D7.1

Version: 1.00
Date: 13/01/2011
Author: Christian Martin (MLAB)
Organisation of lead contractor: MLAB
Dissemination Status: PU (Public)[i]

Project acronym:	ALIAS
Project name:	Adaptable Ambient Living Assistant
Call and Contract:	AAL-2009-2-049
Grant agreement no.:	215175
Project Duration:	01/07/2010 – 30/06/2013 (36 months)

Co-ordinator:	TUM	Technische Universität München (DE)
Partners involved in this deliverable:	MLAB	MetraLabs GmbH (DE)
	IUT	Technische Universität Ilmenau (DE)

This project is co-funded by the Ambient Assisted Living (AAL) Joint programme, by the German BMBF, the French ANR and the Austrian BMVIT

Once completed please e-mail to WP leader with a copy to

eric.bourguignon@tum.de and **frank.wallhoff@tum.de**

WP7 Pilots & System Integration- Objectives	Reminder
- To implement and design the ALIAS pilots and prototype - To integrate additional hardware - To integrate developed software modules, like the human-machine-interface and the dialog manager	

Dissemination Level of this deliverable (*Source: Alias Technical Annex p20 & 22*)	
PU	Public
Nature of this deliverable (*Source: Alias Technical Annex p20 & 22*)	
R	Report

Due date of deliverable	2010-12-31 (M6)
Actual submission date	2011-01-13
Evidence of delivery	Public website: http://www.aal-alias.eu/ Internal website: http://cotesys.mmk.e-technik.tu-muenchen.de/al/content/documents This is password protected. Access to the database can be granted to the relevant AAL CMU Scientific Officer or AAL national agency project officer.

Authorisation			
No.	**Action**	**Company/Name**	**Date**
1	Prepared	MLAB	27/12/2010
2	Reviewed	TUM	10/01/2011
3	Reviewed	IUT	12/01/2011
4	Approved	MLAB	13/01/2011
5	Released		

Table of contents

1. Executive Summary

An interactive mobile service robot (like the ALIAS robot) consists of a number of different software modules. Typical modules are a navigation system, a dialog manager and a human-machine-interface. To bring all these different modules together, a common software framework and/or control architecture is necessary.

Current mobile service robot applications use different modular and flexible control architectures for controlling the hardware and software components of the robot. There are different ways of sharing data and communication between the modules. Some of the existing architectures will be described in Chapter 2 of this deliverable.

Furthermore, the ALIAS robot prototypes are equipped with two PCs. The internal PC, which runs on Linux, and an additional Mac mini which runs on Windows. The different software modules of the ALIAS robots will run distributed on these two PCs. Therefore, the software framework must allow sharing common data between both PCs. Chapter 3 of this deliverable shows the distribution of the different modules on both PCs.

The deliverable covers the following topics:

- Chapter 2: Requirements on the software framework
- Chapter 3: Distribution of the different modules on the two PCs
- Chapter 4: Control architecture on the SCITOS PC
- Chapter 5: Control architecture on the Mac mini
- Chapter 6: Conclusion and Outlook

This deliverable describes the proposed software framework to the ALIAS partners. Progressing towards the first ALIAS prototypes the different modules will be integrated as proposed in this deliverable since there is a remaining risk that it might be necessary to modify the structure and distribution of the different modules, an updated version of the software architecture will be released within the deliverable D2.2 (M10).

2. State of the Art and Requirements

A very important component of each mobile robot is a good control architecture, which controls the whole robot. Beside the integration of modules for navigation and the Human-Robot Interaction skills, an often underestimated component is the application itself. Without a useful application the robot itself doesn't make sense.

