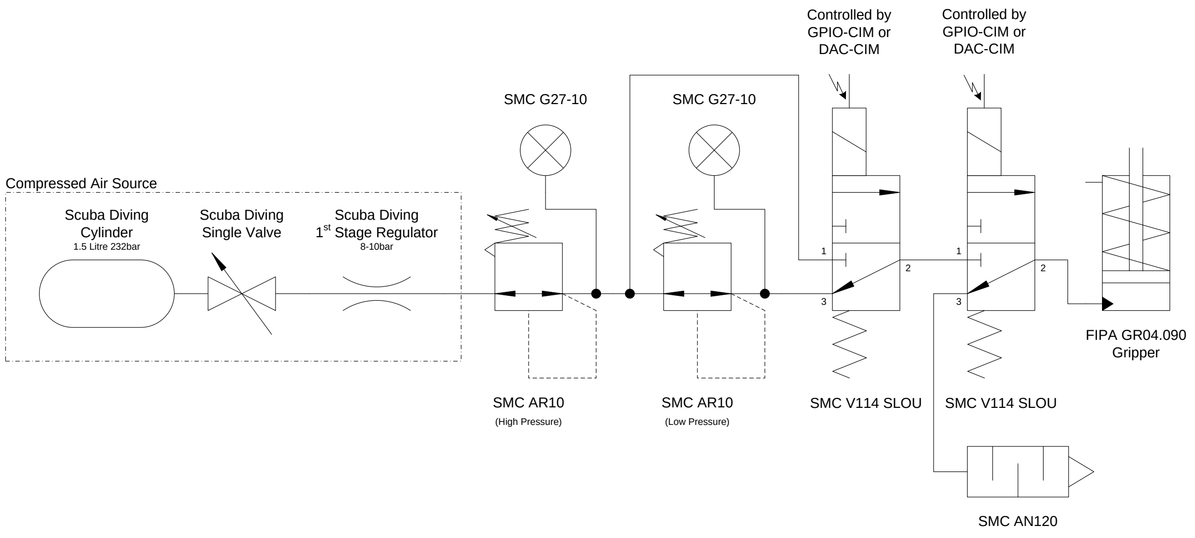


Figure 8: Pneumatic Scheme of the gripper actuator

Specification of the SMC V114 SLOU 3 port solenoid valve:

Operating Pressure:	0 to 0.7 MPa
Flow Characteristics	1→2: 0.037 dm³/(s bar); 2→3: 0.054 dm³/(s bar)
Coil related voltage DC	5V

Power consumtion	0.35W

Table 15: Main Characteristics of the SMC V114 SLOU

3.4 Smart Vision Module

3.4.1 Smart Vision Module (SVM) Concept

The Smart Vision Module (SVM) is one of the two multi-purpose input modules (BCI Evaluation Suite being the other one) that will be developed during WP3 and WP4 and integrated in the AsTeRICS system. The Smart Vision Module will be essentially based on image processing and computer vision algorithms in order to provide means of video-based interaction for AT purposes. Within AsTeRICS project, the SVM will be limited to the study and the implementation of a gaze interaction system.

Based upon an analysis of the state-of-the-art eye-gaze trackers (cf. deliverable D2.4 [3]) in conjunction with user requirements (cf. WP1, especially D1.1 and D1.3), the SVM adequate functions were defined. The goal of the Smart Vision Module will be twofold (cf. Figure 9). Indeed, two types of eye-gaze trackers will be built within the project in order to answer to different needs of AsTeRICS project targeted end-users.

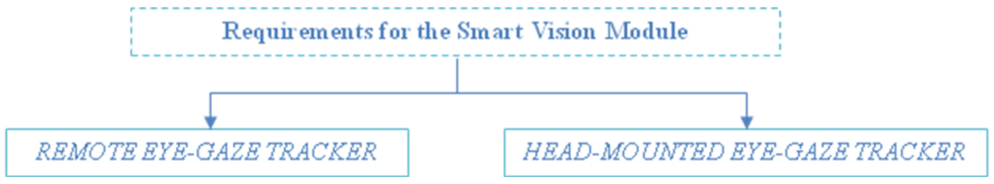

Requirements for the Smart Vision Module
REMOTE EYE-GAZE TRACKER
HEAD-MOUNTED EYE-GAZE TRACKER

Figure 9: Smart Vision Module functions

Improved existing and original algorithms for gaze detection and tracking, and software will be designed and tested in real scenarios (as defined in WP1). A brief overview of new existing eye and gaze tracking approaches is given in [24].

An open-source and scalable software framework will be created. Consequently, it will be possible to extend the initial library of eye-based man-environment interactions to interactions with any part of human body (head, nose, finger, shoulder & elbow, etc.).

3.4.2 SVM Overview

Two prototypes of video-based eye-gaze tracker will be designed and built: remote and head-mounted, both in charge of estimation of the gazed 3D point (point-of-regard, PoR). Why two eye-gaze trackers? Because remote and head-mounted eye-gaze trackers have their own advantages and drawbacks and end-users have different needs for interaction (a particular configuration might be better adapted to certain kinds of disabilities).

The remote eye-gaze tracker is a non-intrusive solution and offers reasonable accuracy for pointing (selection) operation. Its realisation is a challenge, as several complex constraints, such as head movements, prior geometry information, camera-screen referentials, calibration, illumination invariance, etc., must be conveniently solved.

The head-mounted eye-gaze tracker is more intrusive, but is proved to be more accurate than a remote eye-gaze tracker (typically 1° of visual angle or better) and have the advantage of fixed camera-to-head displacements which allows for larger head movements (a camera-head transition matrix is known).

For both solutions, the goal will be to perform mouse emulation thanks to gaze. By mouse emulation, one means that it is possible to do mouse displacements and mouse clicks (via dwelling time).

3.4.2.1 SVM First Prototype (PT1)

Remote eye-gaze tracker

The remote eye-gaze tracker will consist of a digital camera and a personal computer (cf. Figure 10). The choice of the camera will depend on different parameters such as the accuracy (web cameras have low resolution), the position of the camera according to the one of the computer screen, etc. The final architecture will be selected after experimental evaluation in typical scenes.

The camera will be in charge of capturing images from the head of the user. Then, the head and the eyes will be detected via image processing and finally, the gaze will be estimated.

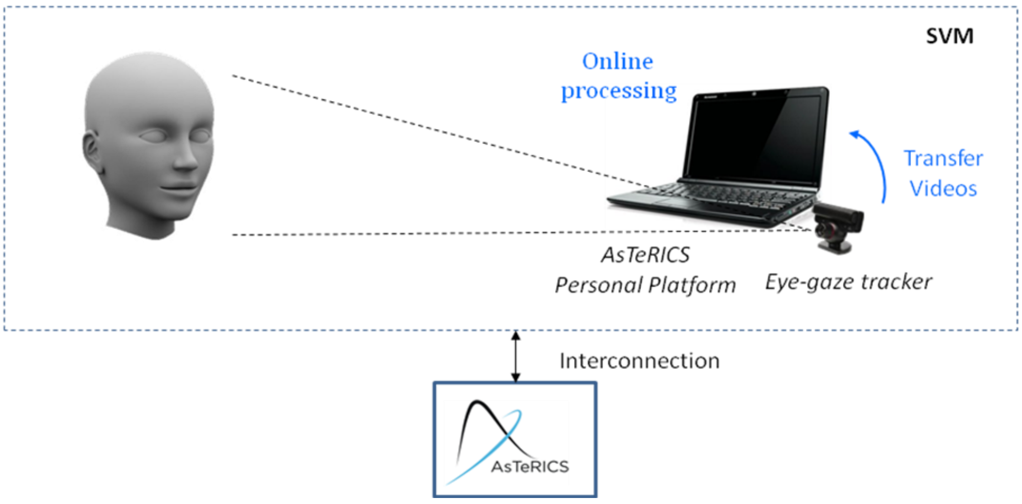


Figure 10: Remote eye-gaze tracker scheme

Head-mounted eye-gaze tracker

The head-mounted eye-gaze tracker will be built around two digital cameras, eye and scene cameras, an inertial measurement unit (IMU) and a personal computer (cf. Figure 11).

The eye (respectively scene) camera will be in charge of capturing images of the eye (respectively computer screen). The inertial measurement unit will provide acceleration and rotation rate measurements of the camera (and thus, the head).

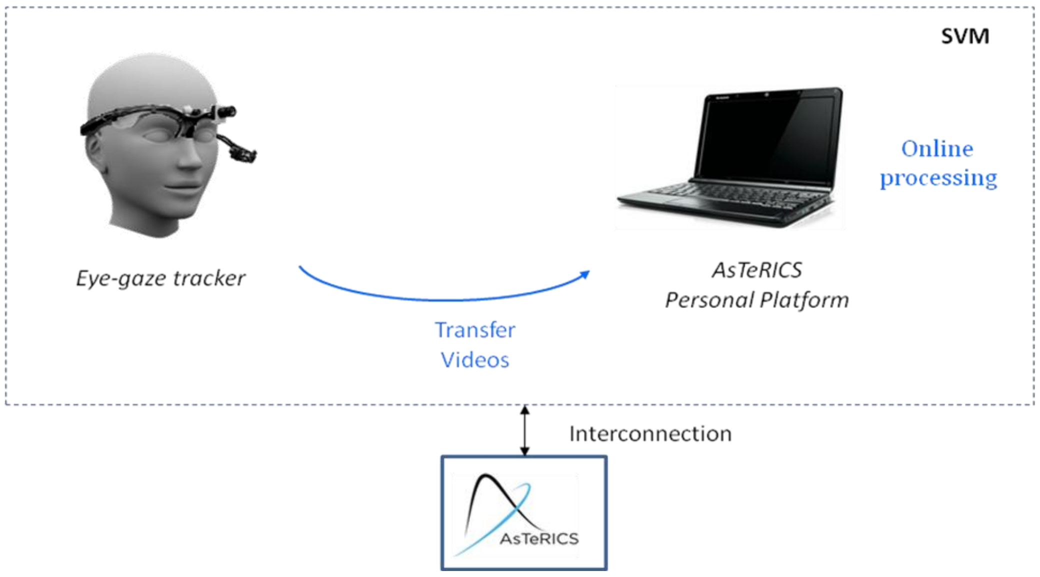


Figure 11: Head-mounted eye-gaze tracker scheme

The proposed head-mounted eye-gaze tracker is also designed with the intention that it can be used for other (future) applications. For example, this design could be seen as a first step toward mobile eye tracking and hence, well-suited for 3D environment interaction (which is a challenging and interesting topic not only in assistive technology).

3.4.2.2 SVM Final Prototype (PT2)

After an evaluation test of the first prototype (PT1), the proposed remote and head-mounted systems will be improved. Improvements of two types will be made: functional and performance, and will lead to both software and hardware modifications (if necessary).

In addition, to improve the performances of the eye-gaze trackers, the final prototype will allow using a virtual keyboard and/or menu-buttons displayed on computer screen to give user ability of entering text or menu-based interaction. Moreover, it will give the possibility to compare the accuracy of the different types of eye-gaze trackers that will be built during the project.

3.4.3 Remote Eye-Gaze Tracker: Hardware Design

The remote eye-gaze will be designed such that it can be easy to install, transport and use. In designing a remote eye-gaze tracker, three main hardware issues have to be solved: choice of the camera, prior geometry constraints, light sources (cf. Table 16).

Hardware issues	**Purpose**
Camera choice	To identify the face and the different eye features (a large field of view is required)
Prior geometry constraint(s)	To obtain the position of the camera relative to the computer screen
Use of light sources	To allow for head movements

Table 16: Remote eye-gaze tracker hardware issues

3.4.4 Remote Eye-Gaze Tracker: Software Design

Gaze estimation principle

Remote eye-gaze trackers usually proceed in 3 main steps (cf. Figure 12) to compute the point-of-regard (PoR): face detection, eyes detection and gaze estimation.

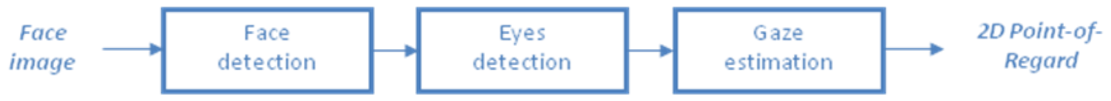


Figure 12: Remote eye-gaze tracker processing pipeline

The principle of remote eye-gaze tracking is described in Figure 13. First, a single camera records images of the user and tries to detect his face. Once the face is detected, the system attempts to localize the left and right eyes of the user. Then, in every eye window, eye features are usually localized and their position computed.

Some available existing VOG (video-oculography) shape-based techniques (for fine eye features detection) combined with appearance-based (for coarse face and eyes detection) techniques will be evaluated in real scenarios in order to select the most appropriate one for remote eye-gaze tracker. If necessary, these techniques will be adapted to our final scenarios.

The main approaches for fine eye feature detection that will be investigated are those based on the extraction of simple (iris, dark/bright pupil, limbus) or complex (eyelids, eye corners, eyebrows, cornea reflections) eye characteristics using their models.

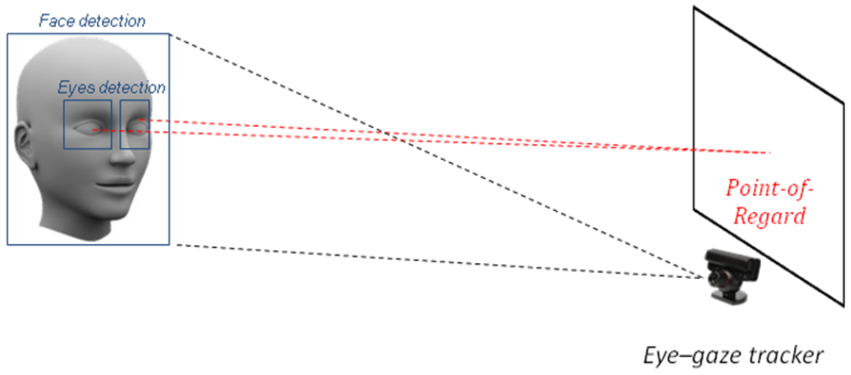


Figure 13: Remote eye-gaze tracker principle

To estimate the gaze in a remote configuration, two approaches exist: pupil-glint vector or model-based. For the pupil-vector method, a mapping is computed between eye features, namely the pupil-glint vector and the screen coordinates. The model-based method estimates a gaze direction vector according to an eyeball model and computes the point-of-regard (PoR) by intersecting the line supporting the gaze direction vector with an object in the environment, usually a computer screen.

Tracking of both eyes allows improving the accuracy of the gaze estimation through the formulation of the epipolar stereo geometry. However, if only one eye is detected, it is possible to estimate the gaze, but to do so, the system must identify if the detected eye corresponds to the left eye or the right eye.

The main contribution will be to design gaze estimation using image processing and vision algorithms only.

3.4.5 Head-Mounted Eye-Gaze Tracker: Hardware Design

3.4.5.1 Hardware Architecture

The head-mounted system will be built around commercial-off-the-shelf (COTS) components and will consist of the following sensors: an eye camera, a scene camera and an inertial measurement unit (IMU)

In designing a head-mounted eye-gaze tracker, the main hardware issues that need to be solved are listed in Table 17.

Hardware issues	**Purpose**
Head-mounted support	To mount and fix the eye-gaze tracker on the head
Synchronization/stabilization	To acquire data at the same instance and frame rate (no delay) and to stabilize acquired images
Eye camera choice	To identify different eye features and perceive sufficient details
Scene camera choice	To get a large view of the environment the user is looking at
Use of a light source	To enhance the contrast between pupil and iris and get a static eye reference point

Table 17: Head-mounted eye-gaze tracker hardware issues

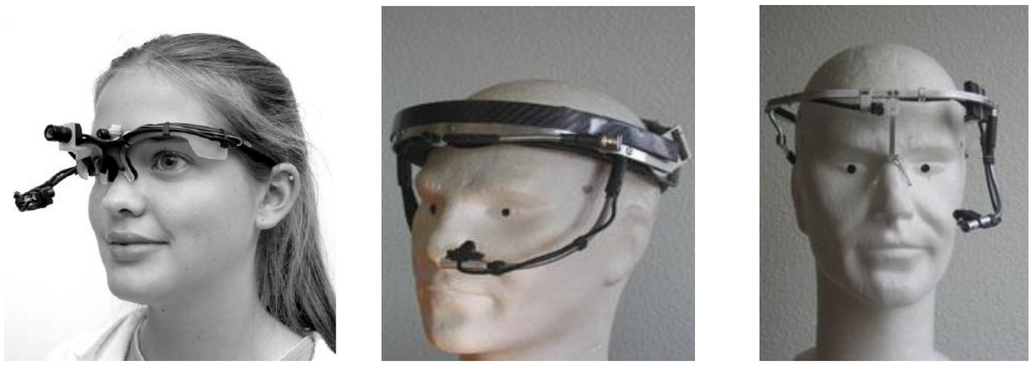
Head-mounted support

Most of existing head-mounted eye-gaze trackers are based on one of the following configurations (cf. Figure 14): pair of glasses, headband-mounted solution or hybrid solution.

Pair of glasses (cf. Figure 14 a))*:* This configuration is not well-suited for persons wearing glasses as they would be bothered by the arms of the head-mounted system. However, it allows guaranteeing stability to the system thanks to a joining nose bridge. Low-cost pairs of safety glasses could be used.

Headband-mounted solution (see Figure 14 b))*:* This configuration is well-suited for people wearing glasses. However, large head movements will make the system less stable than an eye-gaze tracker based on a pair of glasses.

Hybrid mounting for eye trackers (see Figure 14 c)*):* It is also possible to adopt a hybrid configuration combining the advantages of both configurations described above by adding a nose bridge to the headband configuration in order to overcome the problem of stability.



a) Pair of glasses [17] b) Headband-mounted [25] c) Hybrid [25]

Figure 14: Head-mounted supports

The main difficulties in building the head-mounted support are:

- to make it as lightweight as possible (so it can be supported during a long period of time)
- to avoid obstructing the field of view of the user when mounting the eye camera.

Synchronization and stabilization

One of the main issues in eye-gaze tracking is the synchronization of the data acquired by the different sensors (cameras and IMU). Indeed, synchronization refers to the proper coordinate mapping of the eye tracker's reference frame to the application responsible for generating (and stabilization of) the visual stimulus that will be seen by the user [22].

In the literature [26], the most frequent solutions for multiple camera synchronization are the following:

a) **Special-purpose hardware**: it usually requires a synchronization platform. This solution allows delivering external synchronization signals in order to trigger cameras. Camera manufacturers such as Point Grey Research Inc or IDS imaging propose to synchronize the image acquisition of, respectively, multiple firewire or USB cameras.

b) Post-processing synchronization algorithms: these methods usually work on unsynchronized video sequences and estimate the temporal offset between them. Some of them rely on tracking interest points and matching them over time. The main drawback of post-processing algorithms is that they cannot be applied in real-time and are sensitive to occlusions.

c) Software-based methods: most of the methods solve the problem of synchronization by triggering cameras via software, by calculating the latency or sending a start pulse of recordings, just to name a few. The drawback of these methods is that they are designed for multi-camera network i.e. that more than one computer is available.

d) **Network synchronization**: this method works by interconnecting several computers and creating a local area network (LAN) and applying a software-based synchronization.

For AsTeRICS project, the special-purpose hardware approach will be adopted as it is the only working solution to properly and precisely synchronize the cameras and the IMU. A USB interface (cf. Figure 15) will be developed by IMA and will allow attaching two digital cameras and an inertial measurement unit. The inertial data will be acquired in parallel and at the same rate as these of the two cameras.

The result will be a platform that leverages the utility of USB to provide an efficient and accessible means of interfacing custom video and inertial sensors with personal computers.

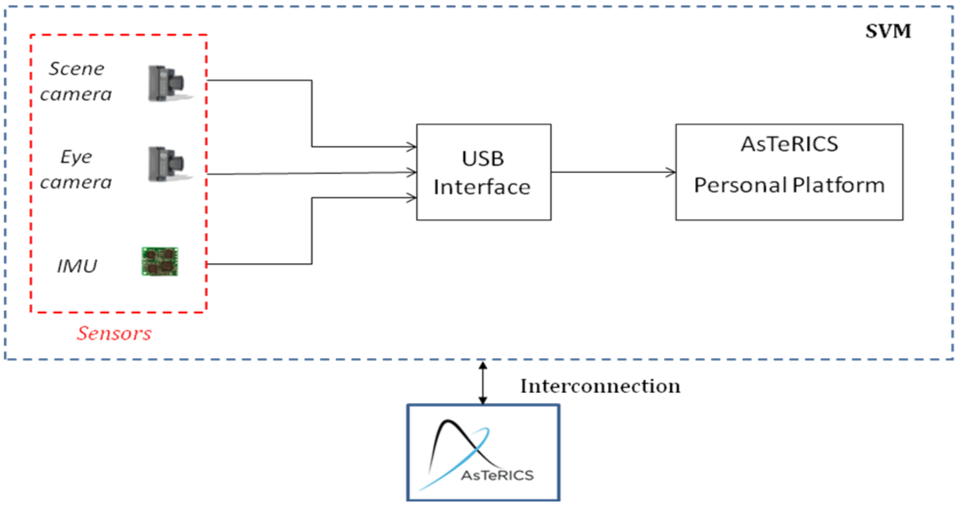


Figure 15: SVM Connections/Synchronisation scheme

3.4.5.2 Camera Modules

The choice of the two cameras for an eye-gaze tracker is essential to achieve good accuracy and low latency. Two cameras have to be chosen from the market: an eye and a scene camera. The features of both cameras are guided by the goal(s) of the application and usually depend on what data they acquire. Main features of cameras which should be considered are: sensor technology, chromaticity and sensitivity to light, resolution, field of view and focal length, shutter type, camera interface.

Sensor technology - CMOS vs. CCD: Today's cameras are built with one of the following technologies:

- CMOS (complementary metal oxide semiconductor)
- CCD (charge coupled device) sensor

Both technologies have their own advantages and disadvantages. CMOS sensors tend to be smaller than CCD sensor and consume less power, but they still generally require companion chips to optimize image quality, increasing cost and reducing the advantage they gain from low power consumption. However, both CMOS and CCD technology accomplish the same tasks of capturing images with almost equal quality.

Chromaticity and sensitivity to light: Depending on the chromaticity of the sensor, either monochrome or colour, information on the image itself is different. Indeed, when using monochrome cameras, only one luminance channel is available for processing the image whereas colour cameras introduce 3 channels (Red, Green and Blue). Hence, upon the information we acquire from the camera, processing can be different. Colour cameras are well-suited for application where object detection/tracking is important because working in different colour spaces can provide more details about the appearance of the object itself.

Moreover, the chromaticity of the sensor also influences the sensor sensitivity to light. In fact, monochrome cameras are traditionally more sensitive than colour cameras. Sensitivity to light plays an important role in environment with poor lighting conditions as it can provide more clarity to the image.

