Deliverable 9.4

"Updated integration architecture"

Contract number**: FP7-215554 LIREC**

LIving with Robots and intEractive Companions

Start date of the project: 1st March 2008 Duration: 54 months

The research leading to these results has received funding from the European Community's Seventh Framework Programme (*FP7/2007-2013*) under *grant agreement* n° 215554.

Identification sheet

Project ref. no.	**FP7-215554**
Project acronym	LIREC
Status & version	Final
Contractual date of delivery	30th September 2010
Actual date of delivery	18th October 2010
Deliverable number	9.4
Deliverable title	Updated integration architecture
Nature	Report / Software
Dissemination level	**PU** Public
WP contributing to the deliverable	9, 8, 5
WP / Task responsible	"WP9/T9.2.1"
Editor	Michael Kriegel
Editor address	Heriot-Watt University, School of Mathematics and Computer Science, Earl Mountbatten Building, EH14 4AS, Edinburgh, United Kingdom
Author(s) (alphabetically)	Ruth Aylett (HWU), Pedro Cuba (INESC-ID), Amol Deshmukh (HWU), Kyron Du Casse (UH), David Griffiths (FoAM), Kheng Lee Koay (UH), Michael Kriegel (HWU), Iolanda Leite (INESC-ID), Lukasz Malek (WRUT)
EC Project Officer	Pierre-Paul Sondag
Keywords	Integration Architecture, Middleware
Abstract (for dissemination)	This document describes the implementation of the LIREC integration architecture, consisting of the 3 sub systems SAMGAR, CMION and FAtiMA.

CONTENTS

1 Purpose of the Document

After a review of the requirements for a LIREC integration architecture in D 9.1 and an architecture design specification in D 9.2, this deliverable 9.4 presents the actual implementation of the LIREC integration architecture. D 9.4 is primarily a software deliverable and this document is an accompanying overview of the delivered software. It outlines the motivation for creating an integrated architecture and the requirements for the architecture, before describing the architecture, its components and its application within the LIREC showcases. It also describes current work and planned future updates that will further increase stability and usability of the architecture. Please note that a brief complimentary overview of the architecture was already given in D 9.3, as the architecture was already fully implemented by then. During the relatively brief period that has passed between deliverables 9.3 and 9.4 the architecture has not been drastically changed and only matured in terms of robustness and stability bug fixes. This document therefore reiterates some of the content of D 9.3 but goes into more depth.

2 Objectives and Motivation

The motivation for creating a common integration architecture in the LIREC project was provided in detail in D 9.1 and D 9.2. Here we once again summarize the most important reasons.

2.1 Modular Software Development

When developing complex artificial companions, one typically deals with heterogeneous systems that consist of a variety of software modules. It would be unpractical to realize the implementation of such a companion as a single stand-alone module. This is especially true if the development work as in LIREC is topically and geographically distributed among work packages and partners. Work Package 3 develops competencies that allow companions to perceive their environment including users and their affective state, Work Packages 4 and 5 develop artificial intelligence that provides the companion with reasoning, action selection and memory capabilities and work package 6 is concerned with the development of robot hardware which also entails special control software. Actual companions built in the LIREC showcases make use of the outputs of all those aforementioned work packages. The purpose of an integration architecture is thus to provide a standardized way to tie those various software modules together. It provides module developers with a framework for developing their modules and allows companion developers to focus on selecting and configuring modules without worrying about how to integrate them with each other. The integrated architecture can be seen as highly specialized middleware for artificial companions. While there exists already a variety of middleware solutions for robotics systems, the remainder of this section presents innovative features that differentiate the LIREC integration architecture from other existing architectures. In the next section we will furthermore show how our architecture itself builds on top of established middleware solutions (ION and Yarp), extending them and adding functionality where required instead of reinventing the wheel.

2.2 An Architecture for Diverse Platforms

One distinguishing feature of the work set out by the LIREC consortium is the variety of companions being used. Existing Robotics architectures have been designed specifically to integrate components for robots but in LIREC we do not only deal with a variety of robotic companions but also virtual companions both for PCs and mobile devices. One requirement for the LIREC architecture that existing robotic architectures do not fulfil is thus its applicability to robotic, virtual and mobile companions. This is not only important because we do not want to replicate the work in implementing the architecture itself for different platforms but also to allow the reuse of competencies across those varying platforms.