Over the last years, many different robot system architectures were developed and introduced. A good overview about some older architectures is given in (Coste-Maniere & Simmons, 2000). On the one hand, there are some architectures consisting of up to three layers (like 3T, CLARAty or OROCOS), which are designed to deal with navigation problems and on the other hand there are some low-level architectures (like Player/Stage (Vaughan, Gerkey, & Howard, 2003)), which only describe a kind of robotic programming interface. Furthermore, there are existing systems in-between (like CARMEN (Montemerlo, Roy, & Thrun, 2003)), which consist of a simple architecture and a full programming interface.
A major problem of all theses system is that they are basically designed to connect and control the different modules and methods of a robot, but they are typically not designed to build a concrete robot application. As far as we know, none of these systems provide a possibility to integrate modules, which are required to build a real interactive mobile robot, like elements for a graphical user interface or a dialog management system. The only way to create an application for an interactive mobile service robot based on the systems mentioned above seems to be to create a step-by-step program, which controls the whole system.

This way, in the last years, several interactive mobile robots with specific architectures were built and introduced. One of these robots is GRACE (Graduate Robot Attending a ConferencE) (Simmons, et al., 2003), which was build to tackle the AAAI Robot Challenge in 2003. The robot's software consists of many programs communicating via IPC (Inter Process Communication) and CARMEN (Montemerlo, Roy, & Thrun, 2003). The different programs exchange messages, e.g. sensor information, commands, or events. Each program provides one skill of the robot. This distribution guarantees the reusability and the modularity of this system. But if someone wants to realize a new application with such an architecture, he has to know the whole system and all programs in detail.
The humanoid robot HERMES (Bischoff, 2000) is controlled by a behavior-based system (Arkin, 1998), which uses a situation module (situation assessment and behavior selection) as core of the whole system. Around this core a set of skills is used to control the robot. The skills and the situation module are communicating via events and messages. The architecture for HERMES was designed to be reusable, but as far as we know, it was never used for another robot.
The robot MOBSY (Zobel, et al., 2001) uses two levels for the software integration. The most abstract level is the task level, where the overall behavior of the system is determined. The second level contains the robot-specific methods, which provides the different skills of the robot. The skills will be used in the task level to execute the behaviors. The architecture of MOBSY seems to be specially designed for the appropriate task and is not reusable for other applications.

The Care-O-bot (Hans & Baum, 2001) uses hybrid architecture. The four most important components of the robot are the Man-Machine-Interaction module, a symbolic planner, a fact manager, and an execution module. Furthermore, the robot uses a database which contains all necessary information about the environment. Although this architecture contains a Man-Machine-Interaction module and is used for different (but similar) robots, it seems not to be easy to create new applications, because some important parts (like to symbolic planner and the fact manager) have to be modified for each new task.

In previous research of the ALIAS partners MLAB and IUT a new modular control architecture for mobile service robots was developed (Martin, Scheidig, Wilhelm, Schröter, Böhme, & Gross, 2005). This control architecture consists of four layers: A hardware layer, a layer for hardware interface and skills of the robot, an abstraction layer and an application layer.

In the last two years Willow Garage introduced a new system architecture called Robot Operating System (ROS, http://www.ros.org). Unlike the name suggests the system is not an operating system, but only a controlling architecture to provide the developer with basic services for robot applications. The foundations of this project are within the player/stage software, so they cover mainly low level functionality. The purpose of ROS is to provide easy to use inter-process communication and also give a huge range of services (like navigation, person tracking, transformations between different coordinate systems, and map building) over a unified interface. The data transfer is done using self defined "messages", containing raw or preprocessed data, which are sent over TCP/IP sockets and received by registered consumers. A simple to use publisher/subscriber mechanism is used to configure and establish the data flow. TCP/IP guarantees a high amount of distributed applications and causes also the main drawbacks: performance loss on data transfer using the same machine, and a high degree of "single agents" where the data flow of the whole application is difficult to reconstruct.

3. Distribution of the software modules on the two PCs

The proposed architecture of the ALIAS robot prototypes contains two PCs. The first PC is an industrial PC, which is integrated into the SCITOS robot hardware. On this PC the hardware of the robot (like motors and range finder sensors), the cameras and the microphones are connected. The second PC (a Mac mini, which is integrated on the ALIAS robot prototypes) has a connection to the Brain-Computer-Interface BCI device, the Nintendo Wii and the display. The SCITOS PC runs on Linux and the second PC on Windows. Figure 1 shows the architecture of the two PCs.