Resolution: Resolution describes the quantity of details an image holds. Thus, the higher the resolution is, the more details a camera can capture.

Field of view and focal length: The field of view (FOV) describes the area of vision of a given scene acquired by the camera's sensor and is directly linked to the focal length. There are different types of FOV: super-telephoto (<1° to 8°), telephoto (10° to 15°), standard (25° to 50°), wide-angle (60° to 100°) or fisheye (up to 180°). Furthermore, the larger the FOV is, the higher optical distortions, especially radial distortions, are. However, these distortions can be corrected via image processing.

Shutter type: There are mainly two shutter types: global and rolling. Global shutter allows acquiring a single snapshot at a fixed time. On the other hand, rolling shutters scan across the frame and the scanned lines can be recorded at a different time. Hence, a rolling shutter can introduce distortions (spatio-temporal aliasing) due to fast motion.

Camera interface: The camera interface allows transferring images from camera sensor to the processing platform (such as a PC). Usually, the type of interface influences the physical dimensions and weight of the camera itself.

Available cameras from the market are mainly using the following standard data interfaces: USB, Firewire (IEEE1394), Composite, CamLink, and GigE. USB, firewire and composite interfaced cameras are the most common cameras available from the market. Cameras with composite video output offer the output signal of analogue type, usually deliver images of poor quality and hence, would affect the image processing. CamLink and GigE interfaced cameras are usually quite heavy and have high physical dimensions and thus, are not well-suited for building a wearable device.

Weight and physical dimensions: A camera's size and weight should be as small as possible, still providing the desired image quality.

Cameras of current head-mounted eye-gaze trackers

In [19], Babcock *et al.* from the Rochester Institute of Technology (RIT) presented an eye tracker using two analogue cameras and an infrared LED (cf. Figure 16). They used two micro-lens cameras, namely the PC206XP (a black and white CMOS imager with 380 lines of resolution) and the PC53XS (a colour CMOS imager) from the company SuperCircuits. However, these inexpensive cameras provide low resolution images which is mainly due to the analogue output video of the cameras.

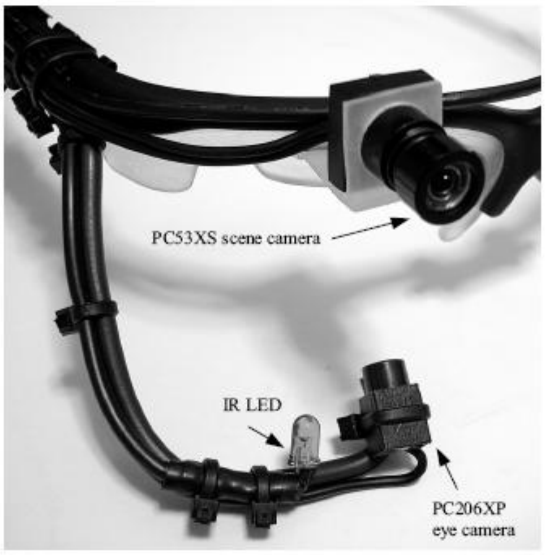


Figure 16: RIT cameras [19]

Based on the design of [19], Li *et al.* investigated various camera candidates for their OpenEyes eye tracker in [17]. They mentioned that the analogue cameras they found were quite expensive to use in eye tracking system, required AD converter or were not suitable for mobile applications. Thus, they moved to digital cameras. They tried to find USB cameras, but their bandwidth limited the resolution and the frame rates. Finally, they investigated IEEE-1394 cameras and opted for the Unibrain Fire-I camera (cf. Figure 17). However, one of their problems was to try to decrease the degree of noise when capturing the images and these cameras are initially attached to a board (they had to detach the cameras from the board). They concluded by mentioning that high resolution cameras would give more accurate eye tracking and the use of higher speed cameras would be more suitable in decreasing motion blur effect.

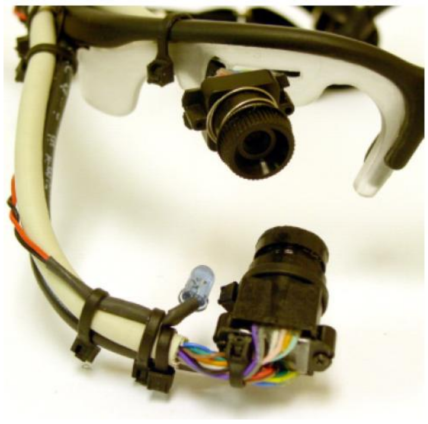


Figure 17: OpenEyes cameras [17]

In [18], Yun *et al.* chose two analogue cameras for their EyeSecret eye tracker (cf. Figure 18): the WAT-704R (a miniature monochrome analogue CCD imager) and the WAT-240 (a miniature colour analogue CCD). However, they do not provide information about image quality.

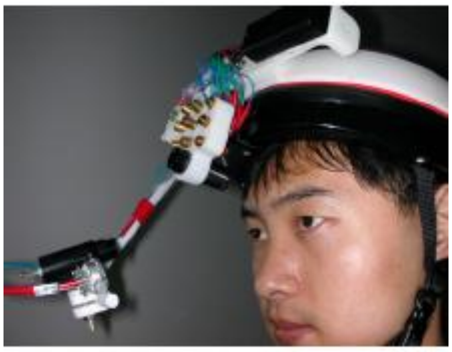


Figure 18: EyeSecret cameras [18]

In [20], Ryan *et al.* pursued the open-source project “openEyes” [17] at Clemson University and improved the algorithm in order to switch between pupil and limbus tracking under variable lighting conditions. They replaced the openEyes cameras, PC206XP and PC53XS, with the digital video minicams called Camwear Model 200 from DejaView and recorded videos using MPEG-4 codec (cf. Figure 19). However, they also provide no details about image quality.

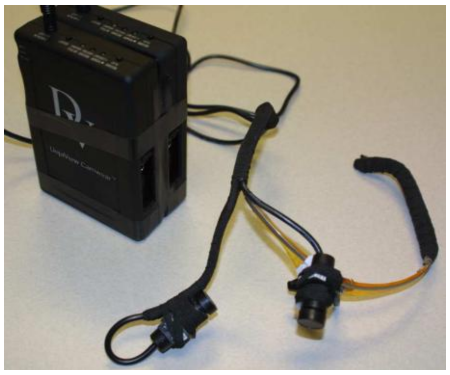


Figure 19: Clemson eye tracker [20]

Interesting cameras for a head-mounted eye tracker

Some commercially available cameras are listed in Table 18 and Table 19. Due to the synchronization constraint (cf. section 3.4.5.1), the choice of the cameras is limited to a certain range of cameras, especially digital cameras. Indeed, eye trackers usually are built with analogue cameras (with composite video output) because they often are small and are widespread in the market; however, they introduce some artifacts when acquiring videos.

For the *eye camera*, it is very difficult to find high-quality miniaturized digital parallel cameras. In Table 18, some cameras are listed keeping in mind the synchronization issue. Another possible solution (that needs to be tested) could be Wafer Level Cameras (WFC), but they usually offer poor light sensitivity.

	MBS032M [27]	**MBS032C [27]**	**00-C4DCM-01 [28]**	**UI-1226LE-M [2929]**	**UI-1226LE-C [29]**
Company	Mobisense Systems (FR)	Mobisense Systems (FR)	C4AV	IDS Imaging (DE)	IDS Imaging (DE)
Sensor	CMOS (Aptina)	CMOS (Aptina)	CMOS (Aptina)	CMOS (Aptina)	CMOS (Aptina)

Chroma	Monochrome	Color	Color	Monochrome	Color
Optical Format	1/3 "	1/3 "	1/4 "	1/3 "	1/3 "
Resolution (pixels)	752 x 480	752 x 480	640 x 480	752 x 480	752 x 480
Frame Rate (fps)	60	60	30	87	87
Focal length (mm)	6	6	X	X	X
FOV (°)	X	X	X	X	X
Distortion	X	X	X	X	X
Sensitivity	4.8 V/lux-sec	4.8 V/lux-sec	5 V/lux-sec	X	X
Shutter type	Global	Global Shutter	Electronic Rolling Shutter	Global	Global Shutter
Interfaces	GPIO parallel	GPIO parallel	GPIO parallel	USB (power supply)	USB (power supply)
Dimensions (mm)	26 x 20 x 28	26 x 20 x 28	X	36 x 36 x 20	36 x 36 x 20
Weight (g)	10	10	X	19 (with lens)	19 (with lens)
Price	74 €	77 €	$ 149	~ 300 € (without lens and USB cable)	~300 € (without lens and USB cable)

X : Information not available

Table 18: Main characteristics of some camera candidates for a wearable eye-gaze tracker

The Mobisense Systems cameras seem to be a good compromise between performances and size. IDS imaging cameras are also listed in the table as they could be used to build an eye-gaze tracking system with synchronization via external triggers (provided by the company with a proprietary SDK).

The definitive choice of cameras (according to field of view, light sensitivity, focal length, etc) will be carried out after having evaluating them experimentally in the real scenarios (defined in WP1).

3.4.5.3 Inertial Measurement Unit

The choice of an inertial measurement unit (IMU) for a new eye tracker is important to provide measurements about the head movement. IMUs drift increasing over time can be corrected thanks to computer vision.

Table 19 summarizes the main features of some IMUs which should be considered during a system design; they are: degree of freedom, sensitivity, angular/acceleration rate range, interfaces.

Degree of freedom and measurements: The degree of freedom characterizes the number of coordinates that it takes to specify the position of a system. The measurements are due to inertial forces acting on the sensor. IMUs combine gyroscopes and accelerometers that sense either angular (pitch, yaw and/or roll) and acceleration (x, y and/or z) rates.

Sensitivity: The sensitivity of the gyroscopes or the accelerometers of the IMU is the ratio of the sensor's electrical output (proportional electrical signal) to mechanical input. The gyroscope's sensitivity is measured in mV/° and the accelerometer's sensitivity in mV/g.

Angular/acceleration rate range: The angular (resp. acceleration) rate range characterizes the *rate* of change of *angular* displacement (resp. acceleration) with respect to time.

IMU interface: Typical interfaces are UART, RS-232 or USB.

Weight and physical dimensions: The IMU is usually positioned on the head of the end-user and therefore, should be as small as possible and lightweight (maximum 10 g).

	Inertia Cube 3	**IMU Combo board**	**IMU 6DOF v4**	**Atomic IMU 6DOF**	**IMU 6DOF Razor - IMU**	**CHR-6d**	**ACE_ 6DoF_IMU**
Company	InterSense	Sparkfun	Sparkfun	Sparkfun	Sparkfun	CHRobotics	Sensor Dynamics
Degree of Freedom (DoF)	3	3	6	6	6	6	6
Measurements	yaw, pitch and roll	x, y and yaw	x,y,z, yaw, pitch and roll	x,y,z, yaw, pitch and roll	x,y,z, yaw, pitch and roll	x,y,z, yaw, pitch and roll	x,y,z, yaw, pitch and roll
Angular rate range (°/s)	+/- **1200**	+/- 300	+/- 500	+/- 300	+/- 300	+/- 400	+/- 300
Acceleration rate range (g)	**--**	+/- 1.5	+/- 1.5g, 2g, 4g or 6g	+/- 1.5g, 2g, 4g or 6g	+/- 3	+/- 3	+/- 5
Miscellaneous	Software Developer Kit	Accuracy depends on ADC handling	16/32 bits micropro-cessor, GUI open source available	Sensors reading in ASCII or binary format	Accuracy depends on ADC handling	Dev. Kit included (otherwise 125 €)	--
Interfaces	RS-232, USB	X	Bluetooth	Available via UART, XBee or RF	X	UART	USB, SPI and UART
Dimensions (mm)	26.2 x 39.2 x 14.8	25 x 17.5	50 x 42.5 x 30	35 x 45 x 30	17.5 x 32.5	20.3 x 17.8 x 2.5	23 x 32 x 22
Weight (g)	17	3	X	X	X	1.5	X
Price	~ 1200 €	84.10 €	302.86 €	84.10 €	60.55 €	200 €	320 €

-- : Not included X : Information not available

Table 19: Main characteristics of some IMU for an eye-gaze tracker

InterSense's IMU provides good accuracy but is expensive; it only gives angular rates. Sparkfun's IMU are too invasive. The IMU 6DOF Razor - Ultra-Thin IMU has a low accuracy and depends on ADC handling. The IMU from SensorDynamics are of big size and is too expensive compared to the CHR-6d which provides 6-DoF measurements on a miniaturized 1.5g-board and seems to be accurate enough.

3.4.6 Head-Mounted Eye-Gaze Tracker: Software Design

Gaze Estimation principle

In order to estimate the Point-of-Regard (PoR) on a computer screen (cf. Figure 20), head-mounted systems typically determine the pupil-glint vector (the glint being a single reflection on the cornea), but it is also possible to compute another vector, limbus-RP vector, based on the centre of the limbus and a static eye image reference point (RP).

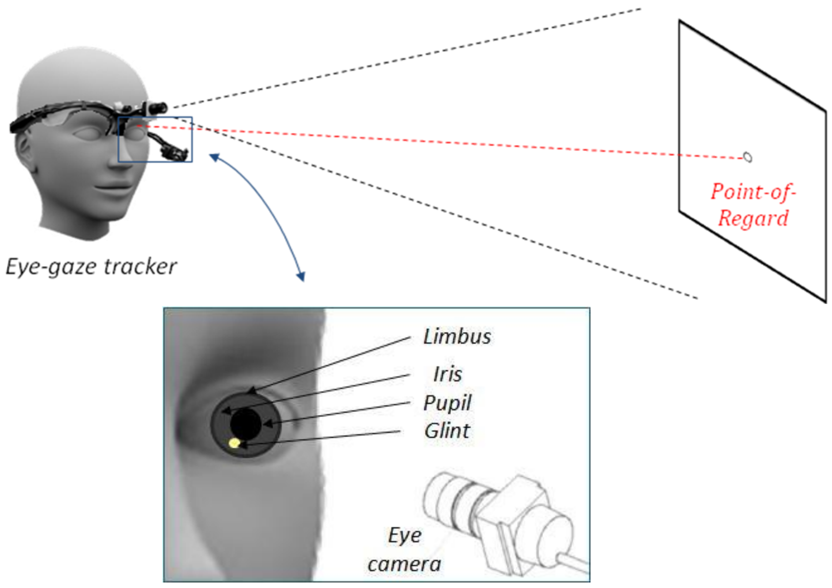


Figure 20: Head-mounted eye-gaze tracker principle

In head-mounted eye-gaze tracking, the image processing chain can be divided in two steps: eye detection and gaze estimation (cf. Figure 21).

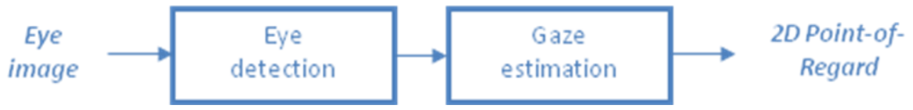


Figure 21: Head-mounted eye tracker processing pipeline

In the images acquired from the eye camera, the coordinate of the centre of the eye need to be computed. A feature-based approach will be used as it is fast and gives good precision. In feature-based approaches, different configurations exist and depend on the use or not of infrared LEDs (IR LEDs); infrared lights allows enhancing the performance of features extraction under varying illuminations. Thus, two methods are possible and are described in Figure 22 (left and right), along with the usual feature-based algorithm (in the centre of the Figure 22).

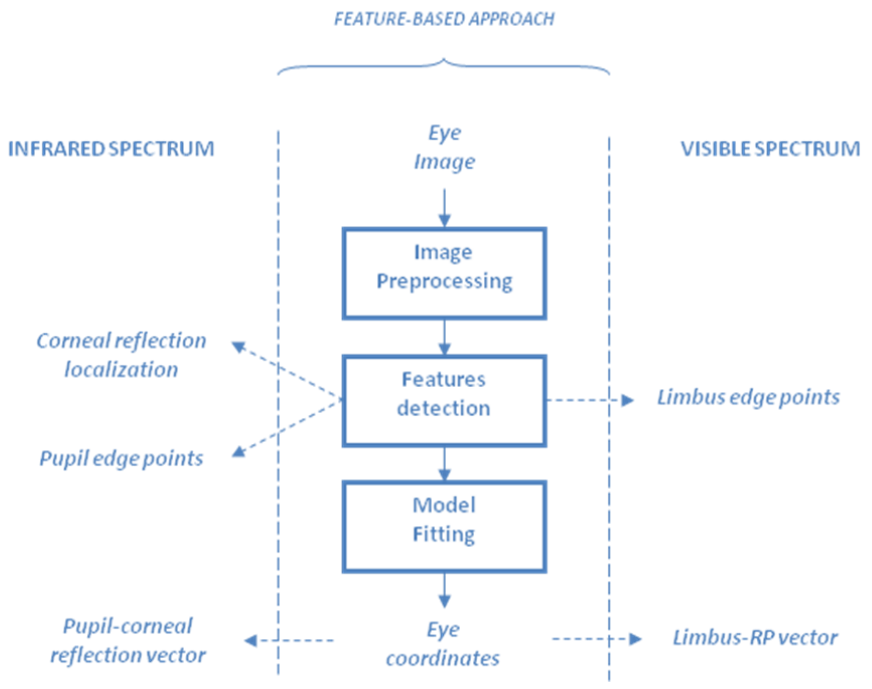


Figure 22: Eye detection flowchart

Thereafter, the goal of the gaze estimation step is to establish a mapping of eye tracker coordinates (here the pupil-glint vector or the limbus-RP vector) to the appropriate range of application. Because in the scenario defined in section, the user needs to interact with the computer, the eye tracker coordinates will be mapped to the coordinates of the computer screen.

3.4.7 Concluding Remarks

Table 20 recalls the functions of every sensor of the Smart Vision Module and consequently, allows establishing a link between hardware and software development:

Configuration	**Sensor**	**Functions**
Remote	Camera	Detect face and eyes and estimate the gaze direction
Head-mounted	Eye camera	Estimate the vector of the pupil-glint (or the limbus centre)
	Scene camera	Establish a mapping between the pupil-glint vector (or limbus centre) and the scene coordinates and eventually detect calibration targets (for semi-automatic calibration)
	Inertial Measurement Unit	Compensate for head movements (image stabilization)

Table 20: Sensors functions

All hardware prototypes will be implemented using off-the-shelf components in order to obtain a low cost solution. All algorithms and corresponding software will be original and will take into account real scenarios in order to provide the appropriate answer to the targeted population.

3.4. Enobio Biosignal Unit Specification

Enobio is a wearable, modular and wireless electro-physiology sensor system for the recording of EEG, ECG and EOG. The transmission of the data collected by its electrodes to a host computer is made through a wireless connection (IEEE 802.15.4) and a receiver (10.5 x 6 x 1.5 cm.), which is provided with the acquisition device.

Enobio interfaces the host computer through 4 data ports and 1 control port. The control port is used to send commands to the Enobio sensor such as sampling start and stop or change the electrode offset signal.

The data ports correspond to the raw data sampled at the electrodes. The sampling rate is 250 samples per second with a 16-bits quantization. The proprietary protocol that encapsulates the data produces an overhead in the data transmission of 27%. As a result, the bit rate received in the host computer is 20,3 kbits/s.

The Enobio sensor can be connected to the AsTeRICS platform in three different ways:

- It can be connected through a built-in IEEE 802.15.4 transceiver in the AsTeRICS platform,
- it can be connected directly through a wire from the Enobio to a serial UART port of the AsTeRICS platform and,
- it can be connected through a provided receiver which has an USB interface.

The first two options imply major modifications to the current Enobio system way of work. In the case the AsTeRICS platform provided a built-in IEEE 802.15.4 transceiver, some major changes would be necessary in the software part of the system. They would consist in substitute the current USB driver control for a new one for the built-in transceiver. On the other hand, this choice implies substitute the provided Enobio acquisition device by the built-in transceiver, thus the AsTeRICS platform would be more compact.

The wired option implies the same software change since a new driver should be controlled, in this case the serial port device. This option also means not to use the provided IEEE 802.15.4 transceiver since the data would be passed through a wire directly from the Enobio sensor, but the system would be less wearable since a new cable would be present from the user head to the AsTeRICS platform.

Finally, the choice of communication between the Enobio sensor and AsTeRICS platform by provided USB receiver would not imply any change in the driver control as stated above, so no extra effort should be necessary in order to integrate the USB driver necessary to connect with the provided receiver.

The FTDI chip driver, necessary for accessing to the Enobio's provided receiver, shall be installed in the AsTeRICS platform. Up to now there are versions of this driver for all the potential Operating Systems that an embedded platform can run, i.e., Windows 7/XP/CE, Windows Mobile, Linux and Pocket PC.

All these Operating Systems are able to handle an USB device since they implement an USB host controller. Furthermore they are also able to perform floating point arithmetic operations at least by software. Therefore, they fit the requirements related with the embedded platform Enobio will connect to.

In order to control the sensor, a Java application that interfaces with the FTDI driver, is already implemented. This application implements the proprietary protocol that Enobio uses and performs the automatic control of the offset signal of the electrodes. In addition, this software is ready for sending the commands that allow the Enboio sensor to start and stop sampling.

Since the AsTeRICS Runtime Environment (ARE) proposed in section 2 works over a Java Virtual Machine and the OSGi framework, this software will be reused and modified properly in order to become the interface of the Enobio sensor with the ARE.

The current user interface of the Java application is not needed, so it shall be removed. The AsTeRICS architecture implements several interfaces, including the IComponentPlugin which allows access to the data and capabilities of the sensors of the system, so in order to interface the ARE, this interface is going to be added to the current Java application.

From the previous analysis it is clear that the option of using the provided interface allows faster integration, since it is not necessary to develop a new piece of software to handle the hardware interface. This strategy is going to be followed in Prototype 1 of the AsTeRICS system in order to integrate the Enobio device with the embedded platform.

4 Software Specification and Architecture

The AsTeRICS software framework provides an open and flexible toolkit for enabling the formation of various AT systems. Its architecture is separated in three parts:

- The AsTeRICS Runtime Environment (ARE)
- The AsTeRICS Configuration Suite (ACS)
- The AsTeRICS Application Programming Interface (ASAPI)

The ARE is used to execute a predefined system design in the target environment, which usually is the Embedded Computing Platform. As such, it provides a middleware architecture that controls and manages the components used to form the deployed applications. The ACS is mainly used to graphically design the layout of the system as a network of interconnected components. Finally, the ASAPI is used to connect the ACS - or other client applications running on the PC (like AT-software by consortium member SENSORY) - to the AsTeRICS Runtime Environment.