2.3 An Architecture Supporting Migration

Another novel feature that the LIREC architecture has to support is migration. This can be formally defined as a companion's ability to a) save its state (this includes state information that is embodiment independent, e.g. personality and memory) b) transfer this state to another embodiment, c) load the state into the other embodiment and d) resume operation in the new embodiment in a way that users perceive the same identity in the new embodiment. The LIREC architecture can support this migration process by:

- Ensuring that mechanisms are in place that allow a companion to keep operating even in drastically different embodiments
- Managing the available embodiments and keeping track about which of them are inhabited
- Adding and removing of migration platforms in realtime
- Implementing the communication protocols for transferring a companion's state from one embodiment to another
- Providing interfaces and wrappers for modules (e.g. competencies, companion's mind) to store and restore their state

3 Architecture Description

3.1 State of Development

A working version of the LIREC architecture was fully implemented by the time of delivery of D9.3. Between then and the current state some stability and robustness bug fixes but no drastic conceptual changes have been introduced to it. All the functionality described in the remainder of this section describes the current state of development, has been implemented and tested and is working. However, further architecture refinement, driven by feedback from applying the architecture in the scenario is still underway. Section 4 briefly describes some of the future plans and current developments for the architecture which focus on issues identified by feedback from applying the architecture in the development of scenarios.

3.2 Overview

The complete LIREC architecture consists of 3 sub systems: FAtiMA, CMION and SAMGAR [Du Casse, Koay, Ho, and Dautenhahn]. Figure 1 gives a broad overview how the 3 systems interact with each other.

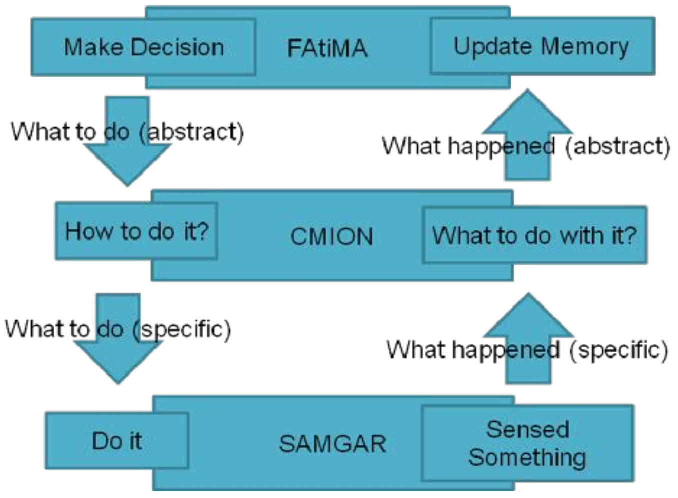


Figure 1: High - Level Overview of the 3 Systems the architecture is made up of

3.2.1 FAtiMA

FAtiMA is the name of the agent’s mind responsible for decision making, planning and emotions. Development of this component is part of WP5. The memory mechanisms developed in WP4 are also integrated within FAtiMA. FAtiMA is implemented in Java and development is lead by INESC-ID. The internal details of FAtiMA are covered in depth in D 5.3, which is why this document treats FAtiMA as a black box.

3.2.2 CMION

CMION stands for **C**ompetency **M**anagement with **ION**. The main purpose of this system is the translation of symbolic to sub-symblolic information and vice versa. That means CMION is responsible for selecting concrete competencies to perform symbolic commands selected by the mind and providing symbolic perception inputs to the mind. CMION is written in JAVA and built on top of INESC-ID's ION framework [Vala et al]. Development of CMION is lead by Heriot-Watt University.

3.2.3 SAMGAR

SAMGAR utilises the YARP [Fitzpatrick, Metta and Natale] framework that supports distributed computation and code re-use by structuring communication between modules (within and between the layers in the architecture). This allows modules with a high computational load to be cleanly decoupled and distributed between computers. SAMGAR adds to the functionality of YARP with a method of encapsulation, allowing multiple instances of identical modules to be present on shared networks without error for differing embodiments, therefore allowing code re-use and the ability for migration, along with a Graphical user interface to allow a more direct and more generic way to alter network parameters outside of hard code. SAMGAR is written in C++ and developed by University of Hertfordshire.