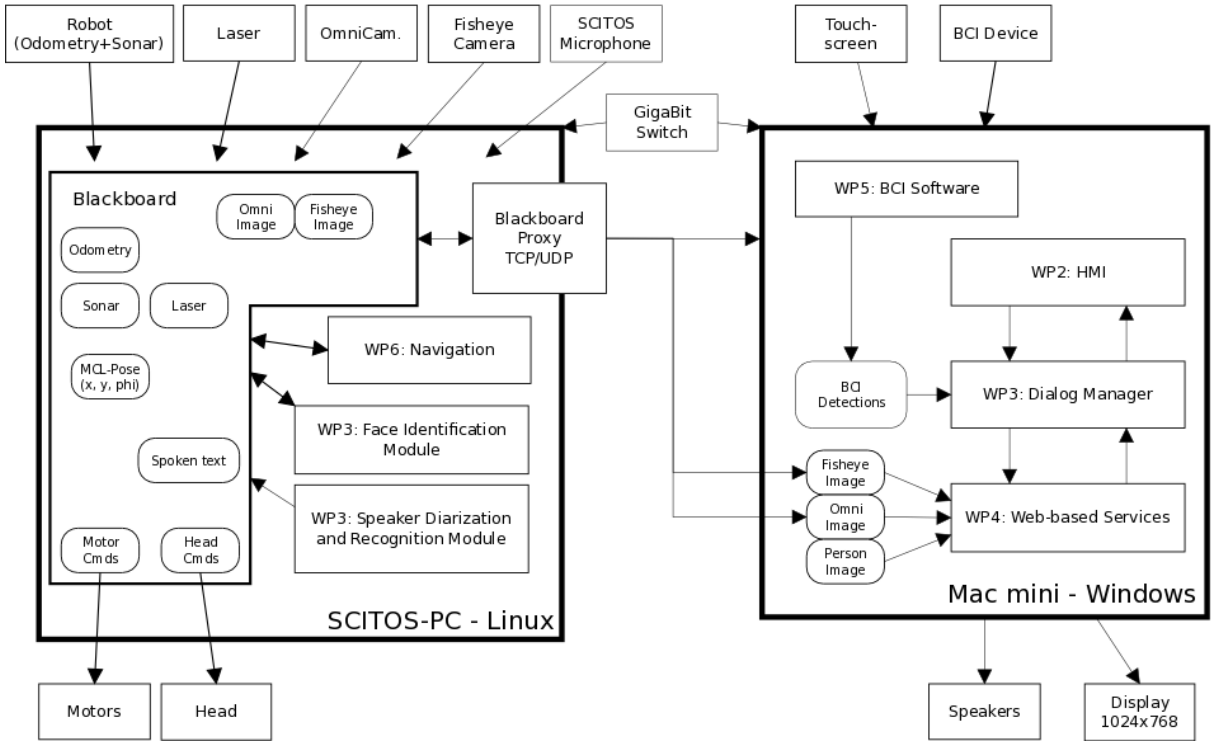

Figure 1: Distribution of the different software modules on the two on-board PCs.

The modules of the different work packages will run distributed on both PCs. On the SCITOS-PC mainly the navigation software from WP6 and the face identification and speech recognition software from WP3 will run. Furthermore, all drivers for the different sensors and actors of the robot have to be run on the SCITOS-PC.

Both PCs are connected via an onboard Gigabit-Ethernet connection, which guarantees a very fast transfer of the data between the two.

4. Software architecture on the Embedded PC

4.1. Existing control architecture

The existing software framework from MLAB and IUT is based on the control architecture described in (Martin, Scheidig, Wilhelm, Schröter, Böhme, & Gross, 2005). The skill layer uses a blackboard (Nii, 1998) architecture for communication between the different skills and hardware interfaces.