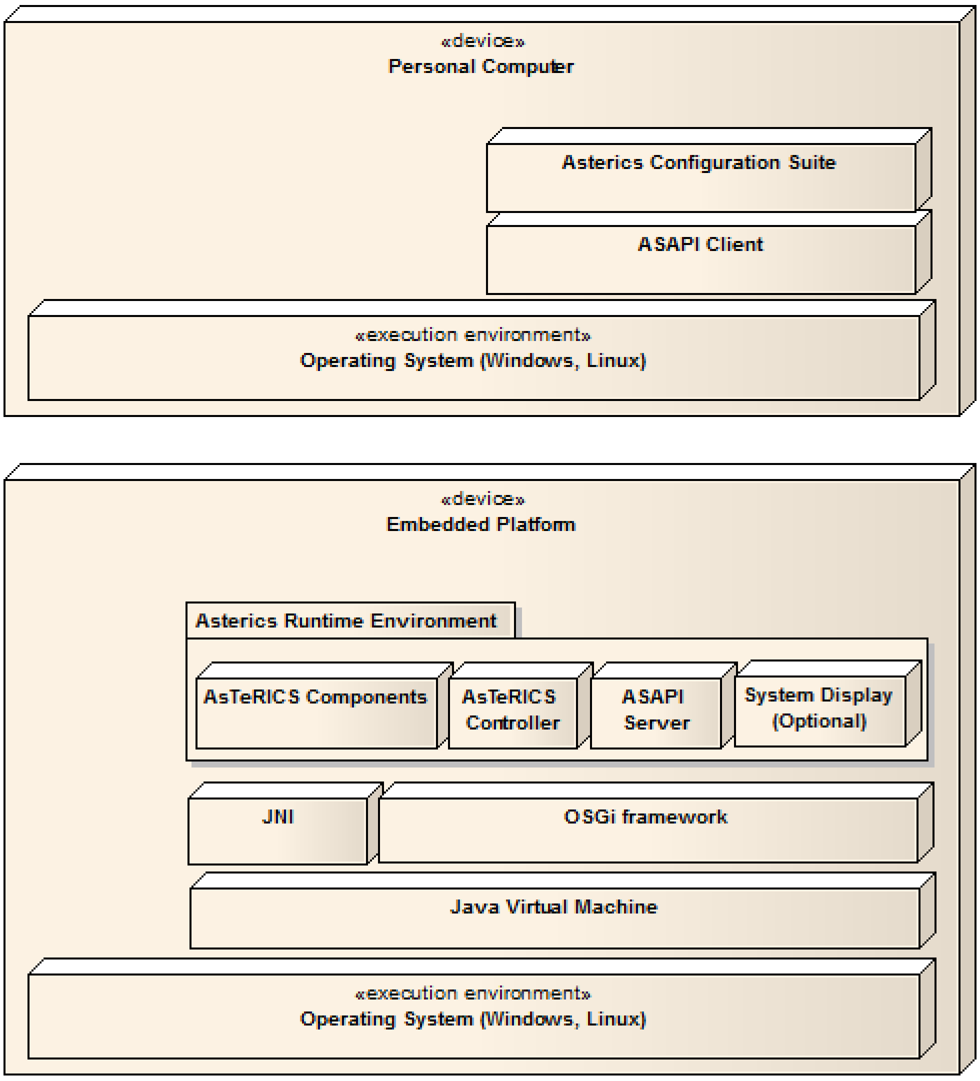


Figure 23: High-level view of the system architecture (deployment model)

Central to this arrangement is the system model, which in case of the configuration suite comprises a set of abstract entities (in the form of components - i.e., sensors, processors and actuators - and channels), and in case of the embedded platform it comprises of the actual software and hardware components realizing them, and the actual bindings established between them.

An overview of the system architecture is illustrated in Figure 23.

The Configuration Suite and the ASAPI are deployed on a personal computer, running a common operating system (such as Windows or Linux).

The ARE is commonly deployed on an embedded device, running an appropriate operating system (OS), typically an embedded variant of Windows or Linux. On top of the OS, an appropriate Java Virtual Machine (JVM) is used to host the OSGi component framework which provides support for modularity and dynamic loading/unloading of components. All the core components of the framework (described in detail later) are defined as OSGi modules. Certain components that need to access legacy code (e.g., written in C or C++) are also deployed on top of OSGi, and are interfaced to the native code using Java Native Interface (JNI) as needed. In this regard, and with the exception of the pluggable components that use native code interfaces with platform-specific JNI bindings, the embedded platform is expected to be *platform independent*.

4.1 System Model

The system model describes the software artifacts that are used for realizing the intended behavior, along with their input and output ports and the channels connecting them. Since the system model represents the main communication means between the ACS and the ARE, it is expected to be a *serialisable*[1] object, easy to transfer and translate.

An example of such a visual representation of a model, as designed in BrainBay, is illustrated in Figure 24. This model shows a simple SIGNAL generator with an output port connected to a MAGNITUDE processor's input port. The output port of the latter is connected to an OSCILLOSCOPE for visualizing the runtime variation of the generated signal.

[1] In software engineering, *serializable* refers to the property of a programming *object* of being transformed to and from a series of bytes which can be communicated or stored via common byte streaming techniques. The resulting series of bytes practically encodes both the type and the state of the corresponding object.

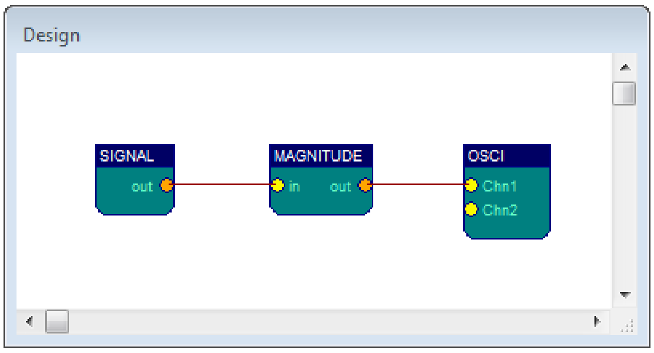


Figure 24: Example of system model as seen in BrainBay

As shown in this figure, the *components* feature *input* and *output ports* which are connected via *channels* (also referred to as *bindings*). Furthermore, each component has a set of *properties* which characterize their behavior (not shown in Figure 24). For instance, the signal frequency and amplitude could be properties of the SIGNAL component.

While in the ACS the system model has a *conceptual* base only, when committed to the ARE it is realized by instantiating, connecting and activating the corresponding software (and their underlying hardware) components.

4.1.1 Pluggable Component Modules (PCOM) and Channels

The PCOMs are the main building blocks for any system design within AsTeRICS. In this case, the components facilitate the following roles:

- Sensors - used to produce data (e.g., a face tracking sensor or the Enobio BCI),
- Processors - used to process data (e.g., perform an FFT),
- Actuators - used to consume data towards a goal (e.g., a mouse controller).

Each component has at least one port. Ports are the outlets used to receive or transmit data. They are classified to input and output ports and are related to certain data types depending on the data that they are expected to receive or transmit. A sensor (i.e. signal source) makes data available, a processor (i.e., signal processor) transforms or combines signal values, and an actuator (i.e., signal sink) consumes data to control the system or initiate data transmission.

The components communicate through channels which are used for representing the data flow between ports. From a designer's perspective, channels can only be formed between an output and an input port, and the channel is assumed to be directional with the data flowing from the source output port to the sink input port. Furthermore, from a modeling perspective, channels can only be formed between ports associated with the same data types.

The graphical system model representation is typically encoded in a well-defined format inside the ACS (e.g., in XML), so that it can be stored and communicated to and from the ARE.

The following core components are implemented to provide the functionalities of the high level software architecture:

- ACS component

- ASAPI subsystem
- ARE Controller component
- System Display component
- Generic plugin components: Pluggable Component Modules

These components are interfaced through appropriate (external) interfaces:

- IModelService: This interface specifies a service that is provided by the ARE and is utilized by the components that require read/write access to the system model.
- IMonitorService: This interface specifies a service that is provided by the ARE and is utilized by the components that require monitoring of the framework operation (typically the ACS and the included display of the ARE, when available).

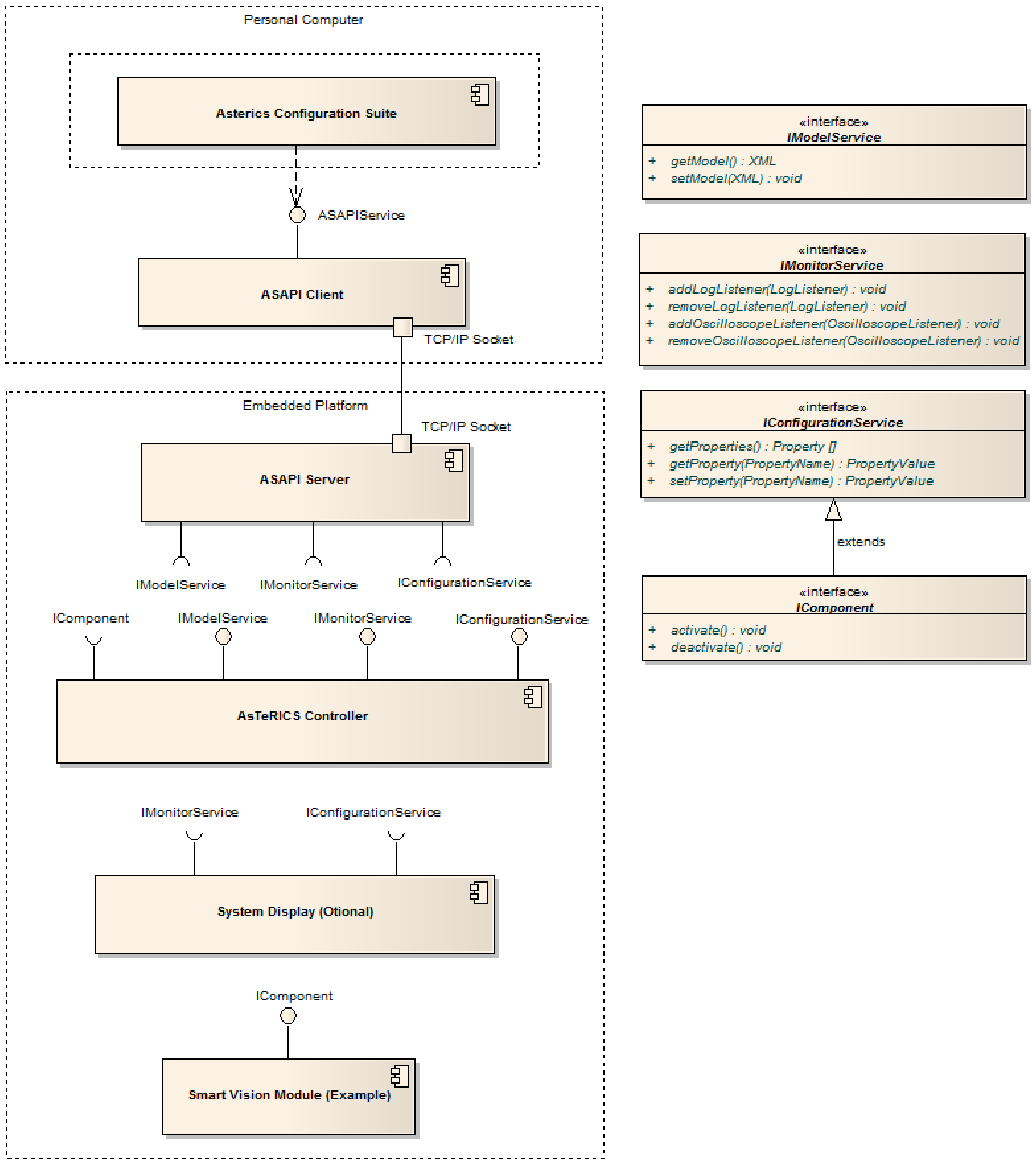

Personal Computer
Asterics Configuration Suite
ASAPIService
ASAPI Client
TCP/IP Socket
Embedded Platform
TCP/IP Socket
ASAPI Server
IModelService
IMonitorService
IConfigurationService
IComponent
IModelService
IMonitorService
IConfigurationService
AsTeRICS Controller
IMonitorService
IConfigurationService
System Display (Otional)
IComponent
Smart Vision Module (Example)
«interface» IModelService
+ getModel() : XML
+ setModel(XML) : void
«interface» IMonitorService
+ addLogListener(LogListener) : void
+ removeLogListener(LogListener) : void
+ addOscilloscopeListener(OscilloscopeListener) : void
+ removeOscilloscopeListener(OscilloscopeListener) : void
«interface» IConfigurationService
+ getProperties() : Property []
+ getProperty(PropertyName) : PropertyValue
+ setProperty(PropertyName) : PropertyValue
extends
«interface» IComponent
+ activate() : void
+ deactivate() : void

Figure 25: Main components of the AsTeRICS software architecture

- IConfigurationService: This interface specifies a service that is also provided by the ARE and is utilized by the components that require access to reading and setting configuration properties of the system model (typically the ACS, and the included display of the ARE when available).
- IComponent: This interface specifies the service that is provided by any component plugin instance, allowing for control of its lifecycle. The component plugins are containers of component instances (such as a web-camera sensor, the Enobio BCI sensor, an FFT processor, a mouse actuator, etc).

This organization is illustrated in Figure 25.

4.1.1.1 Signal processing PCOM for simple HMI

The scope of this section is to describe the pluggable component module that will be necessary to implement the simple Enobio signal processing functionalities to be executed in the ARE. The output of these modules will allow performing a simple HMI in Prototype 1.

The objective of this analysis is to present a design architecture that meets the requirements exposed in section 2.2.1. The functionalities presented there shall be executed and integrated in the ARE defined unterhalb. It implies they have to be defined as an AsTeRICS pluggable component module (PCOM).

The main property of this kind of modules is that they run within an OSGi framework which is hosted by a Java Virtual Machine (JVM). So, the interfaces defined by the OSGi framework and the ARE shall be present in the design of the Starlab simple signal processing modules to be defined in the current section.

A design solution for implementing the functionalities described in the requirement file is to define them as a piece of software based in the framework and interfaces mentioned above. An alternative solution, in case the required functionalities demand high computational performance, which could not be achieved by the AsTeRICS platform, is to implement them in a dedicated external hardware signal processor controlled by a simpler piece of software integrated in the ARE with its corresponding interfaces.

The former solution implies a pure software-based solution, whose success only depends on the computational capabilities of the AsTeRICS platform. There are also several design solution alternatives in order to develop the algorithms of the signal processing functionalities. Taking advantage of the fact that the PCOMs run in a Java-based framework, that means they have to be developed in Java, the algorithms could be also developed in this language. Another approach is to write the algorithms that implement the signal processing functionalities in native C/C++ language and access them through the Java native Interface (JNI). It will allow, in general, a faster execution of the algorithms, furthermore the implementation could be moved faster to other lighter platforms without JVM such as the ones based in micro-controllers, that could be of interest for future Starlab's developments.

The latter solution, based on a hybrid hardware-software solution, is suitable when the algorithms to be developed need high computational demand that is not affordable by the current AsTeRICS platform. In that case, a dedicated piece of hardware would carry out the algorithms and a piece of software running in the ARE would take care of the results output by the dedicated piece of hardware. This solution has a higher level of complexity but can solve almost any bottleneck that may come up in a specific algorithm.

Following algorithms, which are taken into consideration for development, fulfil the functional requirements stated in the requirements section:

- SPROC1-4: FIR filters,
- SPROC6: FFT,
- SPROC7: triggering plus sampling counter,
- SPROC8: sample averaging,

- SPROC9a: matrix product.
- SPROC10: threshold comparator,
- SPROC11: numerical differentiation,
- SPROC12: FIR filter plus down-sampling,
- SPROC13: Euclidean distance,
- SPROC20: cross correlation,
- SBPROC3: Simple blink detection,
- SBPROC4: Double blink detection.

For all the algorithms, it is needed to accomplish with the real time processing requirement, so every sample shall be processed in less than 4 ms (250 samples per second). Thus, if the algorithms ran in an architecture with a performance of 3300 MIPS, for instance, a netbook with an Intel N270 micro-processor the system would have 0,132 MIPS for processing a simple sample. For this calculation it has been taken into account that the algorithms share the micro-processor with 100 more processes (other algorithms, operative system, etc.).

From the algorithm listed above, the ones that are more time consuming are the FFT and the cross correlation. Their final amount of instructions will depend on the number of samples in the input vector of the FFT (the frequency resolution) and how long are the two vectors to cross correlate.

From the previous analysis it makes sense to implement the algorithms in a software-based solution due to the fact that they do not present evidence of being unable to be executed in a regular micro-processor with a regular operating system, which is the case of AsTeRICS. Only in case that resolution of the FFT or the maximum number of iterations for the fuzzy C-means algorithm needed to be great enough to not be able to process the original sampling rate, the alternative of developing these algorithms with a hardware-based solution would make sense. Anyway, in case it is needed to embed the algorithms there will not be time to develop it for the Prototype 1.

In conclusion, in order to get the best performance in terms of time execution and minimize the possibility that a bottleneck comes up, the algorithms will be developed using native C/C++ and wrapping them with the JNI.

4.2 AsTeRICS Runtime Environment (ARE)

The main features of the AsTeRICS Runtime Environment are constituted as follows:

- Includes a mechanism that allows components to be deployed, resolved and activated dynamically, as defined by the system model;
- Provides methods for accessing and setting its running system model;
- Provides methods for accessing and setting certain properties of the running system;
- Provides methods for monitoring the computing load.

4.2.1 Runtime Model Concepts

The ARE hosts and controls the components that realize the Assistive Technology (AT) applications. As such, it features a component-based approach, where various specialized plug-ins (i.e., sensors, processors and actuators) are interfaced together to realize the desired behavior. The main runtime model concepts in the ARE are the *components*, the *ports*, and the *channels* (also known as *bindings*). These concepts are available for *introspection* and *reflection* in runtime (i.e., their properties can be both queried and edited). In more detail, these artifacts are described in the following:

- *Pluggable Component Modules* (or simply *Components*): The components correspond to any of the three main concepts used in the AsTeRICS system, namely the *sensors*, *processors*, and *actuators*. Each component has one or more ports, which are used to connect with other components via *channels*.
- *Ports*: The ports are the main concepts used to indicate an input or output communication outlet of a component. Output ports are used to communicate data *out* of a component, while *input* ports are used to allow data be communicated into a component. When two components are connected together, the output port of one is connected to the input port of the other, in essence realizing a *channel* (or *binding*).
- *Channels* (or *bindings*): A channel is the main concept describing a communication link. Each binding is *unidirectional* (i.e., the data communication flow is one way only). Each channel is explicitly specified via two components and two ports. The source port, and its corresponding source component, is connected to the target port and to the corresponding target component. By definition, the former must be an *output* port and the latter an *input* port. The data flow is naturally from the source (output port) to the target (input port). In order for a channel to be formed, the corresponding data types of both the source and the target ports must match (i.e., they must both produce and consume the same data type). Special ports are also defined for forming event-notification channels, i.e., channels that communicate an arbitrary number of event types between certain components.

It should be noted that these concepts describe merely *types* of runtime artifacts. For instance, *component* specifies a special component type that can be instantiated multiple times. In each instantiation, all attributes are static, except the *properties* that can be edited in runtime. For example, a specialized signal processing *processor* can be instantiated multiple times, with different property values, and can be connected to different components. While both component instances share the same type, they are individually used and maintained in the ARE.

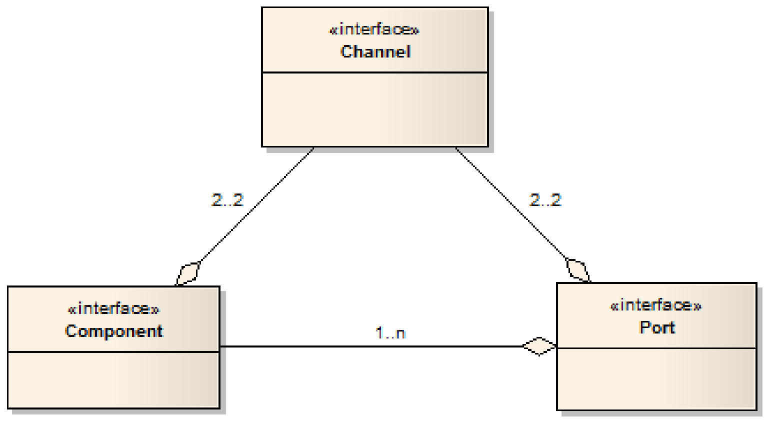


Figure 26: Simple view of the runtime model

These artifacts and their relationships are illustrated in Figure 26. This figure illustrates the relationships between components, ports and bindings. A component consists of one or more ports. A binding, on the other hand connects exactly two components, via two corresponding ports. A more detailed description of the main runtime concepts and their relationships is provided in the following paragraphs.

4.2.1.1 Components

The *components* are the main artifacts in the ARE runtime model. As mentioned before, components can serve one of three main roles:

- *Sensors*: these are components which only feature *output* ports (i.e., they do not depend on input from any other components). Typical sensors are commonly coupled to underlying hardware sensors to generate their output data (e.g., a face tracking sensor which is coupled to a web-camera), but they can also be completely realized internally (e.g., a signal generator).
- *Processors*: these are components which feature both *input* and *output* ports. This is the most common type of components, and provides the foundation for forming applications. The processor components can be either realized completely internally (e.g., an *average* which keeps track of the last *n* values of a scalar value and always outputs their average value) or they can be coupled to some external software library or even coupled to a hardware component (e.g., utilize legacy libraries for complex signal processing, or even utilize specialized hardware accelerators for highly demanding computations).
- *Actuators*: these are components which only feature *input* ports (i.e., they do not produce any output that can be utilized by other components). The main role of actuators is to enable the desired functionality of the applications, and for testing (e.g., a mobile phone actuator allows to place or answer phone calls and to send SMS[2] messages, while an oscilloscope actuator allows for viewing, and thus testing or debugging, of signal generators).

Each component has *at least one port* (otherwise it would be impossible to be interfaced with other components). Furthermore, components can specify both *input* and *output* ports. While

[2] Short Text Message

components typically offer a *fixed* set of ports, sometime it is required that specific ports are defined with a dynamic *multiplicity*. The following two options are provided, allowing for a highly versatile component specification:

- *1..1 (default)*: the one-to-one multiplicity indicates that the port is *mandatory* (i.e., at least one realization is needed) and *up-to-one* realization (i.e., no more than one port can be defined). In simple words, this multiplicity indicates that *exactly one realization* of this port is needed. This is the *default* option.
- *1..n*: the one-to-many multiplicity indicates that the port is *mandatory* (i.e., at least one realization is needed) and *up-to-many* realizations (i.e., more than one port can be defined). This option is useful for certain components that can be interchangeably used with varying numbers of inputs (e.g., a MULTIPLEXER component that can interchangeably multiplex a varying number of inputs).