3.3 Three Level Model

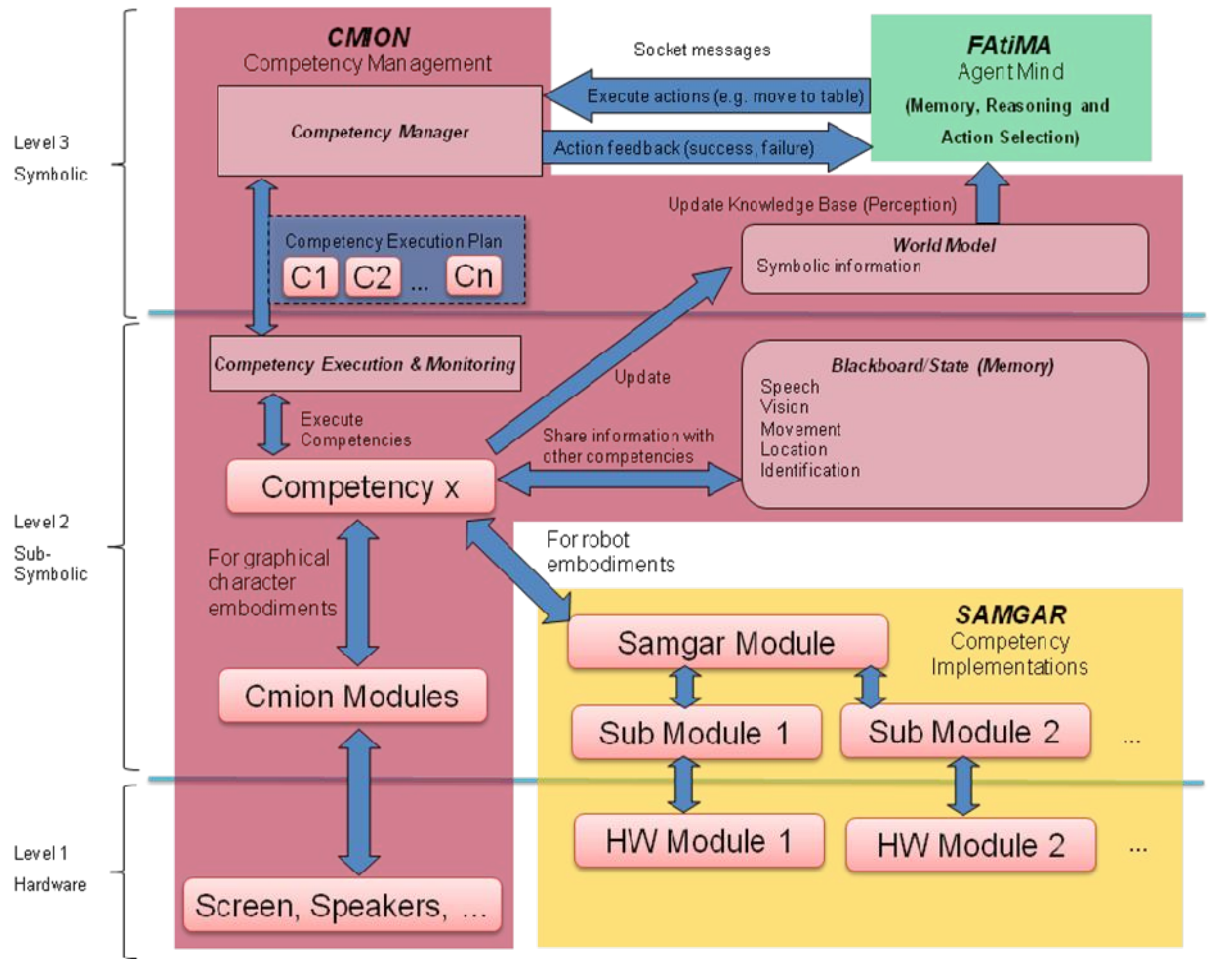


Figure 2: Overview of the different sub systems' place within the three conceptual levels and of the connections between sub systems.

The LIREC software architecture is based on the concept of a three layer architecture [Gat et al.] facilitating software integration. Figure 2 shows how the 3 sub-systems fit within this concept. FAtiMA is exclusively located at level 3 while CMION provides a bridge between levels 3 and 2 (and on platforms where SAMGAR is unavailable even level 1) and SAMGAR bridges levels 2 and 1.

FAtiMA is connected to CMION via a Socket connection. When FAtiMA selects an action for execution it is sent to CMION's competency manager which selects a suitable execution plan that realizes the action on the current embodiment by invoking one or more competencies. Upon completion of this invocation, CMION will send a message to the FAtiMA reporting the success or failure of the action. Instead of being directly invoke by the mind, competencies can also be continuously running. This is usually the case for perception competencies. If such a competency detects information that is relevant for the agent mind it can update the world model component inside cmion, which is directly linked and synchronized with the FAtiMA agent's internal knowledge base.

The connection between CMION and SAMGAR is made on the level of individual competencies. In CMION an individual competency can be linked to a certain SAMGAR module. Whenever that SAMGAR module appears (i.e. is started) the connected competency in CMION will automatically be instantiated and available for invocation. When the competency runs it can communicate with its associated SAMGAR module through a Yarp Port and exchange any kind of data through this port (by using the flexible bottle data structure provided by Yarp).

3.4 Features

3.4.1 Competency Management

Different embodiments will have different ways of performing certain actions. While a companion on a mobile phone can only move in physical space by asking the user to carry it, a robotic companion will be able to move in space autonomously and thus perform this action in a different way. The level 3 content (the symbolic descriptions of actions and goals that FAtiMA uses) should be independent of embodiment. Associating level 3 action symbols to competencies is handled by the Competency Manager inside CMION. It uses a set of rules that are loaded from an xml file when the architecture is started in order to match actions to competencies. Since every embodiment runs its own instance of CMION with its own version of this xml rule file, embodiment specific realization of actions can easily be achieved. The rules for associating actions to competencies offer the following features:

- An action is realized by a pre-assembled competency execution plan, which can contain only a single competency or multiple competencies
- Competencies can be carried out in sequence or in parallel in a competency execution plan
- The order of competencies in the plan is determined by defining dependencies between competencies.
- Parameters can be passed from an action to individual competencies in the competency execution plan