The blackboard is a flexible data control structure, which allows sharing any data structures (*blackboard data elements*) between independent modules (so called *blackboard clients*). The blackboard clients don't have to know each other. The different clients only need to know the data type(s) and the names of the data elements, which they want to use for exchanging data.

Figure 2 shows an example of a blackboard with some data elements. In this example, the blackboard client A is connected to some input hardware (e.g. a camera) and the client C communicates with some external hardware (e.g. the head of the robot). When the blackboard client A gets new data from the input device (1), it writes the data into the data element 1 (2). The blackboard clients B and C are informed about the new available data by the blackboard (3) + (4). The client B processes the new data and generates another new data element 2 (5) (e.g. a face detection on the current camera image). The blackboard client C does the same for the data element *n* (6) and generates a new output on the output hardware (7) (e.g. rotating the head of the robot).

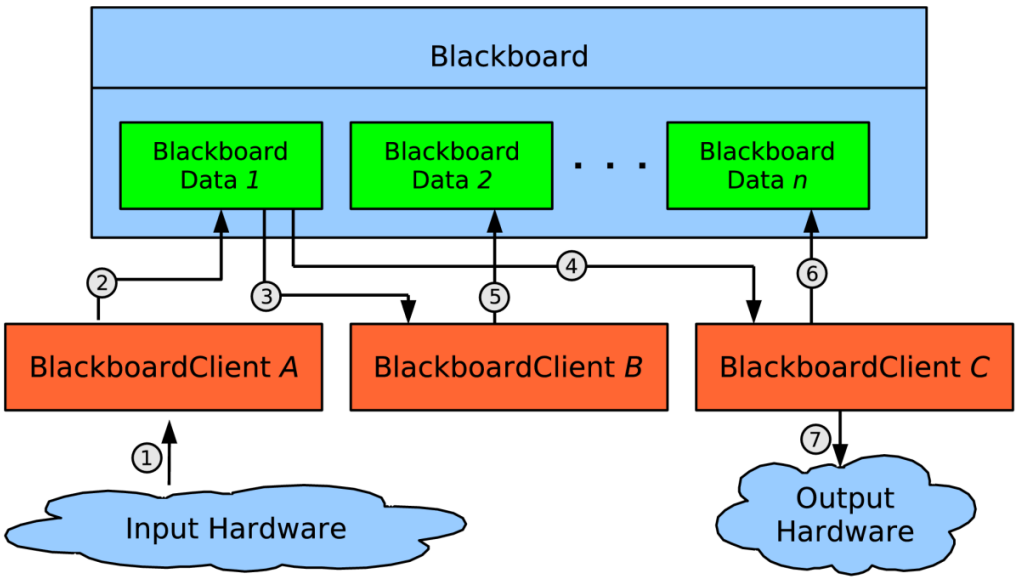

Figure 2: Structure of the blackboard approach. The blackboard contains data elements (green boxes), which can be accessed by different blackboard clients (red boxes).

The current implementation of the blackboard provides the following functions:

- Data management (creation, read- and write-access)
- Callback mechanism for handling and forwarding for data element changes (see below)
- Data logging and playback: The blackboard provides a mechanism to record all data elements (or a subset) into a log file. All data elements will be logged with a time stamp. The recorded log file can be played back in a way that the blackboard clients and registered callbacks can't determine, whether the data comes from a log file or directly from another client. This allows to record data with a mobile robot and to analyze the data later on a different computer without the hardware of the robot. The playback can run in different modes: The data can be played back in the same time scheme as it were recorded or in accelerated or decelerated mode.

Updates/modifications on a data element on the blackboard will be processed in a way that is visualized in Figure 3. After the blackboard client A acquires new data (1) from the hardware, the new data is written (2) by the client on the blackboard data element. This update will be added (3) to an update queue (4) of an internal blackboard update thread. This thread processes all data changes in a sequential order. The thread will send a callback (5) to the blackboard client B. Since this callback can block the update thread of the blackboard, the client B typically triggers (6) an own processing thread (7), which handles the new data.