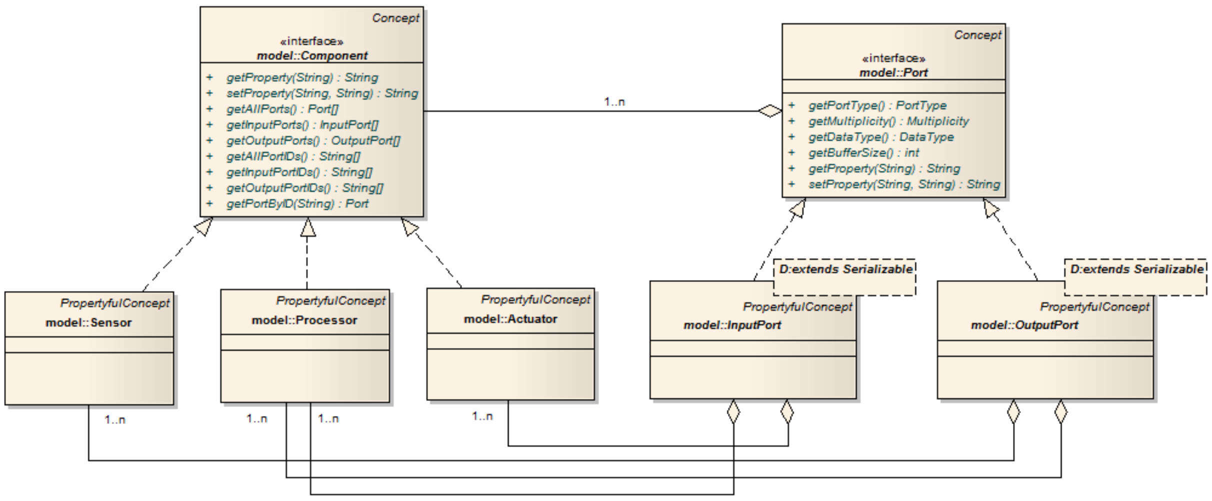

Concept
«interface»
model::Component
+ getProperty(String) : String
+ setProperty(String, String) : String
+ getAllPorts() : Port[]
+ getInputPorts() : InputPort[]
+ getOutputPorts() : OutputPort[]
+ getAllPortIDs() : String[]
+ getInputPortIDs() : String[]
+ getOutputPortIDs() : String[]
+ getPortByID(String) : Port
1..n
Concept
«interface»
model::Port
+ getPortType() : PortType
+ getMultiplicity() : Multiplicity
+ getDataType() : DataType
+ getBufferSize() : int
+ getProperty(String) : String
+ setProperty(String, String) : String
PropertyfulConcept
model::Sensor
PropertyfulConcept
model::Processor
PropertyfulConcept
model::Actuator
D:extends Serializable
PropertyfulConcept
model::InputPort
D:extends Serializable
PropertyfulConcept
model::OutputPort
1..n
1..n
1..n
1..n

Figure 27: Complex view of runtime model for the component concept

These concepts are illustrated in Figure 27. A component can be any of three main realizations: *sensor*, *processor*, and *actuator*. On the other hand, a port can be instantiated either as an *input* or an *output* port. Sensors have one-or-more output ports only, actuators have one-or-more input ports only, and processors have both one-or-more input ports and one-or-more output ports.

4.2.1.2 Ports

The ports are the main concepts allowing interfacing between components. Ports are classified as *input* or *output* ports, depending on their role. Each port features a buffer where data is accumulated before it is communicated outwards (output ports) or before it is internally consumed (input ports).

Furthermore, each port is associated with a specific data type, indicating the type of the data communicating through the port. Examples of such data types, carrying the representation and semantics as inherited from the Java language [35], are:

- Byte: a single byte
- Char: a single character (used to form strings)
- Integer: a 32-bit integer

- Long: a 64-bit long integer
- Float: a 32-bit single precision scalar
- Double: a 64-bit double precision scalar

The main properties and relationships of the *port* concept are illustrated in Figure 28.

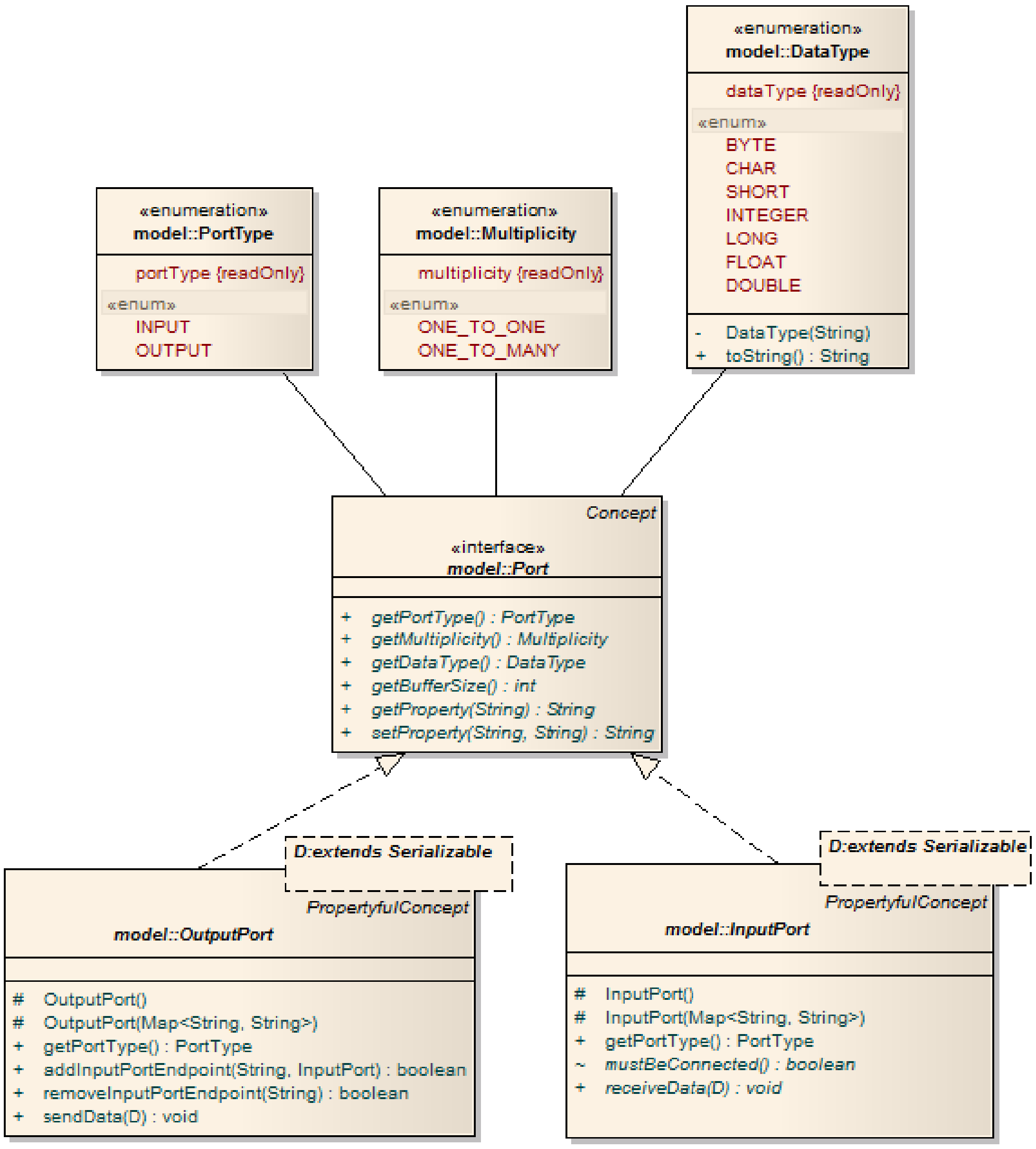


Figure 28: Runtime model for the port concept

The port concept features methods for accessing the *port type*, its *multiplicity*, its *data type*, and also for getting and setting property values. The main subtypes of *port* are the *OutputPort* and the *InputPort*.

It should be pointed out that the *input* port is different from the *output* port by featuring an additional method for checking whether a binding to the port is mandatory or not. This is needed to check whether a component is resolved or not (i.e., by checking whether all its input ports marked as "*mustBeConnected*" are indeed connected). This is important because it ensures that all the defined components are functional, i.e., appropriately connected, before they are activated.

Finally, it should be noted that special port types will also be defined for event communication. Unlike common ports which communicate a fixed data type, event ports will be able to communicate different events, encoded in a uniform way. Input event port types will be defined with the “mustBeConnected” property set to false by definition. Also, output event ports will allow the formation of multiple channels using the same output port as a common endpoint.

4.2.1.3 Channels

The channel is the main concept used for interfacing components through ports. As such, the channels are defined via a source port in a source component and a target port in a target component. When formed, certain checks are performed to ensure that the data types of the source and the target ports are compatible.

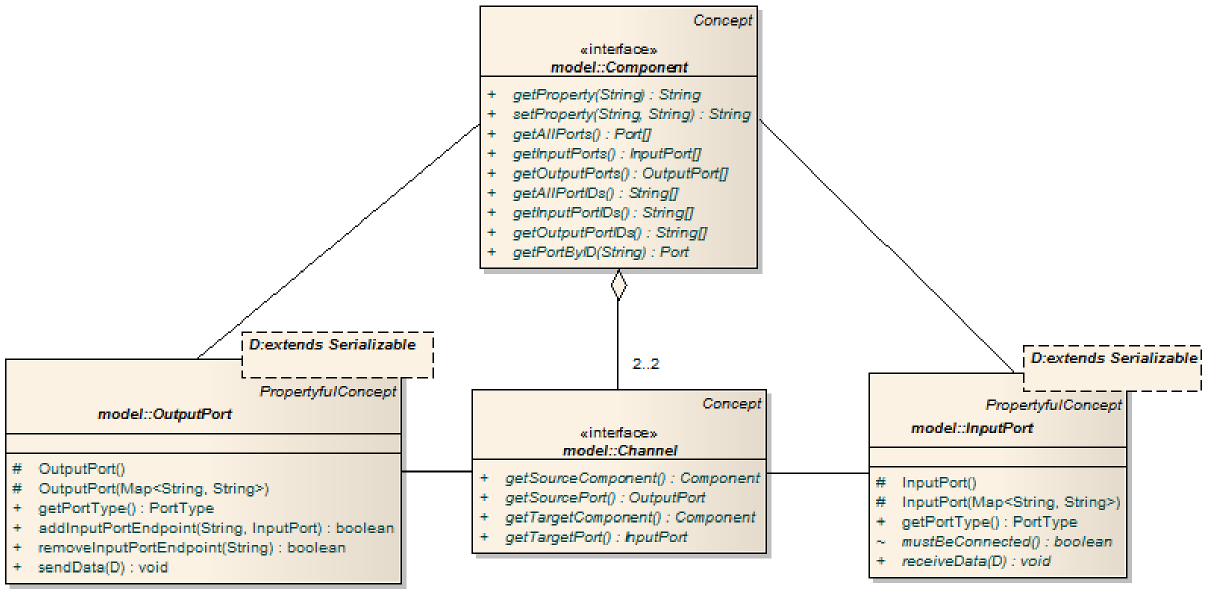

Concept
«interface»
model::Component
+ getProperty(String) : String
+ setProperty(String, String) : String
+ getAllPorts() : Port[]
+ getInputPorts() : InputPort[]
+ getOutputPorts() : OutputPort[]
+ getAllPortIDs() : String[]
+ getInputPortIDs() : String[]
+ getOutputPortIDs() : String[]
+ getPortByID(String) : Port
2..2
D:extends Serializable
PropertyfulConcept
model::OutputPort
# OutputPort()
# OutputPort(Map<String, String>)
+ getPortType() : PortType
+ addInputPortEndpoint(String, InputPort) : boolean
+ removeInputPortEndpoint(String) : boolean
+ sendData(D) : void
Concept
«interface»
model::Channel
+ getSourceComponent() : Component
+ getSourcePort() : OutputPort
+ getTargetComponent() : Component
+ getTargetPort() : InputPort
D:extends Serializable
PropertyfulConcept
model::InputPort
# InputPort()
# InputPort(Map<String, String>)
+ getPortType() : PortType
~ mustBeConnected() : boolean
+ receiveData(D) : void

Figure 29: Runtime model for the binding concept

A typical binding is illustrated in Figure 29. The binding is associated to two components, and an input and output port, one from each of them.

Typically, a source, i.e., output port might be associated to multiple targets, i.e., input ports. Nevertheless, in this runtime model it is assumed that each binding consists of exactly one source and one target port. One-to-many bindings are also implicitly supported via multiple instances of one-to-one bindings.

Special channels can also be formed between event ports. In this case, both input and output ports can be used to connect multiple channels. For instance, the same output event port can be connected to multiple input event ports, and at the same time, the same input event port can receive input, .i.e., events from multiple output event ports.

4.2.1.4 Component Architecture of ARE

This subsection describes the internal architecture of the ARE component. Naturally, the main scope of this component is to maintain and realize the deployed model. As such, it features the following sub-components:

- *Controller*: This component is responsible for coordinating the actions inside the ARE. To achieve this, it uses the other sub-components described below.
- *ModelManager*: The model manager is used to maintain and manage the model (cf. section 4.2.1). As such, it provides methods for transforming the model from and to standard representations (such as XML), for validating its consistency, and for editing the properties of the modeled concepts (i.e., of the components, channels and ports). A special feature of the model manager is that it includes an input event port that allows it to be controlled by the Assistive Technology application for switching between various individual models.
- *Configurator*: The configurator is the component which translates the model into actual components and channels. It is thus responsible for realizing the encoded models and also for coordinating the activation (i.e., *start*) and deactivation (i.e., *pause* and *stop*) of the corresponding components. Before realizing certain models, the configurator utilizes the validation services of the model manager. Also, in order to access existing ones, or create new instances of components, the configurator uses the services available by the component repository. Finally, it also provides support for forming new channels (or dissolving existing ones) between certain ports.

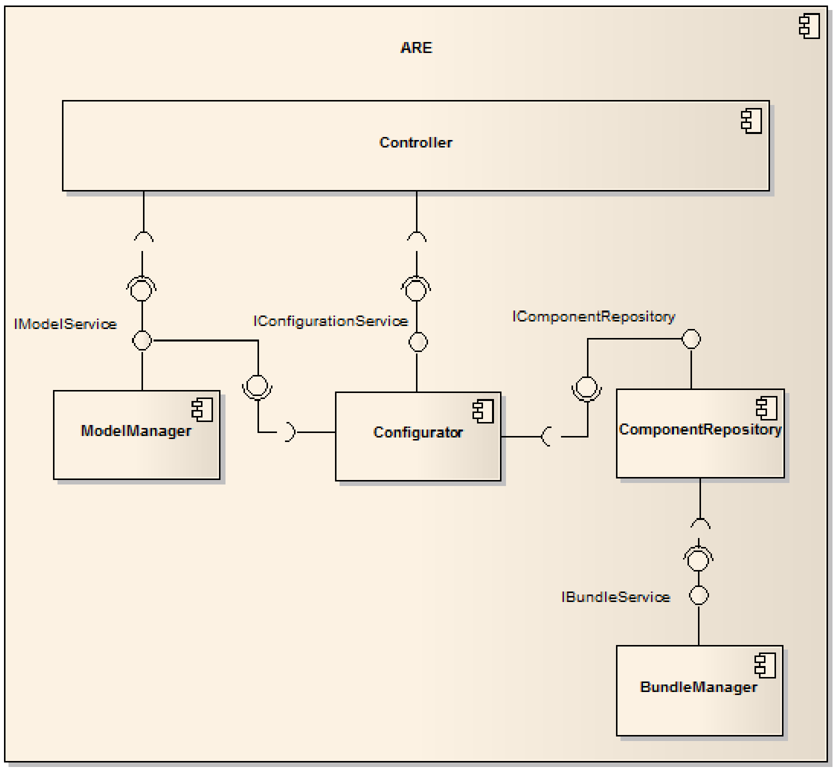


Figure 30: Internal architecture of the ARE

- *ComponentRepository*: The component repository serves two roles. First, it maintains a list with the available component types, which can be changed when new components are installed or existing ones uninstalled. Second, it maintains a repository with the current component instances. New instances can be dynamically created, and existing ones be dissolved.
- *BundleManager*: This component allows for dynamically installing (or uninstalling) software bundles containing one or more components. This is needed to allow for easy updating of ARE instances with new (or updated) component implementations. For this purpose, the OSGi bundle mechanisms will be used. In essence, when a new bundle is installed (or uninstalled), it will be checked whether it contains AsTeRICS components. If it does, the components will be registered (or unregistered) with the component repository by reading appropriate metadata from the bundles.

The relationships between the sub-components are illustrated in Figure 30. Also, to illustrate the interaction between these components in use, consider the sequence diagram illustrated in Figure 31.

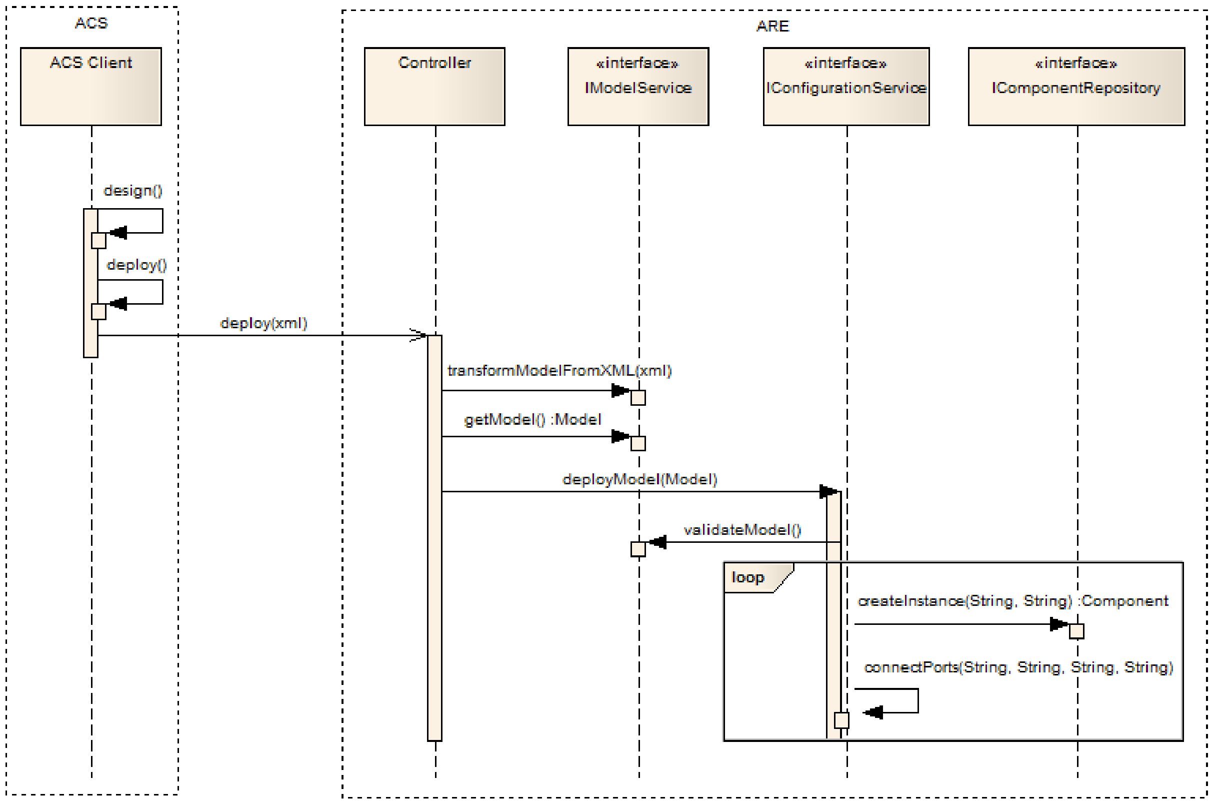

ACS
ACS Client
ARE
Controller
«interface» IModelService
«interface» IConfigurationService
«interface» IComponentRepository
design()
deploy()
deploy(xml)
transformModelFromXML(xml)
getModel() :Model
deployModel(Model)
validateModel()
loop
createInstance(String, String) :Component
connectPorts(String, String, String, String)

Figure 31: Sequence diagram illustrating a typical interaction between ACS and ARE

In this diagram, an ACS client (cf. section 4.3) is used to design an application model (i.e., graphically in an appropriate GUI), which is then deployed in he ARE. For this, the ASAPI protocol is used, which however is not illustrated here to avoid cluttering (for more information on the ASAPI see section 4.4).

On receiving the *deploy* message, the Controller (which is the main component of the ARE) uses the model service to transform the model, which is encoded in XML, into its object representation. The resulting model is then *deployed* using the configuration service. The latter first *validates* the model, using the model service, and then performs a set of commands which aim at realizing the modeled application. These commands include the *instantiation* of component instances, via the component repository, and the physical connection of the corresponding ports.

To better illustrate the use of modeling concepts in the formation of Assistive Technology applications, we present the following example.

4.2.1.5 Assistive Technology Application Example

This example realizes a user interface which is controlled by tracking the face of the user. In essence, the user moves her or his face so that the mouse cursor is controlled by tracking and interpreting her or his motions. The two main functions realized in this example are:

- First, the mouse cursor coordinates are updated by tracking the user’s nose;
- Second, mouse left-clicks are triggered by tracking the distance of user’s mouth to her or his chin (i.e., when the user opens her or his mouth, a click is triggered).

The application consists of the following components:

- Face Tracking Sensor: It connects to a web camera and performs image analysis with the purpose of detecting the user’s face, and producing coordinates for her or his nose and chin. The output of this sensor is provided via four output ports, corresponding to the Cartesian coordinates of the nose and chin.
- Integer Averager Processor: Two instances of this component are used to compute the average value of a predefined set of integers. In essence, this component features a fixed length FIFO[3] list. Input values are pushed to the head, causing older values to be dropped from the tail. Whenever a new value is received, the average of the accumulated values is computed and communicated to the output. From a practical perspective, the two component instances are used to smooth out variations that may occur in the coordinates produced by the face tracking sensor.
- Derivation Processor: This component applies certain, simple computations to the input values, as desired. For instance, in this scenario the component instance is used to compute the difference between the Y value of the node and the Y value of the chin, in essence computing the distance between the user’s nose and chin (it is assumed that her or his face is in horizontal position).
- Threshold Processor: This component is used in combination with components such as the derivation processor to detect when certain conditions apply (in this case the distance between the nose and chin exceeds a predefined threshold). In practice, this component is used to identify when the user opens her or his mouth.
- Mouse Actuator: Finally, the mouse actuator component is used to translate the generated coordinates and the “mouth opening events” to actual mouse actions in the

[3] First In First Out

system. This component features input ports that are used for specifying the mouse coordinates and for passing values indicating left-click events.

It should be noted that these components provide a good example of how different strategies can be used to “drive” the processing in components (i.e., event driven by an input or thread driven). For instance, the Face Tracking Processor is driven by an underlying thread which periodically scans and processes the image produced by the web-cam with the aim of computing the coordinates of the nose and chin of the user. On the other hand, the Integer Averager Processor is event driven, as it is triggered whenever a new input is pushed to its port, causing a resulting output value. For more information, see the implementation issues discussed in section 4.2.1.6.