Code snippet 1 shows an exemplary rule that demonstrates the above features. We see an action Talk with two parameters (the first is the person the agent should talk to, the second is the speech act that encodes the meaning of what the agent intents to say). We see that the rule matches this action with an execution plan involving 5 competencies. The competencies without dependency (1 and 3) can be immediately started in parallel, while other competencies will only be started once all the competencies they depend on have successfully completed. This example also illustrates how parameters can be passed from the action to competencies using the “$parameter” syntax.

```
<Rule>
SpeechAct(MeiYii,Welcome)
<MindAction Name="SpeechAct">
        <Parameter No="1" Value="*" />
        <Parameter No="2" Value="*" />
</MindAction>

<CompetencyExecutionPlan>
        <Competency ID="1" Type="DetectPerson">
                <CompetencyParameter Name="PersonID" Value="$parameter1" />
        </Competency>
        <Competency ID="2" Type="MoveToPerson" Dependency="1">
                <CompetencyParameter Name="PersonID" Value="$parameter1" />
        </Competency>
        <Competency ID="3" Type="LanguageGeneration">
                <CompetencyParameter Name="SpeechActID" Value="$parameter2" />
                <CompetencyParameter Name="TextOutName" Value="TextTag" />
        </Competency>
        <Competency ID="4" Type="TextToSpeech" Dependency="3">
                <CompetencyParameter Name="TextInName" Value="TextTag" />
                <CompetencyParameter Name="AudioOutName" Value="AudioTag" />
                </Competency>
        <Competency ID="6" Type="PlayAudio" Dependency="2,4">
                <CompetencyParameter Name="AudioInName" Value="AudioTag" />
        </Competency>
</CompetencyExecutionPlan>

</Rule>
```

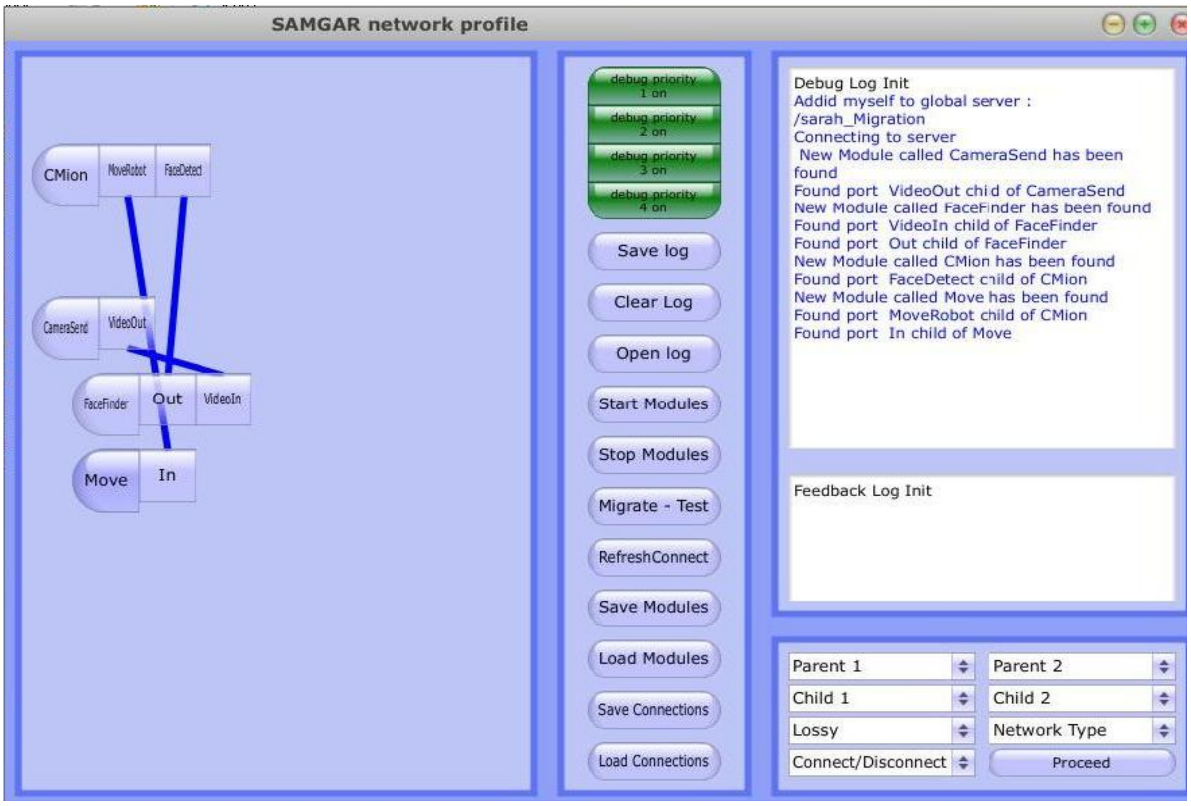
Code Snippet 1: Exemplary Competency Manager Rule

3.4.2 Plug & Play Competencies

As mentioned in 2.1, the LIREC architecture was built with sharing of competencies in mind. It contains several mechanisms to ensure that competencies can be easily swapped, added to or removed from a certain embodiment configuration. In the following we are showing how both SAMGAR and CMION support modularity.