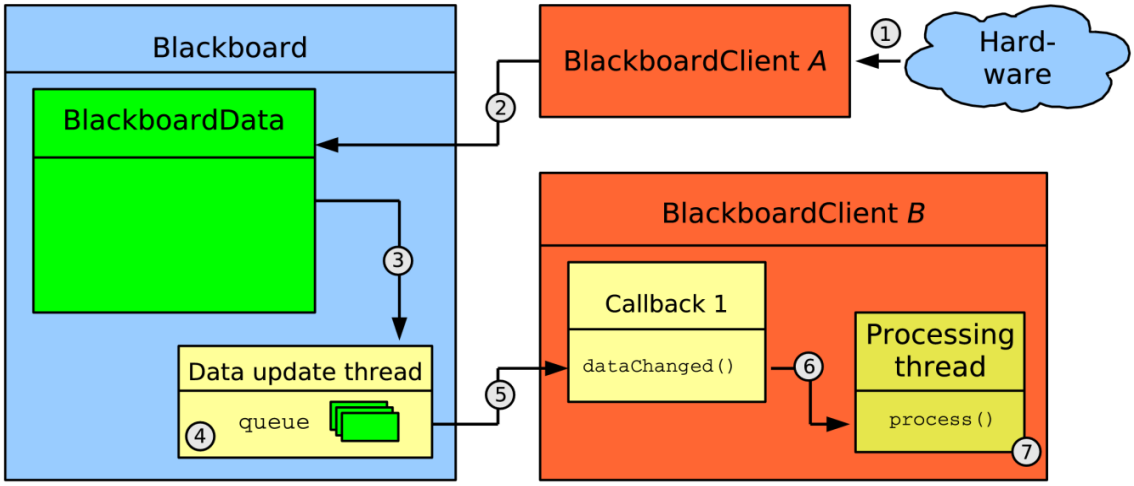

Figure 3: Data distribution. This figure shows how the data flows from a blackboard client A over the blackboard to a client B.

To access data on the blackboard from another program or computer, a so called *BlackboardProxy* (see Figure 1) can be used, which is realized as a blackboard client. This client allows read and write access to the blackboard via TCP or UDP by means of a binary protocol.

The blackboard is part of the Layer L_1 (*Skill Layer*) of the control architecture (see Figure 4). On top of the skill layer, the so called Abstraction Layer contains interfaces from the skills to the Application Layer.

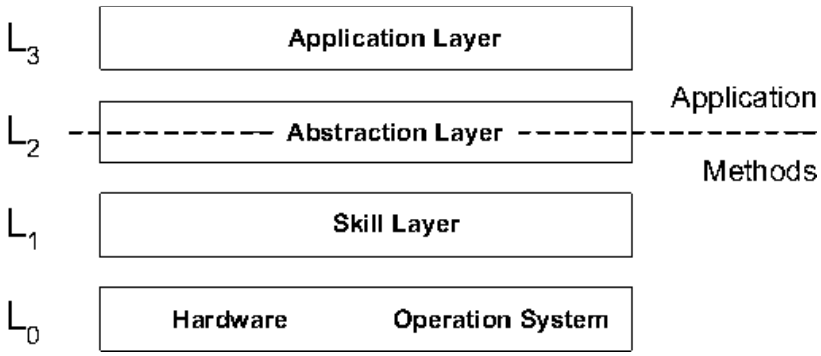

Figure 4: The four different layers of the control architecture.

The Application Layer is mainly based on the open-source scripting language AngelScript (http://www.angelcode.com/angelscript). This layer contains a state machine, a timer and event management module and a set of different behaviors. A typical behavior combines different skills to a more general behavior. For example, a *guiding behavior* for an interactive mobile service robot combines skills for navigation, person tracking and speech synthesis.