In order for the application to be realized, obviously, appropriate channels must be formed to transfer data between the components, essentially realizing the system. These channels are illustrated in Figure 32.

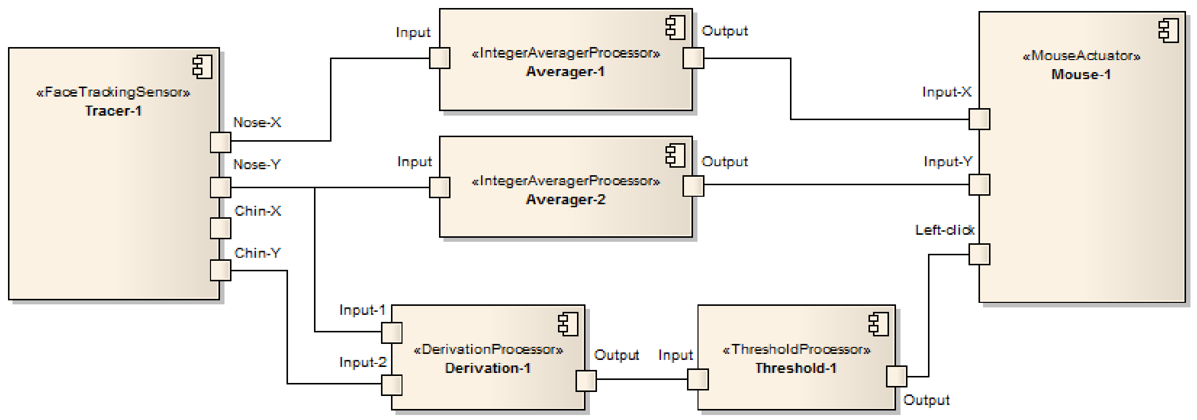

«FaceTrackingSensor» Tracer-1
Nose-X
Nose-Y
Chin-X
Chin-Y
Input
«IntegerAveragerProcessor» Averager-1
Output
Input
«IntegerAveragerProcessor» Averager-2
Output
«MouseActuator» Mouse-1
Input-X
Input-Y
Left-click
Input-1
Input-2
«DerivationProcessor» Derivation-1
Output
Input
«ThresholdProcessor» Threshold-1
Output

Figure 32: Example application with face tracking-driven mouse

On one hand, the nose coordinates produced by the Face Tracking Sensor are first smoothed out by Integer Averager Processors, and then passed on to the Mouse Actuator to drive the coordinates of the mouse cursor. On the other hand, the Y-coordinates of the nose and the chin are first processed by the Derivation Processor which computes their difference. The output of this component is then passed on to the Threshold Processor which computes when the difference exceeds a certain value, in which case it produces an output which is used to drive the left-click port of the Mouse Actuator. It should be noted that in this case, the “Chin-X” output of the Face Tracking Sensor is not connected to any other component, which is not a problem (its values are still produced but ignored). Furthermore, it should be pointed out that the Threshold Processor’s output port is an event output port, and the left-click port of the Mouse actuator is an event input port. The two ports can be easily updated to feature more event types, such as “right click”, “double click”, etc, assuming that the Threshold Processor includes the functionality to produce them, and the Mouse Actuator includes the functionality to consume them.

This example illustrates the use of basic components, providing limited but well-defined functionality, for the formation of functional Assistive Technology applications. The role of the AsTeRICS in this architecture could be described as follows:

- The user designs the system using a GUI in the ACS;

- The modeled application is encoded in a well-known form (e.g., in a predefined, XML-based format) and then communicated to the ARE;
- The ARE parses the XML-encoded model and produces an object version of it;
- For each of the modeled components, the ARE creates a new instance;
- Also, for each of the modeled channels, the ARE creates a new binding that connects the corresponding ports;
- Finally, for each of these concepts (i.e., the components, their ports, and the channels), the ARE set the appropriate properties, as defined in the model.

At that point, the application is ready for activation.

4.2.1.6 Implementation Issues

The following implementation specific issues have been considered:

- *Concurrency and thread pools*: Since the configurations typically feature multiple components that are *concurrently active*, special care is needed with both *driving* the components, and *synchronizing* their processing. The preferred approach dictates: First, *data buffers* inside each input or output port, allowing for better coordination of the data transferring. Second, *thread-pool driven* processing, where a specified set of threads will be commonly used to drive the processing in all of the components. As threads are commonly considered a valuable resource in most JVM implementations, thread pools allow for minimizing their overhead via reuse and dynamic adjustment of the pool size. Since J2SE version 5.0, further support is built in the Java framework for concurrent processing.
- *Active and passive processing of data*: Typical plug-in design dictates two options for processing: First, a plug-in is actively processing and generating data, i.e., by processing in a periodic loop. This can be achieved for example by using a thread (e.g., by registering with the thread-pool discussed above). Second, certain plug-ins can be passively reacting to input (i.e., event-driven), when their processing depends explicitly on received input. For instance, an averager plug-in will *process* its data only when a new input value is registered. Both these options will be supported in the ARE.
- *Sampling rate mismatch*: Another technical issue concerns the interfacing of components with different sampling rates. For instance, a signal generating component could produce values at a higher rate than the desired processing rate of the target signal consumer. In that case, the preferred solution would be the use of appropriate processing components (mediators) for downscaling or upscaling the signal values frequency as needed.

4.2.2 Software Layers in the ARE, Integration of Hardware

Because of the interaction of very different hardware and software components, several layers will be used to separate the different components. The highest level are the OSGi (signal) processing plugins, followed by OSGi connectors to the hardware drivers (some of them will use the Java Native Interface (JNI) to access the drivers). The hardware drivers will be the next layer, followed by the hardware interfaces (WIFI, USB, ZigBee, etc.). While the OSGi layers will be part of the ARE, the hardware components will belong to the AsTeRICS embedded platform (EP). Figure 33 shows the complete data flow, starting from an input hardware device, followed by the interface and the device drivers, next to the OSGi plugins and once again to the hardware layers at EP to the output device.

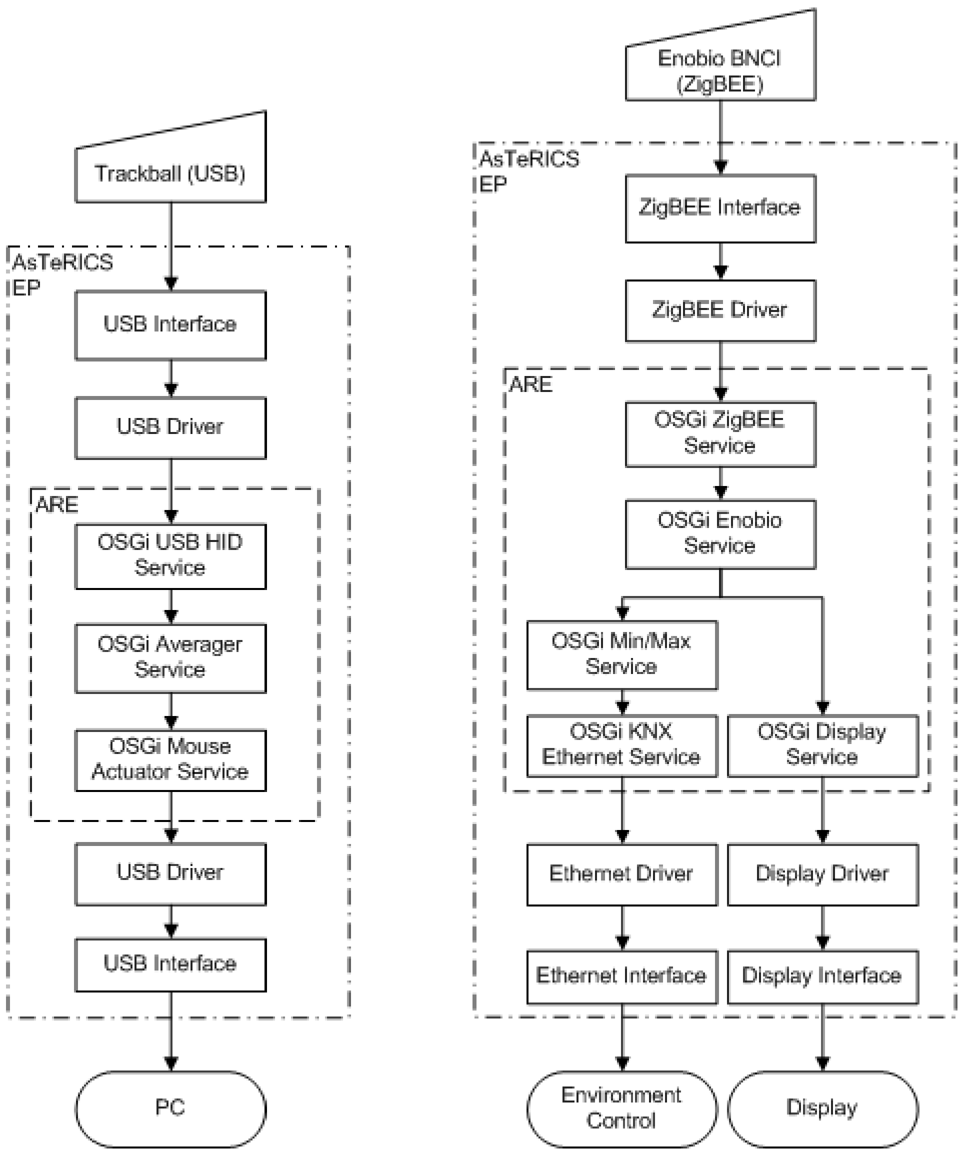


Figure 33: Hardware interfacing and data flow

4.3 AsTeRICS Configuration Suite (ACS)

The main features of the ACS are as follows:

- Provides graphical methods for defining and editing system models
- Provides methods to acquire and deploy system models from/to the ARE (via ASAPI)
- Provides methods to load and store system models on the PC
- Provides local (software-based) oscilloscopes to display actual live data

- Provides a user friendly and accessible way to configure and monitor the ARE

4.3.1 Technologies, Frameworks and Patterns

The basic technology will be the Microsoft .NET framework [7] in combination with the Windows Presentation Foundation (WPF) [8]. These technologies have the advantages of being well supported by Microsoft Windows and also by assistive technologies via the UI Automation [9]. For a clear separation of presentation and the data model in the background, the design pattern of model view controller (MVC) [10] will be used. Another option might be the model view view model (MVVM) pattern [11], which also provides a clear separation of presentation and data. Figure 34 shows the layer architecture of the ACS and the interaction of the ACS with the ARE using the ASAPI.

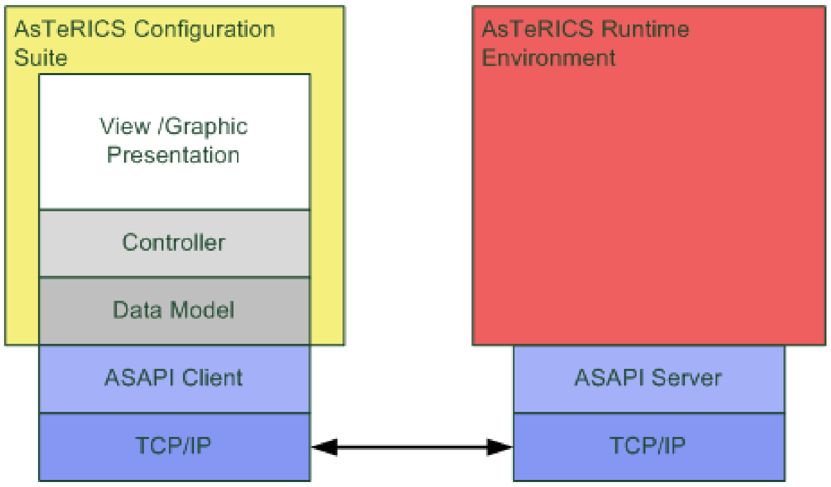

AsTeRICS Configuration Suite
View /Graphic Presentation
Controller
Data Model
ASAPI Client
TCP/IP
AsTeRICS Runtime Environment
ASAPI Server
TCP/IP

Figure 34: Layer architecture and interaction of the ACS with the ARE

4.3.2 Graphical Presentation (View)

Figure 35 shows a possible layout of the graphical user interface of the ACS. The elements for graphical representation will be the standard WPF objects.

During the development process, the language of the UI will be English, for the user tests the language can easily changed by using resource files in the Language of the users (German, Polish and Spanish).

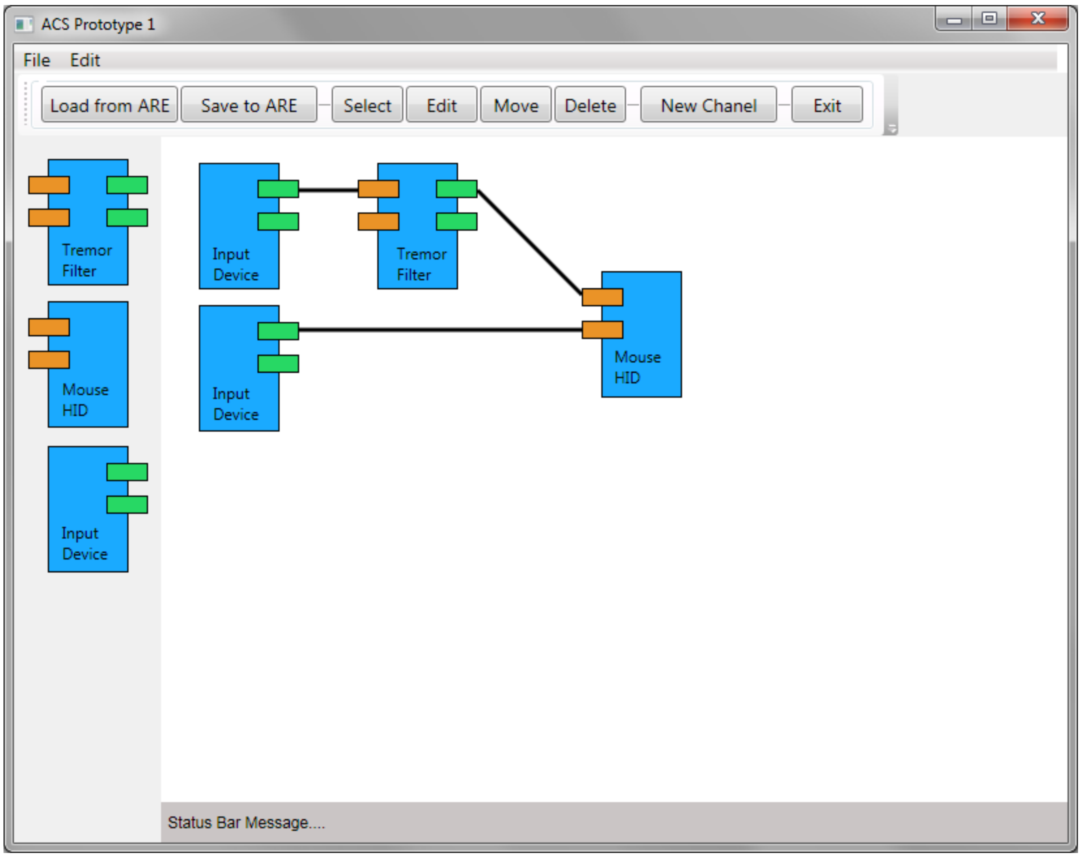


Figure 35: Possible layout of the graphical UI

4.3.3 Data Model

The data model will be loaded from the ARE and represented in a collection containing two classes - plugins and channels. Each class will contain all properties of an object, with the possibility to alter the editable ones. Furthermore, the model will allow storing a graphical representation of a plugin, overloading the standard graphic. After the editing process, the information, stored in the ACS model will be transformed to the ARE data model and transmitted to the ARE.

4.3.4 Functionalities

The following functionalities will be provided by the ACS:

Functionality	**Prototype**
Read configuration from the ARE	PT 1
Write configuration to the ARE	PT 1
Show all available plugins for the connected ARE	PT 1
Add plugins to the model (out of the available ones)	PT 1
Remove plugins from the model	PT 1
Add channels between pluging	PT 1
Remove channels between plugins	PT 1
Change properties of plugins	PT 1
All functionalities accessible with keyboard	PT 1
Visualisation and graphical elements (e.g. Osciloscope plugin)	PT 1
Improvement of the UI following user comments	PT 2
All functionalities are accessible fulfilling the ISO Standard 9241-171	PT 2
Group elements in the graphical layout	PT 2

Table 21: Functionalities provided by the ACS

4.4 AsTeRICS Application Programming Interface (ASAPI)

The AsTeRICS Application Programming Interface (ASAPI) is an interface used to enable advanced communications between the AsTeRICS Runtime Environment (ARE) and external clients. In principle, ASAPI is a *service* that is provided by the ARE and can be consumed by different clients deployed on the same (as the ARE) or remote devices.

While the ARE is implemented on top of JAVA/OSGi, ASAPI clients are assumed to be implemented on top of a variety of platforms. For this purpose, the actual interfacing between clients and the ARE is done at a low TCP/UDP/IP level. For this purpose, either a custom TCP/UDP/IP protocol will be developed, or an existing solution such as Google Protocol Buffers [12], XML RPC [13], or Apache Thrift [14] could be used.

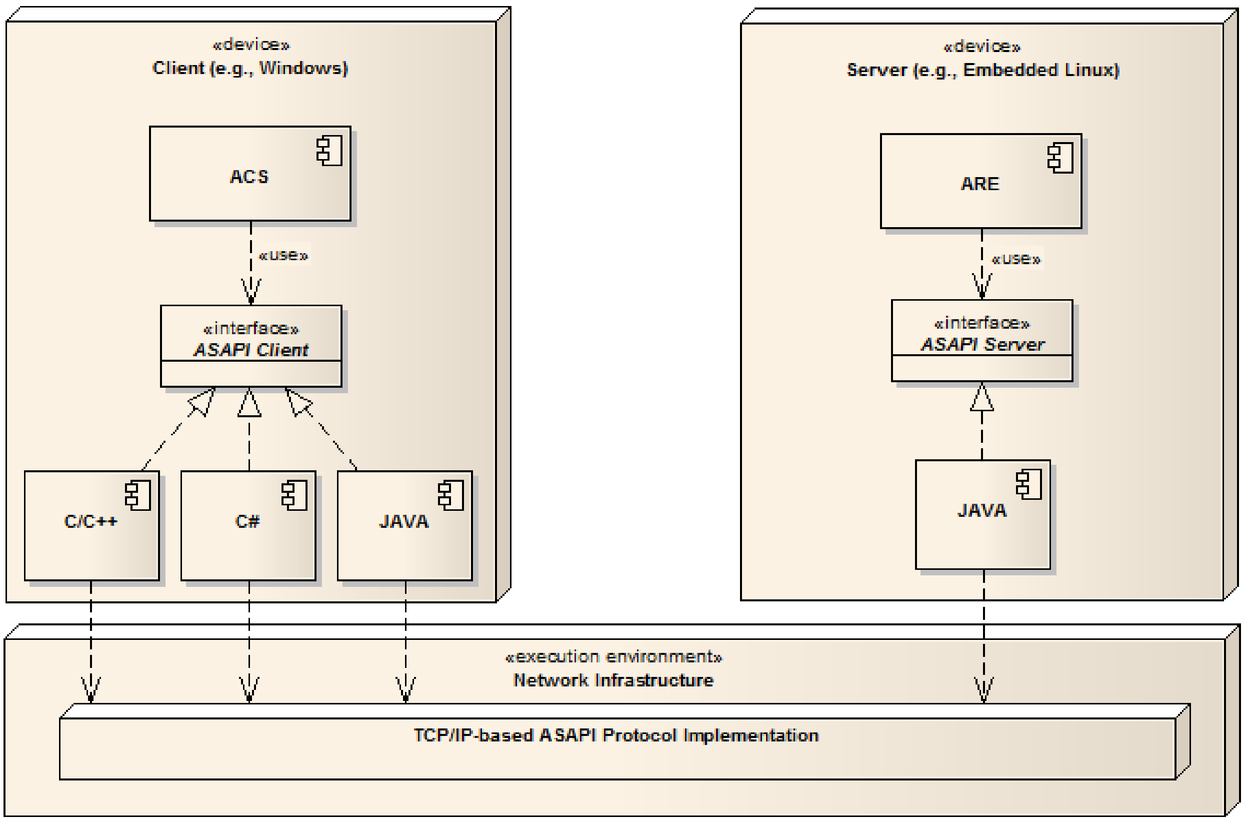


Figure 36: Basic architecture of ASAPI

The basic architecture of ASAPI is illustrated in Figure 36. The “ASAPI Server” is provided by a JAVA based implementation, which utilizes the ARE to provide the specified functionality. On the client side, two interfaces provide the needed functionality: The “ASAPI Client” which extends the “ASAPI Server” with commands for discovering and connecting/disconnecting to the server side, and the “ASAPI Native” which provides specialized functionality for deploying certain components directly in the client, bypassing the ARE. These relationships are illustrated in Figure 37.

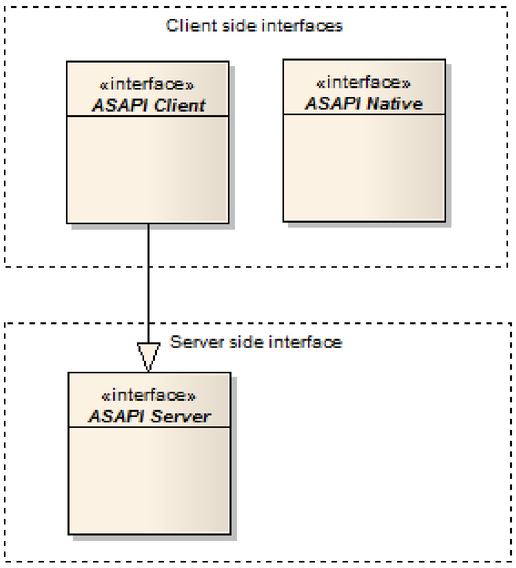


Figure 37: Relationships between the main interfaces of ASAPI

The "ASAPI Server" interface specifies commands that are grouped in the following sections:

- Setup and deploy a model,
- Read and edit the model,
- Read and edit properties (even while running),
- Interface directly to ports in the ARE and
- Status checking.