Modularity in SAMGAR

By building on top of YARP, SAMGAR inherits all of YARP's flexibility of modular software development. Functionality is structured in modules, each module being a separate executable that utilises YARP communication ports to send and receive data. SAMGAR adds a graphical user interface to this setup that provides visualization of running modules and that is used to establish connections between modules (see Figure 3). This decoupling of actual module code and the concrete connections between modules facilitates the exchange and reuse of modules. At runtime connections can be established, deleted or replaced. It is even possible to introduce new modules at runtime into a running configuration, without having to restart SAMGAR. For easy re-launching of a complete configuration, SAMGAR also allows saving and loading connection configurations.



```
Debug Log Init
Addid myself to global server :
/sarah_Migration
Connecting to server
 New Module called CameraSend has been
found
Found port  VideoOut chid of CameraSend
New Module called FaceFinder has been found
Found port  VideoIn child of FaceFinder
Found port  Out child of FaceFinder
New Module called CMion has been found
Found port  FaceDetect child of CMion
New Module called Move has been found
Found port  MoveRobot child of CMion
Found port  In child of Move
```

```
Feedback Log Init
```

Figure 3: SAMGAR's Graphical User Interface

Modularity in CMION

In contrast to SAMGAR, CMION is not a distributed system, which means that CMION together with all its competencies is started as a single executable. This is necessary because components inside CMION communicate with each other through the ION framework, which is a communication framework for components within a single application. It follows that additional competencies cannot be added at runtime; CMION has to be restarted to do that. However, modularity is facilitated by reducing the necessity of recompilation. Like the rules for the competency manager (see Section 3.4.1) competencies to load are also defined in an xml file, this one named the competency library (see Code Snippet 2). Changing the competency configuration thus only requires a change to the competency library xml file and a restart of CMION. By making use of Java's dynamic class loading functionality, one can list classes in here that were unknown to CMION at compile time, as long as the compiled class is within CMION's class path at runtime.

```
<Competency ClassName="cmion.TeamBuddy.competencies.Speak"/>
<Competency ClassName="cmion.TeamBuddy.competencies.MoveToUser" />
<SamgarCompetency ClassName="cmion.TeamBuddy.competencies.FaceDetect"
                                Category="FaceDetect" SubCategory="FaceDetect" />
<SamgarCompetency ClassName="cmion.TeamBuddy.competencies.Motion"
                                Category="MoveRobot" SubCategory="Motion" />
```

Code Snippet 2: Excerpt from a CMION competency library file

3.4.3 Inter-Competency Communication

The integration architecture should not only allow hierarchical passing of information as sketched out in Figure 1 but should also allow competencies on the same hierarchy level to communicate with each other. An example could be for example a speech synthesis competency synchronizing with other competencies controlling gestures or facial animations. The main mechanism for supporting this functionality in the LIREC architecture is the blackboard component within CMION. Any CMION competency can write data to the blackboard that any other competency can read. Data on the blackboard is organized hierarchically and identified by a unique identifier. Besides on-demand reading, competencies can also register to be notified to changes of the blackboard or parts of the information hierarchy stored on it. Besides synchronization, another example of blackboard use is the passing of information between competencies that are executed sequentially after each other. The competency execution plan that was presented in Code Snippet 1 makes use of this and passes the output from LanguageGeneration to TextToSpeech and the output from TextToSpeech to PlayAudio. To ensure 2 communicating competencies refer to the same information on the blackboard, the identifier of the information on the blackboard can be passed to the competencies as a parameter as demonstrated in the example above (TextTag, AudioTag).

3.4.4 Migration

Migration means transferring a partial state of the architecture encompassing the state information that makes up a companion’s identity from one running instance of the architecture to another. The architecture supports this process through several mechanisms. Firstly the separation into 3 conceptual layers with varying degrees of abstraction makes it easier to design companions that can function in different embodiments. While the agent’s symbolic knowledge on level 3 can be described in a platform independent fashion, the competencies that are available on level 2 and the lower level modules of level 1 can differ on every platform. Different competency manager rule configurations (see Section 3.4.1) allow embodiment-specific mappings of the same level 3 actions to different execution plans. At the same time the modular structuring of architecture components on level 2 (CMION) and 1 (SAMGAR) makes it easy to share competencies between embodiments where applicable. Secondly communication protocols for carrying out a migration from one embodiment to another one (at a predefined non-dynamic location) are integrated inside a CMION migration competency that is part of the core CMION package. This migration competency defines an xml format for describing the migration data and provides interfaces for any competency and the agent mind to embed their state in the xml migration data. On the receiving architecture instance the migration competency parses the incoming xml and distributes the different parts of the migrated state to the relevant competencies or the agent mind. Figure 4 illustrates this process. The CMION migration competency also handles the number of agents inside an embodiment (typically only one is allowed).