Experiences of the last years have shown the following advantages of the blackboard approach:

- All data on the blackboard is a share between the different modules.
- Every module can access data on the blackboard. Each module can have read and write access.
- The different modules only exchange data over the blackboard. The single clients don't have to know each other.

But also there are some disadvantages of the blackboard system:

- Due to erroneous use or a faulty module, it is possible, that the whole blackboard communication will be locked.
- Due to some implementation issues, typically each data element must be copied one time before it can be used for computation in a module. For small data field (like odometry data) this is not a problem, but for large images this can lead to a decreased performance.
- It doesn't allow a distributed application.
- The current implementation of the blackboard architecture only runs on Linux.

4.2. New control architecture

Due to the disadvantages of the blackboard approach and the limitations of the existing control architecture, MLAB and IUT started to develop a new architecture in spring 2010. The new architecture is developed to fulfill the following requirements:

- Allow more interoperability between different applications on the same computer or a remote computer
- Provide a small history of data elements per data channel for better data synchronization
- Provide an easy-to-use interface to establish data communication between different skills
- Lower the message distribution overhead and do not allow single threads, skills or applications to block other threads/skills or applications
- Allow user transparent inter- and intra-process communication at maximum speed (on lossless data connections)
- Auto-lock and unlock shared data elements without blocking other publishers or subscribers
- Allow work on the original data element, no need to copy the element first

To achieve these additional requirements, a classical blackboard structure is not appropriate any more. In a blackboard architecture one writer per data element is possible and multiple readers can act upon one element at a time. If the writer blocks infinitely no reader can access the element and if a reader blocks the element no writer can access the element any more. To overcome these drawbacks we use a history of data elements for access. Multiple readers can request access to this element and are normally granted with access to the most actual element in the history. While one or more readers block this element slot, a writer can access an old slot and fill in an actual version of the element. This slot is now automatically the most actual slot and any new read requests will get access to that slot. A well known example of such a mechanism is the double-buffering of graphics hardware. While the monitor needs to read out the pixel information 60 times per second, the graphics card can only provide new data every half a second or even slower. In such a case the monitor (or video DAC) has exclusive access to the most actual *complete* version of the screen image, which is many times the same buffer, while the graphics hardware can lock its write buffer until a new image is complete. After finish writing the graphics hardware can release the buffer and the buffer is ready to be read.

For each slot and such a chain of buffers, a full copy of an object has to be created. This has the inconvenience of wasting memory while keeping the read and write accesses independent and unable to lock, but the number of buffers only depends on the number of consumers and producers and the time they operate on these data. Consumers can operate on the same data; while producers need exclusive write access to a data segment (see figure 5). If consumers and producers operate at different times on the same element, one element is sufficient to use, like it is done in the blackboard concept. If there is overlap in the time intervals, the element is accessed and the optimal length of a buffer chain depends on the number of producers, consumers and the accessing times of all participants only.

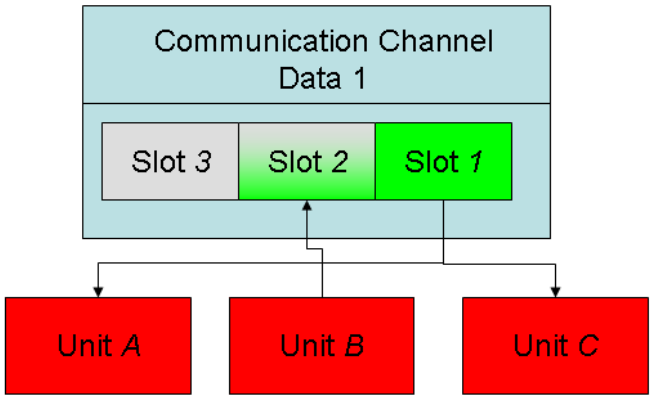

Figure 5: A communication channel where three units communicate, one producer and two readers. While the producer has exclusive access to write new data (Slot 2) the consumers are still able to read (Slot 1) and do not have to block their processing.