The commands of the "ASAPI Server" are shown below (in JAVA)

Method	Description
Methods to setup and deploy a model	
`String [] getAvailableComponentTypes();`	Returns an array containing all the available (i.e., installed) component types. These are encoded as strings, representing the absolute class name (in Java) of the corresponding implementation.
`String getModel();`	Returns a string encoding the currently deployed model in XML. If there is no model deployed, then an empty one is returned.
`void deployModel(String modelInXML) throws AsapiException;`	Deploys the model encoded in the specified string into the ARE. An exception is thrown if the specified string is either not well-defined XML, or not well defined ASAPI model encoding, or if a validation error occurred after reading the model.
`void deployFile(String filename) throws AsapiException;`	Deploys the model associated to the specified filename. An exception is thrown if the specified filename is not found.
`void deployModel(String filename, String modelInXML) throws`	Deploys the model encoded in the specified string

`AsapiException;`	into the ARE, which assigns the specified filename to it. An exception is thrown if the specified string is either not well-defined XML, or not well defined ASAPI model encoding, or if a validation error occurred after reading the model. If a model already exists with the specified filename, then it is replaced.
`void newModel();`	Deploys a new empty model into the ARE. In essence, this is equivalent to creating an empty model and deploying it using deployModel(String) above. This results in freeing all resources in the ARE (i.e., if a previous model reserved any).
`void runModel() throws AsapiException;`	It starts or resumes the execution of the model. It throws AsapiException if an error occurs while validating and starting the deployed model.
`public void pauseModel() throws AsapiException;`	Briefly stops the execution of the model. Its main difference from the stopModel() method is that it does not reset the components (e.g., the buffers are not cleared). It throws an AsapiException if the deployed model is not started already, or if the execution cannot be paused.
`public void stopModel() throws AsapiException;`	Stops the execution of the model. Unlike the pauseModel method, this one resets the components, which means that when the model is started again it starts from scratch (i.e., with a new state). It throws AsapiException if the deployed model is not started already, or if the execution cannot be stopped.
Methods to read and edit the model	
`String [] getComponents();`	Returns an array that includes all existing component instances in the model (even multiple instances of the same component type).
`String [] getChannels(String componentID);`	Returns an array containing the IDs of all the channels that include the specified component instance either as a source or target.
`void insertComponent(String componentID, String componentType) throws AsapiException;`	Used to create a new instance of the specified component type, with the assigned ID. Throws an exception if the specified component type is not available, or if the specified ID is already defined.
`void removeComponent(String componentID) throws AsapiException;`	Used to delete the instance of the component that is specified by the given ID. Throws an exception if the specified component ID is not defined.
`public String [] getAllPorts(String componentID) throws AsapiException;`	Returns an array containing the IDs of all the ports (i.e., includes both input and output ones) of the

	specified component instance. An exception is thrown if the specified component instance is not defined.
`public String [] getInputPorts(String componentID) throws AsapiException;`	Returns an array containing the IDs of all the input ports of the specified component instance. An exception is thrown if the specified component instance is not defined.
`String [] getOutputPorts(String componentID) throws AsapiException;`	Returns an array containing the IDs of all the output ports of the specified component instance. An exception is thrown if the specified component instance is not defined.
`void insertChannel(String channelID, String sourceComponentID,String sourcePortID, String targetComponentID, String targetPortID)throws AsapiException;`	Creates a channel between the specified source and target components and ports. Throws an exception if the specified ID is already defined, or the specified component or port IDs is not found, or if the data types of the ports do not match. Also, an exception is thrown if there is already a channel connected to the specified input port (only one channel is allowed per input port except for event ports that can have multiple event sources).
`void removeChannel (String channelID) throws AsapiException;`	Removes an existing channel between the specified source and target components and ports. Throws an exception if the specified channel is not found.
`Methods to read and edit properties (even while running)`	
`String [] getComponentPropertyKeys(String componentID);`	Reads the IDs of all properties set for the specified component.
`String getComponentProperty (String componentID, String key);`	Returns the value of the property with the specified key in the component with the specified ID as a string.
`String setComponentProperty (String componentID, String key, String value);`	Sets the property with the specified key in the component with the specified ID with the given string representation of the value.
`String [] getPortPropertyKeys(String portID);`	Reads the IDs of all properties set for the specified port.
`String getPortProperty(String componentID, String portID, String key);`	Returns the value of the property with the specified key in the component with the specified ID as a string.
`String setPortProperty(String componentID, String portID, String key, String value);`	Sets the property with the specified key in the port with the specified ID with the given string representation of the value.
`String [] getChannelPropertyKeys(String channelID);`	Reads the IDs of all properties set for the specified component. Reads the IDs of all properties set for

	the specified channel.
`String getChannelProperty(String channelID, String key);`	Returns the value of the property with the specified key in the channel with the specified ID as a string.
`String setChannelProperty( String channelID, String key, String value);`	Sets the property with the specified key in the channel with the specified ID with the given string representation of the value.
Methods to interface directly to ports in the ARE	
`String registerRemoteConsumer(String sourceComponentID, String sourceOutputPortID) throws AsapiException;`	Registers a remote consumer to the data produced by the specified source component and the corresponding output port. In the background, the ARE forms a proxy component that is connected to the specified component and port, which is utilized to communicate the data to the corresponding remote consumer. This is similar to the proxy-based approach used in Java RMI.
`unregisterRemoteConsumer(String remoteConsumerID) throws AsapiException;`	Unregisters the remote consumer channel with the specified ID.
`String registerRemoteProducer(String targetComponentID, String targetInputPortID) throws AsapiException;`	Registers a remote producer to provide data to the specified target component and the corresponding input port. In the background, the ARE forms a proxy component that is connected to the specified component and port, which is utilized to receive the data from the corresponding remote producer.
`void unregisterRemoteProducer(String remoteProducerID) throws AsapiException;`	Unregisters the remote producer channel with the specified ID.
`byte [] pollData(String sourceComponentID, String sourceOutputPortID) throws AsapiException;`	This method is used to poll (i.e., retrieve) data from the specified source component and its corresponding output port. Just one tuple of data is returned. The actual amount of data (i.e., in bytes) depends on the type of the port (it is the responsibility of the developer to appropriately deal with the byte array size).
`void sendData(String targetComponentID, String targetInputPortID, byte [] data) throws AsapiException;`	This method is used to pull (i.e., send) data to the specified target component and its corresponding input port. Just one tuple of data is communicated. The actual amount of data (i.e., in bytes) depends on the type of the port (it is the responsibility of the developer to appropriately deal with the byte array size).
Methods for status checking	
`String queryStatus();`	Queries the status of the ARE system (i.e., OK,

	FAIL, etc).
`String registerLogListener();`	Registers an asynchronous log listener to the ARE platform. Returns an ID which is used to identify the data packets concerning the registered log messages.
`void unregisterLogListener(String logListenerID);`	Unregisters the specified log listener ID from asynchronous log messages.

Table 22: The ASAPI Server Interface

The “ASAPI Client”, on the other side, extends the functionality provided by the “ASAPI Server” by adding commands for discovering and connecting to ARE instances:

- Discover and connect/disconnect to AREs.

The corresponding commands of the “ASAPI Client” are shown (also in JAVA) in the table below:

Method	**Description**
Methods to discover and connect/disconnect to AREs	
`InetAddress [] searchForAREs();`	Searches in the local area network (LAN) for available instances of the ARE. The exact protocol for discovery can vary (e.g., it could be based on UPnP, SLP, or a custom protocol).
`ASAPI_Server connect(InetAddress ipAddress);`	Connects to the ARE at the specified IP address. The method returns an instance of the ASAPI Server interface (described above), masking the functionality provided by the target ARE through ASAPI.
`void disconnect(ASAPI_Server asapi_server);`	Disconnects from the specified instance of the ASAPI Server, invalidating the reference.

Table 23: The ASAPI Client Interface

Furthermore, a more complex architecture involves also the deployment of the “ASAPI Native” module. The “ASAPI Native” interface complements the functionality of the “ASAPI Client” by allowing for native hosting of components directly on the client device. The resulting architecture is depicted in Figure 38. As shown in this figure, the ACS (or a custom client) can use directly both the “ASAPI Client” and the “ASAPI Server” interfaces, in order to realize more complex functionality, e.g., that includes hosting some components locally (utilizing client-specific functionality) and interfacing them with remote components deployed in the server. Commonly, the ASAPI Native functionality is provided by a platform-specific technology, such as a DLL library in the primary client platform (i.e., Windows).

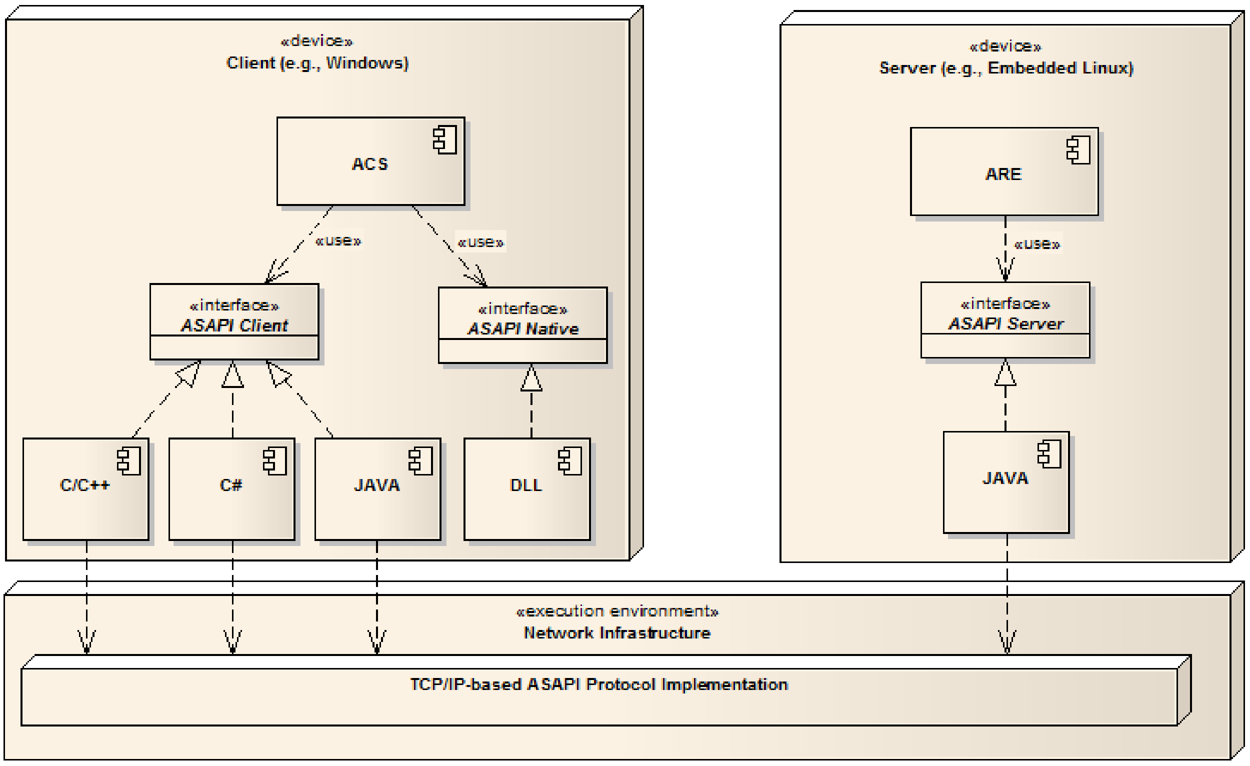


Figure 38: Extended architecture of ASAPI with use of the Native module by the client

Finally, the actual implementations of the “ASAPI Server” and the “ASAPI Client” interfaces leverage either an existing technology (like the Protocol Buffers, XML RPC and Thrift mentioned above) or a custom solution that allows transformation of messages in a suitable form for TCP/UDP/IP communication. An example of such custom implementation is abstracted in the “ASAPI Protocol” interface. The main methods of the ASAPI Protocol are listed in Table 24 and the extended definition of the interface in Appendix A. This interface specifies methods for transforming an invocation into an appropriate *packet* that can subsequently be transformed to and from a byte array, suitable for communication over the network.

Method	**Description**
`Packet createPacket(MessageType messageType, Object [] arguments);`	Creates a packet to be communicated over TCP/UDP/IP, with the specified MessageType and the specified arguments.
`Object [] parsePacket(Packet packet);`	Creates an array containing the arguments encoded in the payload of the packet, as specified by the MessageType
`void send(Packet packet);`	Sends the provided packet to the connected peer, after it is first transformed to a byte array.
`Packet receive();`	Polls the first packet pending for reception from the connected peer. It is assumed that received data are transformed to packets and placed in a FIFO list, pending reception. This is required because of the asynchronous nature of receiving packets. The packet interface abstracts the basic concept for communication between ASAPI clients and servers. It always features a single MessageType. Also, implementations of this interface typically

	specify methods for converting from and to byte arrays, suitable for network transfer.

Table 24: The ASAPI Protocol Interface

4.4.1 ASAPI and ARE Interconnection

The following figure shows the ASAPI protocol connection to the ARE and the ASAPI native interface which provides certain functions for PC AT developers aside the ARE (e.g. mobile phone access or special PC peripherals which will be investigated during WP6). The native interface can provide well defined functions (as sending an SMS) which do not imply signal processing plugins of the ARE, and can thus be accomplished directly on the PC.

As soon as the AsTeRICS Runtime Environment and the embedded platform are involved, the ASAPI command and data protocol can be used to interact with the ARE.

The ASAPI protocol is a platform independent specification per se. To implement an ASAPI client, templates in JAVA (server side) and C# (client side) will be provided as an early outcome of WP4.

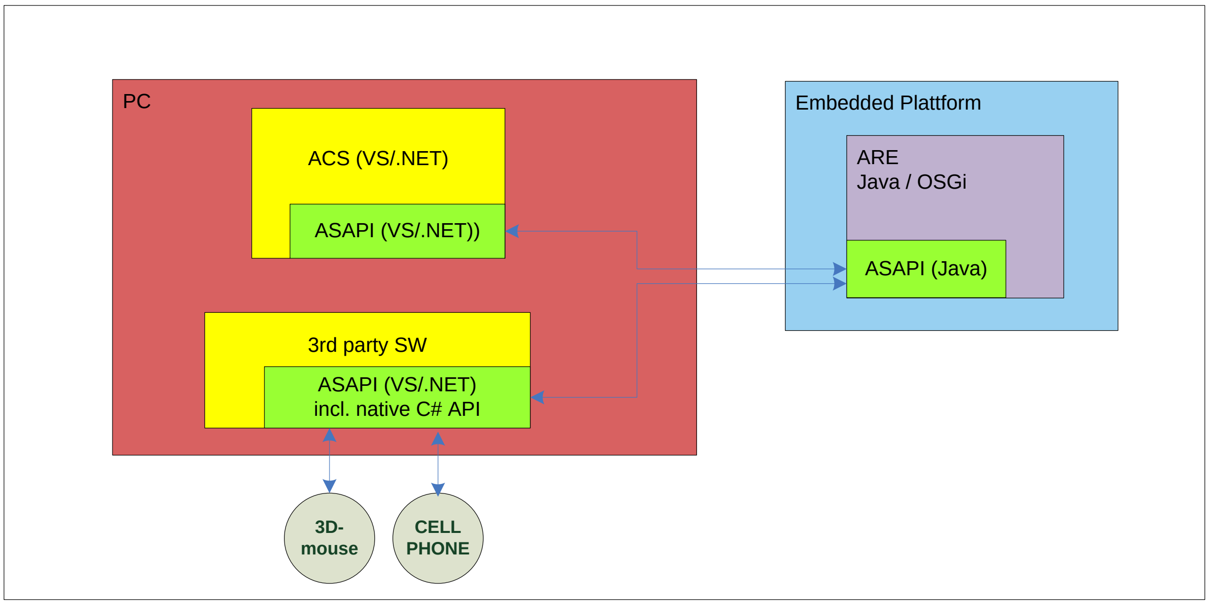


Figure 39: ASAPI client implementations with/without native functions

The following diagrams show two possible scenarios for ASAPI / ARE interconnection (1), one for the configuration of the ARE and one for the operation thereof (2).

Usually, these scenarios will involve primary and secondary users of the AsTeRICS system:

- AT developers use the Configuration Suite to set up the model for the desired AT-configuration, tailored to a specific use case or end user (1),
- End users start the system (power up the embedded platform or start the ARE on PC or netbook) to get their desired AT-configuration (which operated stand alone or in connection with 3rd-party applications on a PC or netbook (2).

4.4.1.1 ASAPI and ARE in the configuration process

Setting up a model

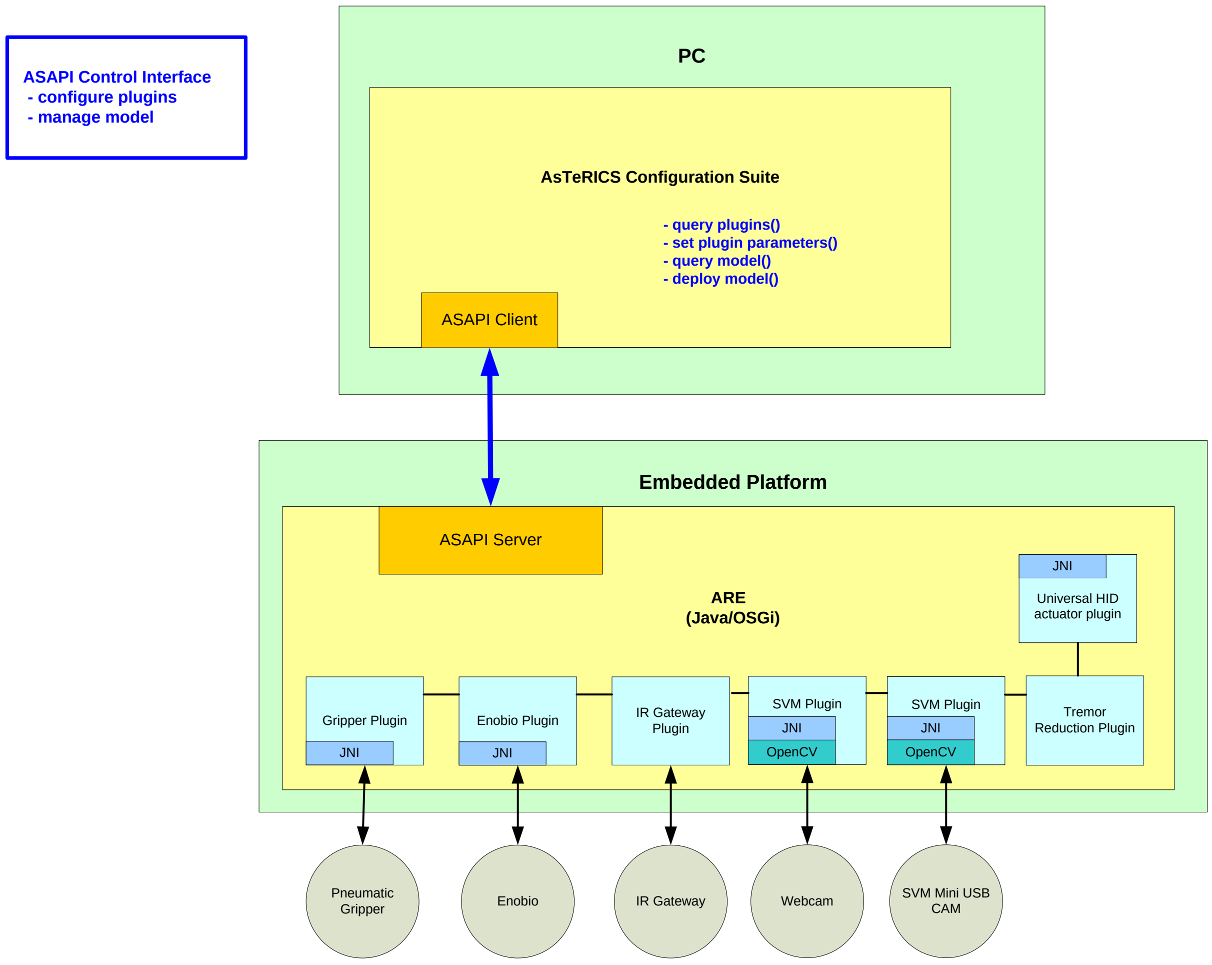


Figure 40: AsTeRICS configuration scenario, model setup

Figure 40 shows the configuration process of the AsTeRICS Runtime Environment by the AsTeRICS Configuration Suite via ASAPI. The ASAPI client of the ACS connects to the ARE's ASAPI server. It queries the available plugins and their parameters. (In the above figure, some exemplary plugins are shown for demonstration purpose).

The ACS offers dynamic graphic configuration dialogs to the user, which allows adjustment of all the plugin parameters. Plugins can be graphically connected. This process does not need any functional representation of the plugins, only a description of the plugins' ports, data types and parameters. All these setup actions are performed via ASAPI control commands. The finalized model can be deployed to the ARE.

Monitoring, verifying and adjusting a model

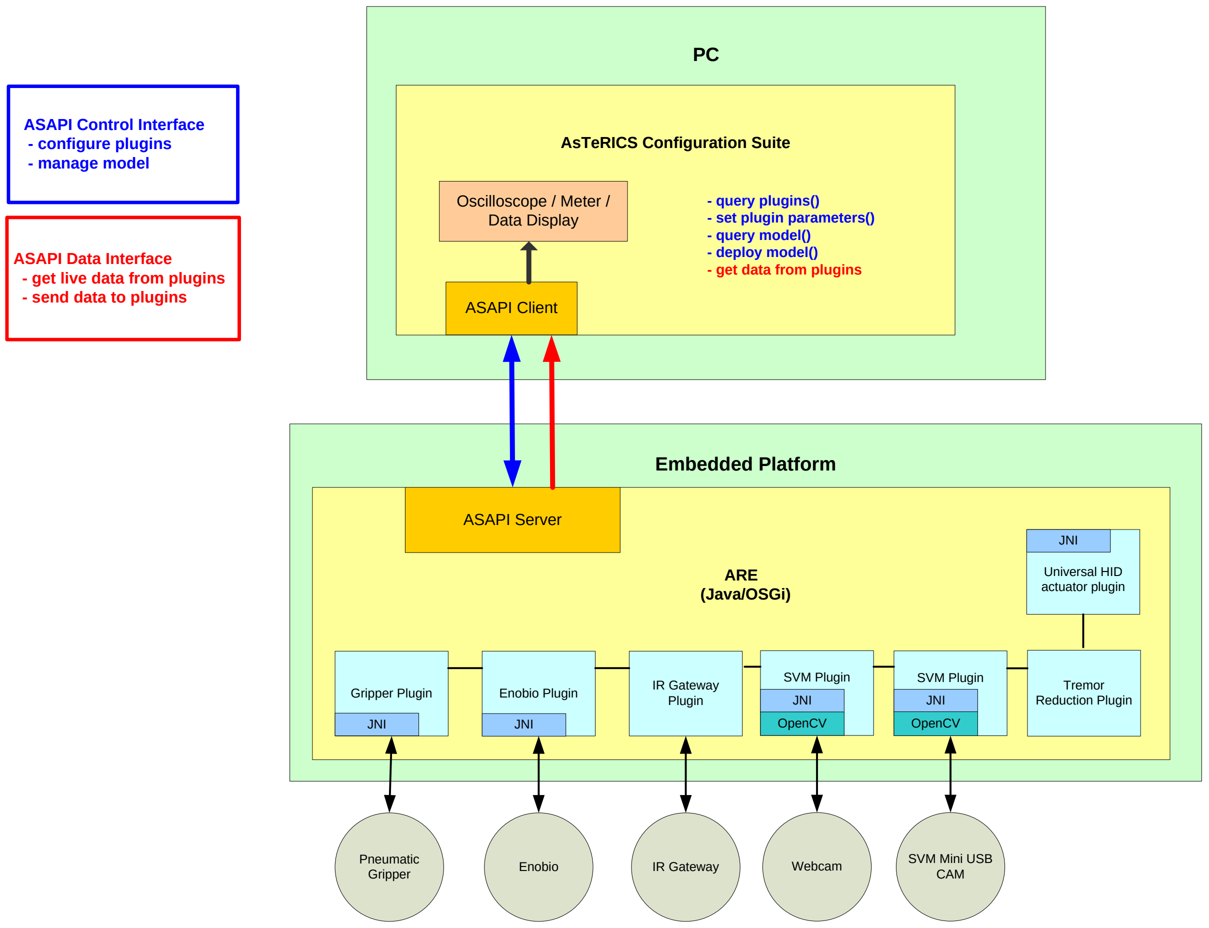


Figure 41: AsTeRICS configuration scenario, verification and error checking

To verify the setup process, a data connection to desired plugins can be opened in the Configuration Suite. Thus, live sensor values and their transformation due to the applied signal processing plugins can be monitored using feedback elements of the ACS like oscilloscope or bar graphs. Parameters of the plugins can be modified using ASAPI control commands until the desired behaviour of the ARE is present.