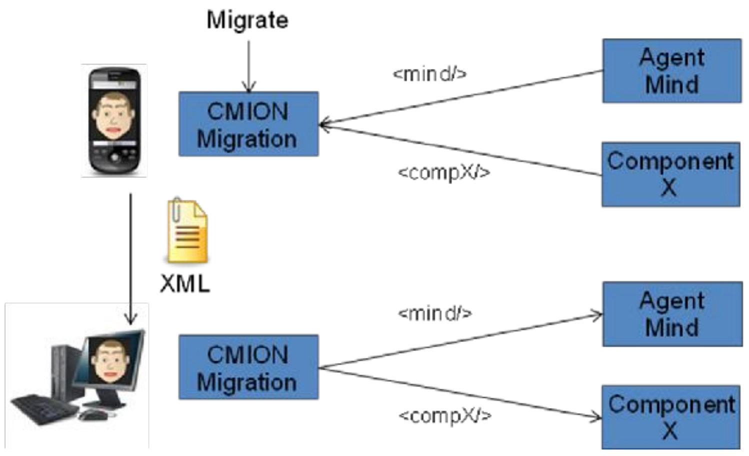


Figure 4: CMION migration competency

SAMGAR also includes support for migration. In particular it allows the dynamic addition and removal of migration platforms through a global migration server. So far the dynamic migration targets provided by SAMGAR in this way have not yet been linked with the CMION migration competency but plans in improving the migration competency in that direction are underway. More information on migration can be found in WP8 documents.

3.4.5 Monitoring and Debugging

Both CMION and SAMGAR include a number of monitoring tools that simplify debugging of complex companion configurations. The simplest of those are built-in logging mechanisms that can be filtered (in SAMGAR according to priority levels and in CMION according to source, i.e. the competency/component that caused the event). SAMGAR also provides a visual feedback that automatically highlights missing/crashed modules and broken connections. Also the modular plug and play approach taken by SAMGAR simplifies debugging by allowing dynamically rerouting inputs and outputs (e.g. routing the output of modules that are error suspects to a visualisation module in addition to its normal destination) and replacing modules with different simplified versions to rule out errors (e.g. modules that perform certain tasks autonomously could be replaced with manually controlled versions). By building its communication on top of the request and event based messaging features provided by the ION framework, CMION is also suited for the insertion of monitoring tools. Some general purpose monitoring tools are already included with CMION, e.g. for inspecting and changing the world model, while more specialised ones can be easily built for the needs of a certain scenario / embodiment (e.g. we have built a WoZ tool in CMION for remote controlling the Embodied Conversational Agent Greta [Niewiadomski, Bevacqua, Mancini and Pelachaud]). There is however a crucial difference between CMION's and SAMGAR's monitoring capabilities, which is inherited from the respective middleware the systems are build upon (ION, YARP).

In general SAMGAR is more suited for dynamic reconfiguration and thus monitoring methods can be applied at runtime, while in CMION the monitoring modules have to be included when the architecture is started. On the other hand monitoring tools in CMION do not require any manual connection due to the event based message handling. Table 1 summarizes this trade-off.

	SAMGAR	CMION
When to insert monitor	During runtime	Before starting up
Manual connection necessary	Yes	No

Table 1: comparison of SAMGAR and CMION monitoring capabilities

3.4.6 Mobile Platform Compatibility

With the inclusion of mobile companions in LIREC, we have to ensure that the LIREC architecture is also compatible with mobile devices. Due to its flexibility, the broad range of supported devices and the Java based SDK, we decided for Android as the mobile operating system in LIREC scenarios. While SAMGAR cannot be run on mobile platforms, CMION and FAtiMA are both written in Java, which made porting them to Android relatively straight-forward. They are now both fully operable on Android devices. One of the main differences between Android applications and Java applications run on the desktop is the application entry point / launcher. Android Java applications are not started through a main method for example. Thus, Android launchers for FAtiMA and CMION have been created and for both projects care has been taken to structure the code modularly, so that core code is separate form the launchers. Also further platform specific code (e.g. different File I/O libraries) has been identified and modularized. This ensures that both desktop and mobile versions can work with the same codebase, meaning both version will not diverge and equally benefit from improvements and bug fixes.

4 Current and Future Developments

As mentioned before in Section 3 we adopt an incremental feedback-driven development approach. At this point in time where partners have started applying the architecture to their scenarios, several improvement suggestions have been pointed out. Below are some of the major current developments for SAMGAR and CMION. We will not discuss future plans for FAtiMA here, as this topic is more related to WP5.