To calculate this needed length beforehand is a complicated task and depends on the application. But luckily it is easy to detect a possible blocking situation. One situation would be that all slots are covered with readers or writers and a writer wants to publish new data and the second situation is that a reader wants to access data and all slots are covered with writers. Only the first situation can be resolved by adding an extra slot and no further disturbances can be expected. In the latter situation the buffer chain appears empty for the reader, since no slot is finished. This will cause a reader error.

This mechanism will guarantee a non-blocking access to data elements. But it does not allow the communication between different processes, since no memory is shared and the buffers of one process are unknown to other processes. So an additional mechanism of synchronizing different buffer chains over process borders is needed to realize a distributed application. Here the consistent de/serialization has to be core components to distribute an application.

Based on the framework for distributed applications and blocking-free communication channels, the communication part of the architecture is established and can be used to realize functionalities around that, to get a robot running in its core functionality. To improve the independency of the different software modules, some basic functionalities, like mathematic functions, image classes, geometry operations, simple robot sensor data, encryption engines and XML parsers will be developed in a common *Base*. These basic functionalities do not necessarily depend on the communication framework, since they provide very low level functions - like matrix operations or image folding - and are used on a library level only. The benefit is here, that no dependencies to the framework are existing and it is easy to separate the communication from the basic functions. Both parts could then be used to create communicating entities (so called *Units*) that provide certain functionalities.

There are two kinds of units: those who reside as toolboxes and those who reside inside application domains like navigation, person- and object tracking. Toolboxes are tools to solve *whole classes* of problems, like least square regression estimators or image feature calculation. The important point is

that these toolboxes could be reused by many other units and that these toolboxes do more complicated work than the basic functions do.

When solutions become very specialized, like tracking a person, building a map, create decisions in dialog situation, localize a robot etc., these solutions are assembled inside so called *domains*. These domains are specialized toward *one class* of problems (like object tracking, navigation or dialog). These domains are allowed to use toolboxes, the communication framework and also the basic functionalities. It is not allowed to use units from other domains inside a domain. When it comes to mixture problems a domain is either allowed to use the outputs and interfaces from another domain (which are public accessible) or a new domain has to be created to support both worlds.

All these parts have to be provided by a new architecture and will increase the usability, stability and robustness of the robot control system.

5. Software architecture on the Mac mini

As described in section 3, the HMI, the dialogue manager and the Web-based services are located on the Mac mini,. The dialogue manager (also see deliverable D3.1) is the central component, which controls the final behavior of the robot. A graphical user interface (GUI) will be realized within the WP2 and WP3, which is responsible for the HMI and the Web-based services.

5.1 Dialogue Manager

Since the dialogue manager is the central component, which controls the behavior of the robot, it needs access to other different software modules from both PCs.

The connection to the blackboard (or the new architecture on the SCITOS-PC) will be realized by means of a blackboard client, which provides the relevant data to the dialogue manager using JSON via HTTP. The following data will be provided by the blackboard:

- Robot pose:
  - Position and view direction
- Status of the battery
- Error states
- Person position:
  - Position relative to the robot
  - View direction
  - Sitting / Status
  - Interaction flag

The face recognition module of WP3 will provide the following information to the dialogue manager:

- ID of the recognized person
- Probability density of the recognition

The connection to the BCI (WP5) and the graphical user interface (WP2) isn't defined in details yet. JSON, XMLRPC or REST will be used for both. The final interface for these components will be described in deliverable D2.2.

The dialogue manager itself is described in more details in deliverable D3.1.