Additionally to the live data transmission for feedback purpose, status and error information can be queried from the ARE to determine the state of particular plugins.

ASAPI and ARE in the runtime system

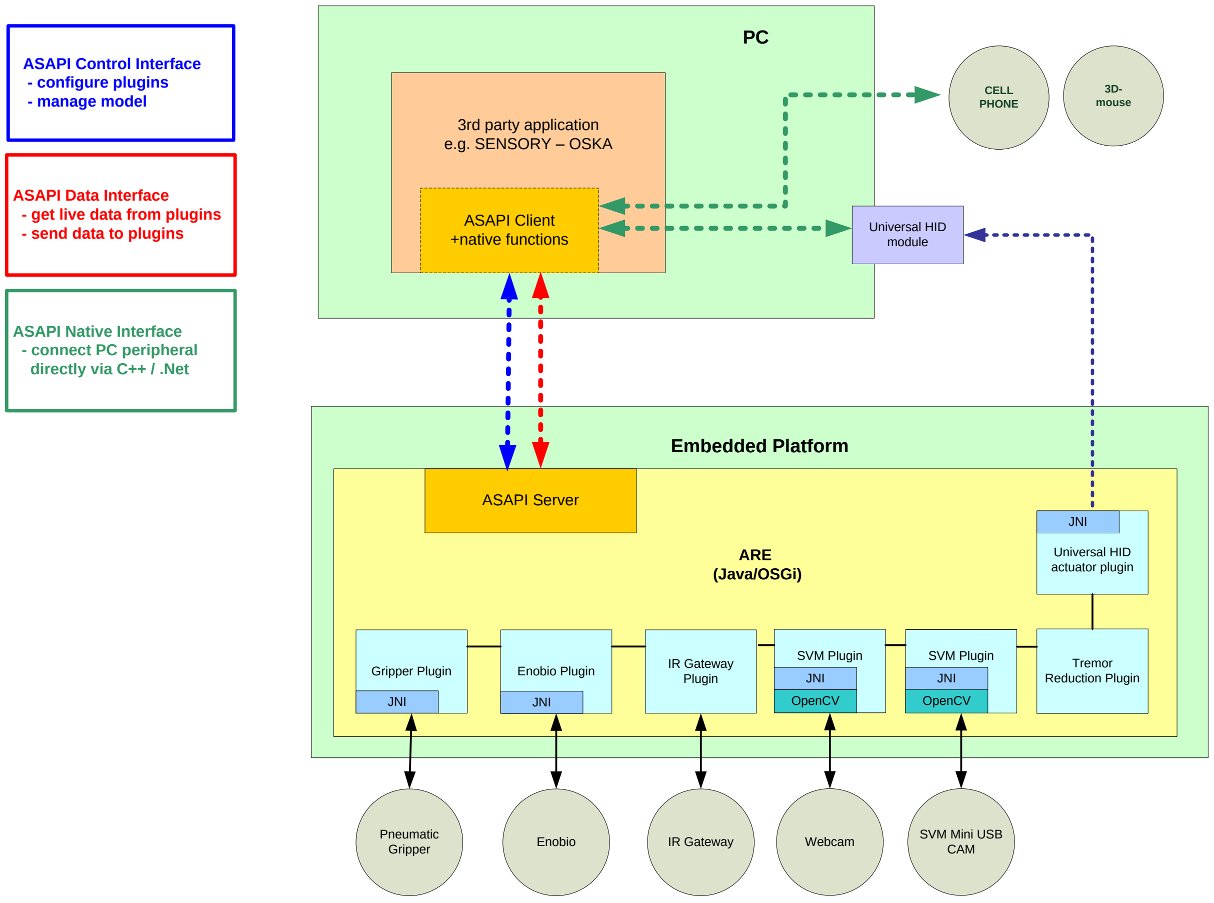


Figure 42: AsTeRICS runtime scenario

A fully configured ARE can run as a stand-alone process providing its functionality or communicate with PC AT-software. A connection between ARE and ACS is no longer required at that time.

The above runtime scenario consists of a configured ARE, with connected plugins that interface the external sensors (Enobio, SVM) and actuators (pneumatic gripper, IR gateway).

Third party applications running on the PC can optionally:

- query or send data by using ASAPI data commands
- use the ASAPI native interface to access supported PC peripherals like mobile phone, 3D mouse
- use ASAPI to connect to the running ARE and send control commands to modify model or plugin settings

If the Universal HID actuator USB dongle is used, the PC application can obtain data from the embedded platform via a mouse, joystick or keyboard hook which is provided via the ASAPI native interface (thereby omitting a dedicated TCP/IP connection to the ARE via the ASAPI client).

4.4.2 Native ASAPI

Native ASAPI is a software development kit for 3rd party developers to help them adapt their application for people with motor disabilities. Native ASAPI will be delivered as a set of DLL libraries and COM objects for the Microsoft Windows Operating system. Native ASAPI works independently of ARE. The main native ASAPI functions are:

- Interfacing the devices described in the table below and adapting them for the needs of individual users,
- Adapting the standard computer keyboard and mouse for the needs of individual users,
- Providing keyboard hook procedures, including for prediction dictionaries.
- Key-press emulation,
- Implementing selected algorithms from the ARE, such as the tremor reduction algorithm.

The table below lists the devices interfaced by Native ASAPI:

Nr	Device	Priority
1.	3D Mouse	M
2.	Windows Mobile phone	H
3.	USB HID / PS/2 Keyboard	L

Table 25: Devices interfaced by Native ASAPI

Native ASAPI will be based on the experience gained in developing of ARE, so some of its requirements may change during development. The integration of other devices with Native ASAPI will be considered. The description of Native ASAPI functions, and examples of Native ASAPI use will be delivered in the Native ASAPI reference document. The actual design and development of Native ASAPI will be done in the next period, and the detail specification of the interface will be described in D2.2.

5 BNCI Evaluation Suite

The realization of complex Brain/Neuronal Computer Interaction functionalities is mainly based on the application of pattern recognition methodologies for the interpretation of brain and neuronal physiological signals. Therefore they are normally formed by modules implementing a set of functional stages: data pre-processing, feature extraction and classification/decision making. In this context it is well known [DOR] that such complex systems have to be adapted to the user in order to achieve feasible performance. This fact does not only apply to the adaptation of the classifier parameters to some particular subject, which is achieved through the realization of training procedures, but to the structure of the classifying system as well. Results of benchmark competitions, e.g. BCI competition, show that the same system presents a great degree of performance variability with respect to the operating subject. If we take for instance the results of the IV BCI competition analyzing the performance in the classification of motor imagery, we can observe that performance index used for evaluation, which is defined in the interval [0,1], ranges from 0.27 to 0.77 over the different subjects for the winning methodology.

The fact mentioned in the former paragraph is supported by theoretical considerations. The so-called No Free Lunch and Ugly Duckling Theorems [DUD] state that there is no a priori information on the quality of the features to be used in a classification system, and no a priori superiority of one classifier over other ones. Therefore these theorems boost the consideration of pattern recognition as an experimental research field. The consequence for our work in AsTeRICS is clear. Since we cannot establish a general outperforming classification system, there is a need for a potential user of the system to select the components of this classification system. The BNCI Evaluation Suite will implement a performance evaluation procedure. This procedure will be applied on different configurations of the pattern recognition system. The main goal of the evaluation performance is the selection of a pattern recognition structure adapted for a particular application undertaken by a particular user.

The BNCI Evaluation Suite has to be developed within WP4 of AsTeRICS project. The Evaluation Suite attains the off-line performance evaluation of BCI frameworks. Performance will be measured with standard performance measures in BCI, e.g. kappa, TPR/FPR, ITR. These frameworks may later serve the configuration of BCI systems that may work on-line for a particular user. Therefore the Evaluation Suite shall be used for prototyping BCI systems, but not for implementing them in real-time. In this context the Evaluation Suite will extend the available StarEEGlab toolkit with some functionalities missing. Moreover it should include as well functionalities for taking data from disk, which is acquired following some BNCI protocol, and prototype a BNCI system. The requirement table (see Table 8) includes a list of functionalities that shall or might be implemented within WP4.

The functional requirements detailed in Table 8 are a consequence of the state of the art analysis included in AsTeRICS D2.4. It includes the consortium requirements, but mainly the functionalities that the AsTeRICS team need to achieve three goals. The first goal is to complete the StarEEGlab with some functionality that seems to be used currently in state of the art systems. The second goal is to reproduce the results of some selected works of the state of the art. The third goal is to reproduce some standard BCI paradigms, namely P300 and motor imagery.

The two main functionalities to be covered by the BNCI Evaluation Suite are the feature extraction and the classification. It will also contain a number of routines for pre-processing data and for evaluating the performance of the different classifiers that will be implemented.

The Evaluation Suite is meant for easily extracting features from physiological data and for their posterior classification. Simple routines will be integrated in order to make the whole process as automate as possible. An important part of the suite will deal with the data pre-processing in order to be able to read the data in the format most usually used by acquisition devices to be integrated in the AsTeRICS system.

5.1 BNCI Evaluation Suite User Classes and Requirements

The BNCI Evaluation Suite is intended for AsTeRICS technical users. Therefore its main users will be technicians involved in the configuration of the AsTeRICS system with previous programming skills in low as well as high level abstraction languages.

The users have expressed very loose requirements for the BNCI Evaluation Suite, most due to lack of knowledge on the Evaluation Suite purpose at the time of completion of this deliverable and of the BNCI technologies itself. However, following common sense reasoning, we can state that it shall be capable of prototyping BNCI complex systems following some of the currently available protocols, i.e. P300, SSVEP, Mu rhythm. The Evaluation Suite might be capable of saving output parameters of included functionalities, which could be eventually used in the recall phase of the BNCI systems taken into account. Prototyping of non-classical user interfaces based on BNCI might be possible with the Evaluation Suite.

5.2 BNCI Evaluation Suite Assumptions and Dependencies

The BNCI Evaluation Suite will be developed for integrating functionalities of pre-existing BNCI toolkits. The dependencies will be mentioned in the functions documentation in order to ensure its applicability.

5.3 BNCI Evaluation Suite Processing Architecture

The software will be organized in 3 different packages. The different functions included in the packages usually flow in a sequential processing chain, although its stand alone utilization should be ensured in the design process. A brief description of these packages follows:

- Data pre-processing: Preparation of data as acquired by sensory units.
- Feature Extraction: Measurement of different signal features.
- Classification: Different Computational Intelligence Techniques for mapping the input data into different categories.

These packages can be functionally organized in order to implement a BNCI framework as depicted in the following figure.

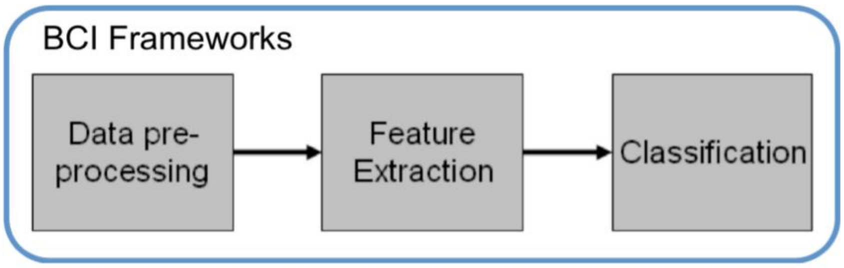


Figure 43: Block Diagram of BNCI Evaluation Suite packages

The Data Pre-processing package is formed by 2 sub-packages. The first one includes the routines for putting the input data in the right format, while the second deals with pre-processing techniques such as filters, decimation and/or artifacts corrections.

The feature extraction process deals with the measurement of signal characteristics. As a further goal the extracted features should be significant for the classification problem to be solved. It is worth to mention that it is not always easy to separate the pre-processing from the feature extraction. Different types of features can be distinguished depending on the number of EEG channels taken into account for computing a particular feature. Therefore we will group the feature extraction procedures in different sub-packages depending on the number of EEG channels taken into account in their computation.

The classification package will include two sub-packages. In the first one, all functions for classifying data will be included. In the second one, different functionalities for the performance evaluation of the classification result will be implemented.

6 Summary and Conclusions

In this document, the requirements and specifications of the hardware and software architecture of the AsTeRICS system have been defined. The comprehensive list of requirements is grouped into hardware and software sections for the main AsTeRICS components and will guide the development process for both system prototypes. The subsequent specification sections give detailed insight into the architectural concept for the software layers and describe the features of hardware components which either have been chosen yet or are being designed in course of the project.

In particular, the decision for a suitable computing platform has been accomplished, following the state of the art analysis of deliverable 2.4 [3]. The Kontron pico-ITX Single Board Computer (SBC) has been selected because of the high computational power, sufficient memory (which is needed e.g. for the SVM) and compatibility to Microsoft Windows, which makes the development of drivers, especially in the area of HID devices much easier. This selection also has disadvantages, in particular the bigger form factor and the higher power consumption of the Atom platform compared to e.g. OMAP-3 systems. These disadvantages probably will vanish with the next generation of ultra tiny, low power computing platforms, which can be utilized for Prototype 2 of the AsTeRICS project.

In sections 3.2 to 3.4, important hardware components and their integration into the system have been described. This includes the Communication Interface Modules (CIMs) which extend the interfaces of the Embedded Platform, the Universal HID actuator for mouse, keyboard and joystick emulation, and the Smart Vision Module with its design stages for first and second prototype.

The software for the AsTeRICS Runtime Environment (ARE) is based upon Java OSGi, which will make it possible to build a very flexible and component based system, where new plugins can easily be added to or removed from the system model. All parts of the model (plugins, attached ports, channels between the ports) will be stored in an XML-based data structure.

The AsTeRICS Configuration Suite (ACS) can create, edit, read, store and deploy system models. The ACS allows modification of plugin properties and connection of plugins via their ports. Additionally, graphical feedback components (e.g. oscilloscope, diagram) will be available in the ACS, so that the users can get live feedback helping them to find the optimal configuration. The development technology used for the ACS will be Microsoft .Net because of an excellent accessibility support (e.g. UI-Automation technology).

The communication between ACS and ARE will be established via the AsTeRICS Application Programming Interface (ASAPI), a client/server model, where the ARE will act as server and the ACS will act as client. ASAPI provides functions to transmit the data model, start/stop services, connect plugins, set properties and many more. ASAPI can be used also by 3rd party software applications to integrate AsTeRICS functionalities. Additionally, a native ASAPI implementation will provide certain functions without a connected ARE. In a first stage, these native functionalities will comprise the connection to a mobile phone and the connection to a 3D mouse peripheral.

Furthermore, the requirements and features of the BNCI Evaluation Suite have been defined.

The current document will serve as a central resource for the whole development process, especially for the software implementation of the middleware components and the interaction of different software and hardware entities of the AsTeRICS system. As such, the hardware requirements and specifications defined in this document provide the basis for subsequent development work in WP3, as the software requirements and specifications do for WP4 development.

In course of the upcoming tasks and work packages, certain changes and refinements to the outlined system architecture will be applied. This process is even intended by the chosen development methodology (user cantered design, see Description of Work [1]). All such modifications to the given architectural concept will be described in the architectural specification of the second system prototype which is subject to deliverable D2.3.

References

1 AsTeRICS Description of Work, technical annex 1 of the AsTeRICS Grant agreement

2 AsTeRICS Deliverable D2.3 – “Report on API-specification for sensors to be integrated into the AsTeRICS Personal Platform”

3 AsTeRICS Deliverable D2.4 – “Report on the State of the Art”

4 AsTeRICS Deliverable D4.7 – “Report on Feasibility of OSGi porting to the AsTeRICS Personal Platform Prototype 1”

5 Damasio, A. (2003) Looking for Spinoza: Joy, Sorrow, and the feeling Brain. Harcourt. Translated to French edition Odile Jacob 2003.

6 Developer Resources for Java Technology, “ Java: Primitive Data Types” [Online], Available: http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html. [Accessed: June, 07, 2010].

7 Microsoft .NET framework [Online], Available: http://www.microsoft.com/net/. [Accessed: June, 10, 2010].

8 .NET Framework 4 - Windows Presentation Foundation, “Introduction to WPF” [Online], Available: http://msdn.microsoft.com/en-us/library/aa970268.aspx. [Accessed: June, 10, 2010].

9 .NET Framework 4, “UI Automation and Microsoft Active Accessibility” [Online], Available: http://msdn.microsoft.com/en-us/library/ms788733%28v=VS.100%29.aspx. [Accessed: June, 10, 2010].

10 Wikipedia, the free encyclopedia, “Model-View-Controller” [Online], Available: http://en.wikipedia.org/wiki/Model-view-controller. [Accessed: June, 12, 2010].

11 Wikipedia, the free encyclopedia, “Model-View-ViewModel” [Online], Available: http://en.wikipedia.org/wiki/Model_View_ViewModel. [Accessed: June, 12, 2010].

12 Protocol Buffers - Google's data interchange format [Online], Available: http://code.google.com/p/protobuf. [Accessed: June, 14, 2010].

13 XML-RPC, “*Simple cross-platform distributed computing, based on the standards of the Internet.”* [Online], Available: http://www.xmlrpc.com/. [Accessed: June, 14, 2010].

14 Apache Thrift [Online], Available: http://incubator.apache.org/thrift/. [Accessed: June, 14, 2010].

15 Kontron AG, “Kontron Pico-ITX Single Board Computer specifications”, [Online], Available: http://emea.kontron.com/industries/infotainment/pos++poi/single+board+computers+sbcs/pitxsp.html [Accessed: June, 14, 2010]

16 Mimomonitors.com , MIMO USB-driven Monitor, “MIMO720-S description” [Online], Available: http://www.mimomonitors.com/products/mimo-720-s [Accessed: June, 16, 2010]

17 D. Li, J. Babcock and D.J. Parkhurst, openEyes: a low-cost head mounted eye-tracking solution, Proceedings of the 2006 Symposium on Eye Tracking Research and Applications Conference (ETRA), San Diego, California. New York: ACM Press, pp. 95-100, 2006.

18 Z. Yun, Z. Xin-Bo, Z. Rong-Chun, Z. Yuan and Z. Xiao-Chun, EyeSecret: An inexpensive but high performance auto-calibration eye tracker: Proceedings of the Proceedings of the 2008 Symposium on Eye Tracking Research and Applications Conference (ETRA), March 26-28, 2008, Savannah, Georgia. New York: ACM Press, pp. 103-106, 2008.

19 J. Babcock and J. Pelz, Building a lightweight eyetracking headgear: Proceedings of the 2004 Symposium on Eye Tracking Research and Applications Conference (ETRA), March 22-24, 2004, San Antonio, Texas. New York: ACM Press, pp. 109-114, 2004.

20 W. J. Ryan, A. T. Duchowski and S. T. Birchfield, Limbus/pupil switching for wearable eye tracking under variable lighting conditions: Proceedings of the 2008 symposium on Eye tracking research & applications (ETRA), March 26-28, 2008, Savannah, Georgia. New York: ACM Press, pp. 61-64, 2008.

21 D. W.Hansen and Q. Ji, In the Eye of the Beholder: A Survey of Models for Eyes and Gaze. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.32, No.3, pp. 478-500, 2010.

22 A. Duchowski, Eye Tracking Methodology: Theory and Practice. Springer-Verlag, 2003.

23 D., Li and D. J. Parkhurst, Open-source software for real-time visible-spectrum eye tracking: Proceedings of the 2nd Conference on Communication by Gaze Interaction (COGAIN), Turin, Italy, pp. 18-20, 2006.

24 A. Villanueva, R. Cabeza, S. Porta, M .Böhme, D. Droege and F. Mulvey, “D5.6 Report on New Approaches to Eye Tracking. Summary of new algorithms”, Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 5.6, 2008. [Online], Available: http://www.cogain.org/w/images/c/c9/COGAIN-D5.6.pdf. [Accessed: June 7 2010].

25 Pertech, [Online], Available: http://en.pertech.fr/. [Accessed: June 7 2010].

26 G. Litos, X. Zabulis and G. Triantafyllidis, Synchronous Image Acquisition based on Network Synchronization: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW),June 17-22, 2006, New York, New York. Washington DC: IEEE Computer Society, pp.167, 2006.

27 Mobisense Systems, “Camera Modules”.. [Online], Available: http://www.mobisensesystems.com/fics/MBS032_datasheet.pdf. [Accessed: June 7 2010].

28 C4AV, “Camera Modules”. [Online], Available: http://centerforartificialvision.com/Digital_Camera_Modules_C4AV.php. [Accessed: June 7 2010].

29 IDS Imaging, “USB cameras”. [Online], Available: http://www.ids-imaging.de/frontend/products.php?interface=USB&family=LE&lang=e. [Accessed: June 7 2010].

30 InterSense, “InertiaCube Sensors”. [Online], Available: http://www.intersense.com/InertiaCube_Sensors.aspx?. [Accessed: June 7 2010].

31 Sparkfun. [Online], Available: http://www.sparkfun.com/. [Accessed: June 7 2010].

32 SensorDynamics, “6 DoF IMU”. [Online], Available: http://sensordynamics.cc/cms/cms.php?pageId=73. [Accessed: June 7 2010].