4.1 SAMGAR v2

Samgar has been updated into a more refined form, following experiences in practical use, and facilitated by an update in a core part of Samgar, YARP. YARP now allows topics such as are found in ROS (Robot operating system), and a more beneficial update for use with Samgar, persistent connections. This allows the re-initialisation of connections and disconnections to work at a lower level, allowing an increase in efficiency and a higher level of reliability, a core benefit of which is an interrupt driven system.
Reliability and ease of use and understanding have been highlighted in the refinement, and to achieve these a new graphics library is used (QT) which has allowed facilitation of drag and drop connections, an improvement over button control, and the workspace allocation method. The workspace allocation method is now not computer generated but user generated, allowing the user greater control over how to display modules, allowing users to create data flow diagrams which allow easier understanding. These personal preferences are can be saved and loaded allowing much greater flexibility, increasing the efficiency of systems and collaboration between colleagues.
Refinements have also been produced within the lower level module design, allowing specific interrupt driven ports, and the copying of classes over a network, allowing more elegant and efficient designs with less Samgar specific overhead. There already exist a number of new modules for the refined Samgar and it allows much more flexibility in transferring modules to the new system as it is an efficient and easy process.

4.2 CMION improvements

We are currently planning the inclusion of the following additional features to CMION to create a more flexible and powerful architecture:

- **Resource management:** Currently the architecture does not include any mechanism for resolving conflicts between competencies that require the use of the same resources. Currently the designer of execution plans for the competency manager (see 3.4.1) has to take care to design the plans so that no conflicting competencies are executed at the same time. We plan to provide mechanisms to make those resource conflicts more explicit and automate their resolution. The basic idea involves the option to tag certain competencies with resource identifiers and the runtime ability of the system to schedule competency execution based on this information so that no resource conflicts occur.

- **Stronger integration with FAtiMA:** We also noticed that a tighter integration with FAtiMA would be desirable. In the downward communication direction (level 3 to 2) we plan to support the ability for FAtiMA to abort currently executing actions. On the upwards direction a closer integration with FAtiMA's motivational system seems desirable so that the levels of certain physiological needs (e.g. Energy) that are currently modelled in FAtiMA can be set by the lower levels of the architecture to better reflect the real physiological needs of the systems (e.g. battery level of a robot).

- **Advanced migration features:** In conjunction with WP8 we aim to improve the migration competency to allow for advanced features such as authorization, dynamic migration targets (linking with SAMGAR's migration features) and automatic backups

5 Application in the LIREC Scenarios

In this section we present the first efforts of the technical partners within the consortium of applying the common integration architecture. This is crucial not only for demonstrating the usability of our produced architecture in a variety of contexts but also for showing technical integration within the project. Below the application of the architecture in various LIREC companion scenarios is detailed. As in Deliverable 9.3 the descriptions below are complemented by online videos on the LIREC website. The videos corresponding to this deliverable can be found at:

http://www.lirec.eu/integrated-companions-progress-fall-2010

5.1 Robot House (University of Hertfordshire)

The Robot House video on the above website demonstrates the updated integration architecture of the LIREC companion for the UH (University of Hertfordshire) Robot House Showcase. Various robot behaviours (migration, expressions of internal states e.g. “happiness”, “excitement”, etc.) and functions (food selection etc.) have been added, as well as improving the speed of the Companion’s migration process. These advancements are demonstrated in the video in three scenarios based on typical daily activities (i.e. medicine reminder, directing user attention to a courier at the door, assisting users on food selection etc.), which forms part of the UH Robot House showcase. Various capabilities of the Companion robot are illustrated:

- FAtiMA is utilised to determine the likely location of the user, based on their interaction history (i.e. to remind the user of his medicine).
- Utilising physical movements, sound and display lights to express the companion’s internal states (i.e. “happy”, “sad”, “excited”, “bored”, etc.) and attracting the user’s attention.
- A faster and improved migrating process is also demonstrated (where the companion sequentially can inhabit different embodiments, currently pre-specified as embodiments being most suitable to a particular task).
- Ability to store and migrate temporary data related to a current task.

5.2 Spirit of the Building (Heriot-Watt University)

The complete integration architecture has been applied in the Spirit of the Building scenario developed at Heriot-Watt University. The “In The Wild” and “Team Buddy“ embodiments utilise all 3 architecture sub-systems (FAtiMA, CMION and SAMGAR), while the “Personal Guide” as a handheld device (see Section 3.4.6) only uses 2 (FAtiMA and CMION). The video for this scenario demonstrates migration across those 3 platforms. The Heriot-Watt team has also provided a short tutorial / integration example based on the Team Buddy showcase for explaining and demonstrating the use of the architecture. Current work on the scenario includes the integration of more competencies and the configuration of the FAtiMA planning domain for more complex autonomous behaviour.