5.2 Graphical User Interface

The current proposal of the involved ALIAS partners is to realize the whole graphical user interface as a HTML5 based web application, which will run on a local installed web server. Therefore, the following software modules have to be installed on the Mac mini:

- Web server:
  - Apache Server (http://httpd.apache.org) + Tomcat (http://tomcat.apache.org)
- Database:
  - MySQL (http://www.mysql.com) or Virtuoso (http://virtuoso.openlinksw.com/)
- Development environment:
  - JAVA 1.6+ (http://www.oracle.com/technetwork/java/javase/overview/index.html)
- Web browsers:
  - Firefox 3.6+ (http://www.mozilla.com/en-US/firefox/all.html)
  - Chrome 8+ (http://www.google.com/chrome)
- Web browser plugins:
  - Flash
  - QuickTime
  - Silverlight
- Multimedia codecs:
  - DivX and others
- Instant Messaging software:
  - ICQ (http://www.icq.com)
  - Skype (http://www.skype.com/)

Based on this HTML5 technology the following functionalities and applications can be easily realized:

- Web browsing
- E-Mail
- Calendar
- Notifications, reminders and alerts
- Banking
- Event management
- Entertainment: Video/TV on demand
- Gaming (using the Nintendo Wii)

Due to the proposed HTML5 framework on the Mac mini, all components have to be accessible by the Web Server using HTTP requests (RPC, XML-RPC, JSON, etc.). Therefore, for all software modules (including the dialogue manager, the blackboard and the GUI) appropriate interfaces have to be implemented in the ALIAS project.

6. Summary

This deliverable described the proposed software framework to the ALIAS partners. Due to the different requirements and applications on both PCs, each will run with different software architectures. On the embedded PC the existing blackboard architecture will be used and later replaced by a new more flexible architecture. On the MacMini the graphical user interface will be implemented as a web-based application using HTML5. Other modules (like the dialogue manager) will run as separate processes, which communicate over HTTP protocol. The connection between both PCs will be realized by means of the BlackboardProxy and the HTTP protocol.

Progressing towards the first ALIAS prototypes the different modules have to be integrated as proposed in this deliverable. Since there is a remaining risk that it might be necessary to modify the structure and distribution of the different modules, an updated version of the software architecture will be released within the deliverable D2.2 (M10).

7. References

Arkin, R. (1998). *Behavior-Based Robotics.* Cambridge, MA: MIT Press.

Bischoff, R. (2000). Towards the Development of "Plug-and-Play" Personal Robots. *IEEE-RAS International Conference on Humanoid Robots.*

Coste-Maniere, E., & Simmons, R. (2000). Architecture , the Backbone of Robotic Systems. *Proc. of IEEE International Conference on Robotics and Automation.*

Hans, M., & Baum, W. (2001). Concept of a Hybrid Architecture for Care-O-Bot. *Prof. of ROMAN*, (S. 407-411).

Martin, C., Scheidig, A., Wilhelm, T., Schröter, C., Böhme, H.-J., & Gross, H.-M. (2005). A new Control Architecture for Mobile Interaction Robots. *Prof. of the 2nd European Conference on Mobile Robots (ECMR)*, (S. 224-229).

Montemerlo, M., Roy, N., & Thrun, S. (2003). Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. *Prof. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, (S. 2436-2441).

Nii, N. P. (1998). Blackboard Systems. In *The Handbook of Artifical Intelligence 4* (S. 1-82).

Simmons, R., Goldberg, D., Goode, A., Montemerlo, M., Roy, N., Sellner, B., et al. (2003). Grace: An autonomouse robot for AAAI robot challenge. *AAAI Magazine, vol 24. no. 2* , 51-72.

Vaughan, R., Gerkey, B., & Howard, A. (2003). On device abstractions for portable, resuable robot code. *Prof. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Zobel, M., Denzler, J., Heigl., B., Nöth, E., Paulus, D., Schmidt, J., et al. (2001). MOBSY: Integration of Vision and Dialogue in Service Robots. *Computer Vision Systems, Proceedings Second International Workshop (ICVS)*, (S. 50-62).

[i] PUBLIC here means that the deliverable will be listed on the website, can only be communicated after request to the coordinator. This is meant to ensure secure dissemination of information in the interests of the consortium.