33 The BioSig Project [Online], Available: http://biosig.sourceforge.net/ [Accessed: June 9 2010].

34 EEGLAB: interactive Matlab toolbox for processing continuous and event-related EEG, MEG and other electrophysiological data. [Online], Available: http://sccn.ucsd.edu/eeglab/. [Accessed: June 9 2010]

35 The Java Tutorials: “Primitive Data Types [Online]”. Available: http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html. [Accessed: June 15 2010]

Appendix A

1 The ASAPI Server Interface in JAVA

```java
package org.asterics.middleware.asapi;

/**
 * Define an abstract service, describing the basic commands available in the
 * ASAPI protocol (excluding commands for detecting and connecting to AREs,
 * which are limited to the client side of the protocol only). This interface
 * reflects the functionality provided by the server side.
 *
 * @author Nearchos Paspallis [nearchos@cs.ucy.ac.cy]
 * @author Konstantinos Kakousis [kakousis@cs.ucy.ac.cy]
 *         Date: Jun 8, 2010
 *         Time: 2:56:56 PM
 */
public interface ASAPI_Server
{
    // --------------- Methods to setup and deploy a model ------------------ //

    /**
     * Returns an array containing all the available (i.e., installed) component
     * types. These are encoded as strings, representing the absolute class
     * name (in Java) of the corresponding implementation.
     *
     * @return an array containing all available component types
     */
    public String [] getAvailableComponentTypes();

    /**
     * Returns a string encoding the currently deployed model in XML. If there
     * is no model deployed, then an empty one is returned.
     *
     * @return a string encoding the currently deployed model in XML
     */
    public String getModel();

    /**
     * Deploys the model encoded in the specified string into the ARE. An
     * exception is thrown if the specified string is either not well-defined
     * XML, or not well defined ASAPI model encoding, or if a validation error
     * occured after reading the model.
     *
     * @param modelInXML a string representation in XML of the model to be
     * deployed
     * @throws AsapiException if the specified string is either not well-defined
     * XML, or not well defined ASAPI model encoding, or if a validation error
     * occured after reading the model
     */
    public void deployModel(final String modelInXML)
            throws AsapiException;

    /**
     * Deploys a new empty model into the ARE. In essence, this is equivalent
     * to creating an empty model and deploying it using
     * {@link #deployModel(String)}. This results to freeing all resources in
     * the ARE (i.e., if a previous model reserved any).
     */
    public void newModel();

    /**
     * It starts or resumes the execution of the model.
     *
     * @throws AsapiException if an exception occurs while validating and
     * starting the deployed model.
     */
    public void runModel()
        throws AsapiException;

    /**
     * Briefly stops the execution of the model. Its main difference from the
     * {@link #stopModel()} method is that it does not reset the components
     * (e.g., the buffers are not cleared).
     *
     * @throws AsapiException if the deployed model is not started already, or
     * if the execution cannot be paused
     */
    public void pauseModel()
        throws AsapiException;

    /**
     * Stops the execution of the model. Unlike the {@link #pauseModel()}
     * method, this one resets the components, which means that when the model
     * is started again it starts from scratch (i.e., with a new state).
     *
     * @throws AsapiException if the deployed model is not started already, or
     * if the execution cannot be stopped
     */
    public void stopModel()
        throws AsapiException;

    // ------------ End of methods to setup and deploy a model -------------- //

    // ------------- Methods to read and edit the model ---------------------- //

    /**
     * Returns an array that includes all existing component instances in the
     * model (even multiple instances of the same component type).
     *
     * @return an array of all the IDs of the existing component instances
     */
    public String [] getComponents();

    /**
     * Returns an array containing the IDs of all the channels that include the
     * specified component instance either as a source or target.
     *
     * @param componentID the ID of the specified component instance
     * @return an array containing the IDs of all the channels which include
     * the specified component instance
     */
    public String [] getChannels(final String componentID);

    /**
     * Used to create a new instance of the specified component type, with the
     * assigned ID. Throws an exception if the specified component type is not
     * available, or if the specified ID is already defined.
     *
```

```java
     * @param componentID the unique ID to be assigned to the new component
     * instance
     * @param componentType describes the component type of the component to be
     * instantiated
     * @throws AsapiException if the specified component type is not available,
     * or if the specified ID is already defined
     */
    public void insertComponent(final String componentID, final String componentType)
        throws AsapiException;

    /**
     * Used to delete the instance of the component that is specified by the
     * given ID. Throws an exception if the specified component ID is not
     * defined.
     *
     * @param componentID the ID of the component to be removed
     * @throws AsapiException if the specified component ID is not
     * defined
     */
    public void removeComponent(final String componentID)
        throws AsapiException;

    /**
     * Returns an array containing the IDs of all the ports (i.e., includes
     * both input and output ones) of the specified component instance. An
     * exception is thrown if the specified component instance is not defined.
     *
     * @param componentID the ID of the specified component instance
     * @return an array (non empty) containing the IDs of all the ports of the
     * specified component instance
     * @throws AsapiException if the specified component instance is not defined
     */
    public String [] getAllPorts(final String componentID)
        throws AsapiException;

    /**
     * Returns an array containing the IDs of all the input ports of the
     * specified component instance. An exception is thrown if the specified
     * component instance is not defined.
     *
     * @param componentID the ID of the specified component instance
     * @return an array (possibly empty) containing the IDs of all the input
     * ports of the specified component instance
     * @throws AsapiException if the specified component instance is not defined
     */
    public String [] getInputPorts(final String componentID)
        throws AsapiException;

    /**
     * Returns an array containing the IDs of all the output ports of the
     * specified component instance. An exception is thrown if the specified
     * component instance is not defined.
     *
     * @param componentID the ID of the specified component instance
     * @return an array (possibly empty) containing the IDs of all the output
     * ports of the specified component instance
     * @throws AsapiException if the specified component instance is not defined
     */
    public String [] getOutputPorts(final String componentID)
```

```
        throws AsapiException;

    /**
     * Creates a channel between the specified source and target components and
     * ports. Throws an exception if the specified ID is already defined, or
     * the specified component or port IDs is not found, or if the data types
     * of the ports do not match. Also, an exception is thrown if there is
     * already a channel connected to the specified input port (only one
     * channel is allowed per input port).
     *
     * @param channelID the ID to be assigned to the formed channel
     * @param sourceComponentID the ID of the source component
     * @param sourcePortID the ID of the source port
     * @param targetComponentID the ID of the target component
     * @param targetPortID the ID of the target port
     * @throws AsapiException if either of the specified component or port
     * IDs is not found, or if the data types of the ports do not match, or if
     * there is already a channel connected to the specified input port
     */
    public void insertChannel(final String channelID,
            final String sourceComponentID,
            final String sourcePortID,
            final String targetComponentID,
            final String targetPortID)
        throws AsapiException;

    /**
     * Removes an existing channel between the specified source and target
     * components and ports. Throws an exception if the specified channel is
     * not found.
     *
     * @param channelID the ID of the channel to be removed
     * @throws AsapiException if the specified channel ID is not found
     */
    public void removeChannel(final String channelID)
        throws AsapiException;

    // ---------- End of methods to read and edit the model ------------------ //

    // ------ Methods to read and edit properties (even while running) ------ //

    /**
     * Reads the IDs of all properties set for the specified component.
     *
     * @param componentID the ID of the component to be checked
     * @return an array (possibly empty) with all the property keys for the
     * specified component, or null if the specified component is not found
     */
    public String [] getComponentPropertyKeys(final String componentID);

    /**
     * Returns the value of the property with the specified key in the
     * component with the specified ID as a string.
     *
     * @param componentID the ID of the component to be checked
     * @param key the key of the property to be retrieved
     * @return the value of the property with the specified key in the
     * component with the specified ID as a string, or null if the specified
     * component is not found
     */
    public String getComponentProperty(
            final String componentID, final String key);

    /**
```

```
     * Sets the property with the specified key in the component with the
     * specified ID with the given string representation of the value.
     *
     * @param componentID the ID of the component to be checked
     * @param key the key of the property to be set
     * @param value the string-representation of the value to be set to the
     * specified key
     * @return the previous value of the property with the specified key in the
     * component with the specified ID as a string, or an empty string if the
     * property was not previously set, or null if the specified component is
     * not found
     */
    public String setComponentProperty(
            final String componentID, final String key, final String value);

    /**
     * Reads the IDs of all properties set for the specified port.
     *
     * @param portID the ID of the port to be checked
     * @return an array (possibly empty) with all the property keys for the
     * specified port, or null if the specified port is not found
     */
    public String [] getPortPropertyKeys(final String portID);

    /**
     * Returns the value of the property with the specified key in the
     * component with the specified ID as a string.
     *
     * @param componentID the ID of the component to be checked
     * @param portID the ID of the port to be checked
     * @param key the key of the property to be retrieved
     * @return the value of the property with the specified key in the
     * component and port with the specified IDs as a string, or null if the
     * specified component or port are not found
     */
    public String getPortProperty(final String componentID,
            final String portID, final String key);

    /**
     * Sets the property with the specified key in the port with the
     * specified ID with the given string representation of the value.
     *
     * @param componentID the ID of the component to be checked
     * @param portID the ID of the port to be checked
     * @param key the key of the property to be set
     * @param value the string-representation of the value to be set to the
     * specified key
     * @return the previous value of the property with the specified key in the
     * component and port with the specified IDs, as a string, or an empty
     * string if the property was not previously set, or null if the specified
     * component or port are not found
     */
    public String setPortProperty(final String componentID,
            final String portID, final String key, final String value);

    /**
     * Reads the IDs of all properties set for the specified component.
     *
     * Reads the IDs of all properties set for the specified channel.
     *
     * @param channelID the ID of the channel to be checked
     * @return an array (possibly empty) with all the property keys for the
     * specified channel, or null if the specified channel is not found
     */
    public String [] getChannelPropertyKeys(final String channelID);
```

```java

    /**
     * Returns the value of the property with the specified key in the
     * channel with the specified ID as a string.
     *
     * @param channelID the ID of the channel to be checked
     * @param key the key of the property to be retrieved
     * @return the value of the property with the specified key in the
     * channel with the specified ID as a string, or null if the specified
     * channel is not found
     */
    public String getChannelProperty(final String channelID, final String key);

    /**
     * Sets the property with the specified key in the channel with the
     * specified ID with the given string representation of the value.
     *
     * @param channelID the ID of the channel to be checked
     * @param key the key of the property to be set
     * @param value the string-representation of the value to be set to the
     * specified key
     * @return the previous value of the property with the specified key in the
     * channel with the specified ID as a string, or an empty string if the
     * property was not previously set, or null if the specified channel is
     * not found
     */
    public String setChannelProperty(
            final String channelID, final String key, final String value);

    // --- End of methods to read and edit properties (even while running) -- //

    // ---------- Methods to interface directly to ports in the ARE --------- //

    /**
     * Registers a remote consumer to the data produced by the specified source
     * component and the corresponding output port. In the background, the ARE
     * forms a proxy component that is connected to the specified component and
     * port, which is utilized to communicate the data to the corresponding
     * remote consumer. This is similar to the proxy-based approach used in
     * Java RMI (see <a href="http://java.sun.com/developer/technicalArticles/RMI/rmi">
     * http://java.sun.com/developer/technicalArticles/RMI/rmi</a>
     * and <a href="http://today.java.net/article/2004/05/28/rmi-dynamic-proxies-and-evolution-deployment">
     * http://today.java.net/article/2004/05/28/rmi-dynamic-proxies-and-evolution-deployment
     * </a>).
     *
     * @param sourceComponentID the ID of the source component instance
     * @param sourceOutputPortID the ID of the source output port from where
     * data will be communicated
     * @return remote consumer ID - a unique ID used to select the data received
     * for this link
     * @throws AsapiException if the specified component ID or port ID are not
     * defined
     */
    public String registerRemoteConsumer(final String sourceComponentID,
            final String sourceOutputPortID)
        throws AsapiException;

    /**
     * Unregisters the remote consumer channel with the specified ID.
     *
     * @param remoteConsumerID the ID of the channel to be unregistered
     * @throws AsapiException if the specified channel ID cannot be found
     */
    public void unregisterRemoteConsumer(final String remoteConsumerID)
        throws AsapiException;

    /**
     * Registers a remote producer to provide data to the specified target
     * component and the corresponding input port. In the background, the ARE
     * forms a proxy component that is connected to the specified component and
     * port, which is utilized to receive the data from the corresponding
     * remote producer.
     *
     * @param targetComponentID the ID of the target component instance
     * @param targetInputPortID the ID of the target input port where data will
     * be communicated to
     * @return remote producer ID - a unique ID used to mark the data sent
     * @throws AsapiException if the specified component ID or port ID are not
     * found, or if the input port already has an assigned channel
     * @see #registerRemoteConsumer(String, String)
     */
    public String registerRemoteProducer(final String targetComponentID,
            final String targetInputPortID)
        throws AsapiException;

    /**
     * Unregisters the remote producer channel with the specified ID.
     *
     * @param remoteProducerID the ID of the channel to be unregistered
     * @throws AsapiException if the specified channel ID cannot be found
     */
    public void unregisterRemoteProducer(final String remoteProducerID)
        throws AsapiException;

    /**
     * This method is used to poll (i.e., retrieve) data from the specified
     * source component and its corresponding output port. Just one tuple of
     * data is returned. The actual amount of data (i.e., in bytes) depends
     * on the type of the port (it is the responsibility of the developer to
     * appropriately deal with the byte array size).
     *
     * @param sourceComponentID the ID of the source component
     * @param sourceOutputPortID the ID of the corresponding output port
     * @return an array of bytes that includes the requested tuple of data
     * (can be null if no data were produced)
     * @throws AsapiException if the specified component ID or port ID are not
     * available
     */
    public byte [] pollData(final String sourceComponentID,
            final String sourceOutputPortID)
        throws AsapiException;

    /**
     * This method is used to pull (i.e., send) data to the specified target
     * component and its corresponding input port. Just one tuple of
     * data is communicated. The actual amount of data (i.e., in bytes) depends
     * on the type of the port (it is the responsibility of the developer to
     * appropriately deal with the byte array size).
     *
     * @param targetComponentID the ID of the target component
     * @param targetInputPortID the ID of the corresponding input port
```

```
         * @param data an array of bytes that includes the communicated tuple of
         * data (cannot be null)
         * @throws AsapiException if the specified component ID or port ID are not
         * available
         */
        public void sendData(final String targetComponentID,
                final String targetInputPortID,
                final byte [] data)
            throws AsapiException;

        // ------- End of methods to interface directly to ports in the ARE ----- //

        // ------------------- Methods for status checking ----------------------- //

        /**
         * Queries the status of the ARE system (i.e., OK, FAIL, etc)
         *
         * @return a string representation of the current status of the ARE
         */
        public String queryStatus();

        /**
         * Registers an asynchronous log listener to the ARE platform. Returns an
         * ID which is used to identify the data packets concerning the registered
         * log messages.
         *
         * @return an ID which is used to identify the data packets concerning the
         * registered log messages
         */
        public String registerLogListener();

        /**
         * Unregisters the specified log listener ID from asynchronous log messages.
         *
         * @param logListenerID the ID of the log listener to be removed
         */
        public void unregisterLogListener(final String logListenerID);

        // ---------------- End of methods for status checking ------------------- //
    }
```

2 The ASAPI Client Interface in JAVA

```
package org.asterics.middleware.asapi;

import java.net.InetAddress;

/**
 * Define an abstract service, describing the basic commands available in the
 * ASAPI protocol (excluding commands for detecting and connecting to AREs,
 * which are limited to the client side of the protocol only). This interface
 * reflects the functionality provided by the client side.
 *
 * @author Nearchos Paspallis [nearchos@cs.ucy.ac.cy]
 * @author Konstantinos Kakousis [kakousis@cs.ucy.ac.cy]
 *         Date: Jun 15, 2010
 *         Time: 10:27:58 AM
 */
```

```java
public interface ASAPI_Client extends ASAPI_Server
{
    /**
     * Searches in the local area network (LAN) for available instances of the
     * ARE. The exact protocol for discovery can vary (e.g., it could be based
     * on UPnP, SLP, or a custom protocol).
     *
     * @return an array with the IP addresses where ARE instances were
     * discovered
     */
    public InetAddress [] searchForAREs();

    /**
     * Connects to the ARE at the specified IP address. The method returns an
     * instance of the {@link org.asterics.middleware.asapi.ASAPI_Server}
     * interface, masking the functionality provided by the target ARE through
     * ASAPI.
     *
     * @param ipAddress the IP address of the target ARE instance
     * @return an instance of {@link org.asterics.middleware.asapi.ASAPI_Server}
     * masking the ASAPI functionality provided by the target ARE
     */
    public ASAPI_Server connect(final InetAddress ipAddress);

    /**
     * Disconnects from the specified instance of the
     * {@link org.asterics.middleware.asapi.ASAPI_Server}, invalidating the
     * reference.
     *
     * @param asapi_server the instance of
     * {@link org.asterics.middleware.asapi.ASAPI_Server} to be disconnected
     */
    public void disconnect(final ASAPI_Server asapi_server);
}
```

3 Example of ASAPI Protocol Implementation in JAVA

```java
package org.asterics.middleware.asapi;

/**
 * The ASAPI_Protocol is a helper class that provides the functionality required
 * to interface implementations of the {@link ASAPI_Client} and the
 * {@link ASAPI_Server} interfaces with the actual code that enables actual
 * communication at the network level through TCP/UDP/IP.
 *
 * @author Nearchos Paspallis [nearchos@cs.ucy.ac.cy]
 * @author Konstantinos Kakousis [kakousis@cs.ucy.ac.cy]
 *         Date: Jun 15, 2010
 *         Time: 11:26:20 AM
 */
public interface ASAPI_Protocol
{
    /**
     * Creates a packet to be communicated over TCP/UDP/IP, with the specified
     * {@link MessageType} and the specified arguments.
     *
     * @param messageType the type of the message
     * @param arguments the arguments defined in the message signature
     * @return an instance of {@link Packet} encoding the specified message and
     * arguments
     */
    Packet createPacket(final MessageType messageType, final Object [] arguments);

    /**
     * Creates an array containing the arguments encoded in the payload of the
     * packet, as specified by the {@link MessageType}.
     *
     * @param packet the packet of which the payload is to be parsed
     * @return an array containing the arguments encoded in the payload of the
     * packet
     */
    Object [] parsePacket(final Packet packet);

    /**
     * Sends the provided packet to the connected peer, after it is first
     * transformed to a byte array.
     *
     * @param packet the packet to be transmitted
     */
    void send(final Packet packet);

    /**
     * Polls the first packet pending for reception from the connected peer. It
     * is assumed that received data are transformed to packets and placed in
     * a FIFO list, pending reception. This is required because of the
     * asynchronous nature of receiving packets.
     *
     * @return a packet corresponding to the received bytes
     */
    Packet receive();

    enum MessageType
    {
        // methods to setup and deploy a model
        GET_AVAILABLE_COMPONENT_TYPES("GET_AVAILABLE_COMPONENT_TYPES", 0x0001),
        GET_MODEL("GET_MODEL", 0x0002),
        DEPLOY_MODEL("DEPLOY_MODEL", 0x0003),
        NEW_MODEL("NEW_MODEL", 0x0004),
        RUN_MODEL("RUN_MODEL", 0x0005),
        PAUSE_MODEL("PAUSE_MODEL", 0x0006),
        STOP_MODEL("STOP_MODEL", 0x0007),
        // methods to read and edit the model
        GET_COMPONENTS("GET_COMPONENTS", 0x0008),
        GET_CHANNELS("GET_CHANNELS", 0x0009),
        INSERT_COMPONENT("INSERT_COMPONENT", 0x000a),
        REMOVE_COMPONENT("REMOVE_COMPONENT", 0x000b),
        GET_ALL_PORTS("GET_ALL_PORTS", 0x000c),
        GET_INPUT_PORTS("GET_INPUT_PORTS", 0x000d),
        GET_OUTPUT_PORTS("GET_OUTPUT_PORTS", 0x000e),
        INSERT_CHANNEL("INSERT_CHANNEL", 0x000f),
        REMOVE_CHANNEL("REMOVE_CHANNEL", 0x0010),
        // methods to read and edit properties (even while running)
        GET_COMPONENT_PROPERTY_KEYS("GET_COMPONENT_PROPERTY_KEYS", 0x0011),
        GET_COMPONENT_PROPERTY("GET_COMPONENT_PROPERTY", 0x0012),
        SET_COMPONENT_PROPERTY("SET_COMPONENT_PROPERTY", 0x0013),
        GET_PORT_PROPERTY_KEYS("GET_PORT_PROPERTY_KEYS", 0x0014),
        GET_PORT_PROPERTY("GET_PORT_PROPERTY", 0x0015),
        SET_PORT_PROPERTY("SET_PORT_PROPERTY", 0x0016),
        GET_CHANNEL_PROPERTY_KEYS("GET_CHANNEL_PROPERTY_KEYS", 0x0017),
        GET_CHANNEL_PROPERTY("GET_CHANNEL_PROPERTY", 0x0018),
        SET_CHANNEL_PROPERTY("SET_CHANNEL_PROPERTY", 0x0019),
        // methods to interface directly to ports in the ARE
        REGISTER_REMOTE_CONSUMER("REGISTER_REMOTE_CONSUMER", 0x001a),
        UNREGISTER_REMOTE_CONSUMER("UNREGISTER_REMOTE_CONSUMER", 0x001b),
        REGISTER_REMOTE_PRODUCER("REGISTER_REMOTE_PRODUCER", 0x001c),
        UNREGISTER_REMOTE_PRODUCER("UNREGISTER_REMOTE_PRODUCER", 0x001d),
        POLL_DATA("POLL_DATA", 0x001e),
        SEND_DATA("SEND_DATA", 0x001f),
        // methods for status checking
        QUERY_STATUS("QUERY_STATUS", 0x0020),
        REGISTER_LOG_LISTENER("REGISTER_LOG_LISTENER", 0x0021),
        UNREGISTER_LOG_LISTENER("UNREGISTER_LOG_LISTENER", 0x0022),
        // methods for communicating data in remote channels
        SEND_STREAMING_DATA("SEND_STREAMING_DATA", 0x0023),
        RECEIVE_STREAMING_DATA("RECEIVE_STREAMING_DATA", 0x0024);

        private final String messageType;
        private final int code;

        private MessageType(final String messageType, final int code)
        {
            this.messageType = messageType;
            this.code = code;
        }

        public int getCode()
        {
            return code;
        }

        public String toString()
        {
            return messageType + "(" + code + ")";
        }
    }

    /**
     * The packet interface abstracts the basic concept for communication
     * between ASAPI clients and servers. It always features a single
     * {@link org.asterics.middleware.asapi.ASAPI_Protocol.MessageType}. Also,
     * implementations of this interface typically specify methods (e.g.,
     * static) for converting from and to byte arrays, suitable for network
     * transfer.
     */
    interface Packet
    {
        public MessageType getMessageType();

        /**
         * Defined in the actual implementation of the {@link Packet} to allow
         * for getting the packet as a byte array, ready to be communicated.
         *
         * Used by {@link ASAPI_Protocol#send(Packet)}
         *
         * @return a byte array representation of the packet
         */
        // static byte [] getBytes();

        /**
         * Defined in the actual implementation of the {@link Packet} to allow
         * for creating a packet from a byte array, as it was received.
         *
         * Used by {@link ASAPI_Protocol#receive()}
         */
        // static Packet fromBytes();
    }
}
```