5.3 FLASH (Wroclaw University of Technology)

WRUT applied the three-layer architecture for LIREC companions on its hardware. Due to the fact that WRUT use 64 bits CPU architecture system, a straightforward application of software caused some difficulties. To overcome this problem a set of modifications was performed within SAMGAR. The most significant change is a GUI library change from JUICE to Qt (see 4.1). Further work resulted in the implementation of CMION and FATIMA in WRUT test systems. All components work both on 32 bits and 64 bits Ubuntu 10.04 LTS that is a target WRUT Linux system for the rest of LIREC project. An initial integration of the three-layer architecture was performed and the Team Buddy example (see 5.2) was implemented on WRUT test hardware. After small modifications Team Buddy was successfully implemented on FLASH.

5.4 MyFriend (INESC-ID)

MyFriend Showcase consists of three different scenarios: The Game Companion, The Multi-player Game Companion and the Pleo Scenario. The Game Companion scenario was one of the first working prototypes in the Lirec project that was used for studies with users. As it was implemented in the beginning of the project, this scenario is not using the architecture subsystems (FAtiMA, CMION or SAMGAR), as most of them were not complete by that time. However, some of the conceptual ideas implemented in this scenario were then reused in the implementation of the architecture (for example, the migration competence).

The Multi-Player Game Companion is in a very early stage of development. There are two different possibilities in terms of embodiment for this companion, either the iCat or FLASH’s head. Especially in case of the second option, the scenario will benefit from using some of the architectural components. We also have plans to integrate some of the components of the FAtiMA architecture in the agent’s decision process (level 3).

A prototype of the Pleo scenario in a handheld device was developed and has been tested by owners of the Pleo robot. As in the Game Companion scenario, this prototype runs on Android. At the moment, efforts are being made in order to analyse the possibilities of using a lighter version of FAtiMA in Pleo’s mind.

5.5 Germination X (FoAM)

Germination X is an online experiment started by FoAM in the second half of the project as a strategy to demonstrate exploitation and dissemination of Lirec technology developed so far. This experiment demonstrates the use of the

AgentMind in an online social gaming situation. Such platforms are expected to scale to the order of 10,000's of players. We will investigate the feasibility of this in order to prove the applicability of the software in new areas.
The experiment will additionally provide effective dissemination for Lirec by allowing many members of the public to interact with companions first hand, via a web browser. We also hope to provide a potential source of long term interaction data and an investigation into the use of social networking information in a socially aware agent architecture (and ethics thereof).
The initial phase of this project has involved embedding part of level 3 (FAtiMA) in a web server to allow browser based interaction with multiple Lirec companions. A prototype of this configuration has been completed, allowing online users access to FAtiMA agents running in a web application.

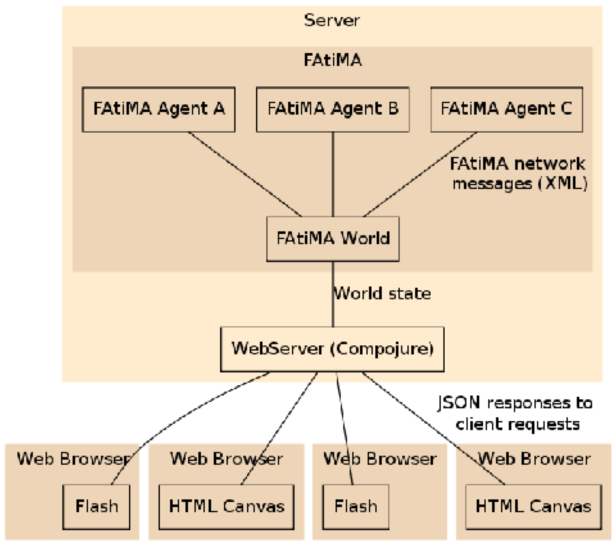


Figure 5: Germination X components overview

The webserver converts FAtiMA events and state into JSON which is dispatched to multiple clients, so many people can interact with the same companions remotely.

6 Installation Instructions

The source code for the full architecture is freely available from the LIREC subversion server. For download links, instructions and documentation please visit the following website:

http://trac.lirec.org/wiki/SoftwareReleases

7 References

M. Vala, G. Raimundo, P. Sequeira, P. Cuba, R. Prada, C. Martinho, A. Paiva, *ION Framework --- A Simulation Environment for Worlds with Virtual Agents*, Proceedings of the 9th International Conference on Intelligent Virtual Agents, 2009, pages 418-424

P. Fitzpatrick, G. Metta, L. Natale, *Towards Long-Lived Robot Genes,* Robotics and Autonomous Systems, January 2008,Volume 56, Issue 1, Pages 29-45.

E. Gat, *On Three-Layer Architectures*, Artificial Intelligence and Mobile Robots, 1998

R. Niewiadomski, E. Bevacqua, M. Mancini, C. Pelachaud, *Greta: an interactive expressive ECA system*, Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, 2008

K. Du Casse, K. L. Koay, W. C. Ho, K. Dautenhahn, *Reducing the cost of robotics software: SAMGAR, a generic modular robotic software communication architecture*, Int. Conf. on Advanced Robotics 2009, pages 1-6