VLE architecture and design: Report on design and specifications for the tutors' architecture

Work package	VLE architecture and design (WP6)	
Task	Architecture specification (T6.1) Architecture Design (T6.2)	
Dissemination Level	Public	
Publishing date	Contractual M12	Actual
Deliverable	6.1	
WP / Task responsible	Ruth Aylett – HWU	
Contact person	Ginevra Castellano, UoB	
Contributors	HWU, UOB, INESC-ID, Ydreams	
Short abstract	This document proposes specification of a generic architecture that is required to support the showcases in EMOTE project.	
Keywords	Architecture, scenarios, robot	
Documents	xxxx	

This work was partially supported by the European Commission (EC) and was funded by the EU FP7 ICT-317923 project EMOTE (EMbOdied-perceptive Tutors for Empathy-based learning). The authors are solely responsible for the content of this publication. It does not represent the opinion of the EC, and the EC is not responsible for any use that might be made of data appearing therein.

Table of Contents

Introduction 1

Hardware 1

Touch Table 1

Robot 2

Sensors 3

Requirements Analysis 4

Showcases 5

Showcase 1: Map Activity (Treasure hunt) 5

Scenario 2: Game Activity (Enercities) 5

Architecture 6

Architecture Components: 6

Thalamus: 6

The Thalamus Character 7

Behaviour Markup Language (BML): 9

Dialogue Manager (DM): 10

Perception Capabilities/Competencies: 12

Source code Repository: 13

Conclusion 14

Future Work 15

References: 15

Appendix I - 16

Introduction

This document proposes and specifies a generic architecture that is required to support the showcases in the EMOTE project. Developing the showcases requires the integration of hardware (multi-touch table and robotic tutor). The architecture also includes development of interaction modalities, both for users and the robotic tutor, and the integration of software that can process user affective state and implement the decision-making capabilities of the robotic tutor. This document describes the approach being taken to address the key issues with respect to the design of the tutor architecture. Also issues with the integration of the theoretical and pedagogical work of WPs 2 and 3 with the technology developed in WPs 4 and 5 to provide two showcase VLEs in specific geography-related domains. The consortium will develop two showcases in the area of geography, one focusing on map reading skills and the other on energy-related environmental issues.

Hardware

The main hardware required for the scenarios in EMOTE, is a multi-touch touch table, a robot and sensors capable of tracking the user.

Touch Table

After an extensive survey of touch tables evaluated against the requirements of the project and with an agreement of project partners, a touch cell from Mutlitaction was acquired by 4 partners, namely: Heriot-Watt, UOB, INESC-ID and UGOT. Refer figure 1A: left, the touch cell; right, an initial wooden table design recommended by INESC-ID (Figure 1A: right) to setup the touch cell in horizontal position for the learner to interact with the touch cell and the robot.

Figure 1A Left: MultiTaction Cell 55" Full HD LCD with Embedded PC with Windows 8, full specs can be accessed at [Touch table specs, Touch table video], right: table design

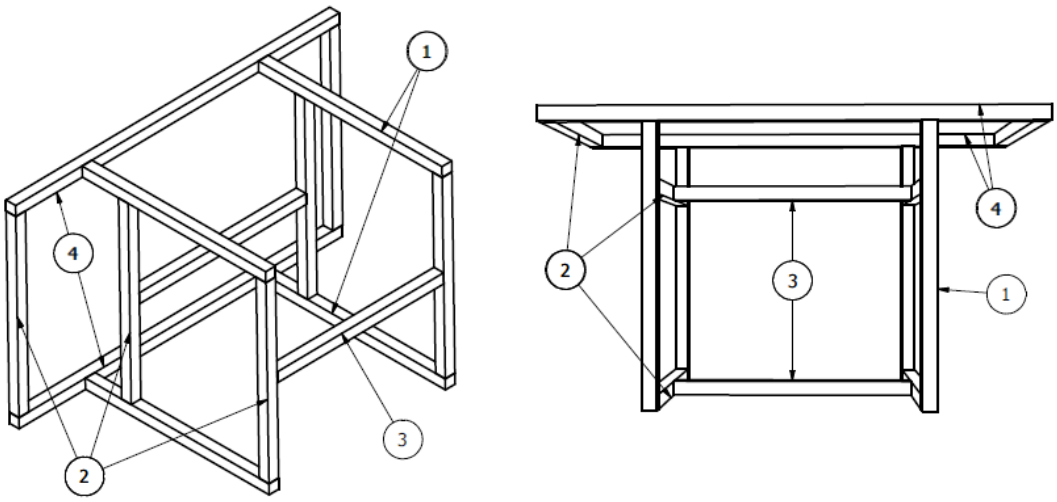

Figure 1B : Alternative table design by UOB

At this stage, the partners are in discussion about the final requirements for the table setup considering the position of the robot (tutor) in regards to the learner (Front/Lateral). An alternative table design suggested by UOB is also being conisdered at the moment using an aluminium profile (Figure 1B). This design is lightweight, strong and would be far more resilient to the regular assembly and disassembly required when visiting schools etc. The problem with building wood furniture is, the more you move it and unscrew the screws and bolts, the weaker it becomes. Following the mock-up study performed by UGOT with child participants and a real teacher, the position of the robot on the table and the position of sensors will be finalised.

Robot

The consortium decided to use the Aldebaran NAO robot torso as the robotic tutor platform shown in Figure 2. The robot has 14 degrees of freedom, consisting of two 5-degrees-of-freedom arms with 2 hands and a head with 2-degrees-of-freedom for pan and tilt. The robot head is equipped with multi-colour LED lights, two speakers in its ears for producing synthesised sound through text-to-speech, and a camera that can capture 30 images/second. The NAO torso has an embedded computer with WIFI and LAN connectivity to touch table accessible via the provided API [Aldeberan SDK]. EMOTE will be mainly using the robot speech, LED lights in the eyes, head and hand movements for generating behaviours.

Figure 2: Robot Nao Torso

Sensors

Based on the project's requirements for automatic user engagement, UOB decided to use a setup that consists of a video camera for capturing human-robot and human-task interaction, a Microsoft Kinect sensor for predicting body posture, position, real-time face recognition software for extracting facial expressions, head direction and eye gaze and finally a Q Sensor for measuring affective states via electro-dermal readings (Figure 3). More specifically, a web camera will be placed in two different positions around the touch table depending on the current setup (Setup 1 or Setup 2) in order to capture facial features for automatic engagement such as smiles so as to infer various emotional states (happiness, sadness, fear etc.), and eye gaze information.

In this early phase of the project, we will test and evaluate the optimal robot position that delivers the most engaging experience to the user along with the best performance in terms of user-robot and user-task interaction. Additionally, various facial expression detection SDKs (e.g. OKAO[1]) will be tested in order to evaluate the performance and accuracy on this specific task with users gazing towards the table or the robot. The Kinect sensor will be placed opposite the user in order to primarily measure the lean angle towards the table and secondary to track the head orientation of the user along with some basic facial action units. Lastly, a Q Sensor will be used to wirelessly deliver the electro-dermal activity of the user while interacting with the task in order to acquire the arousal of the user.

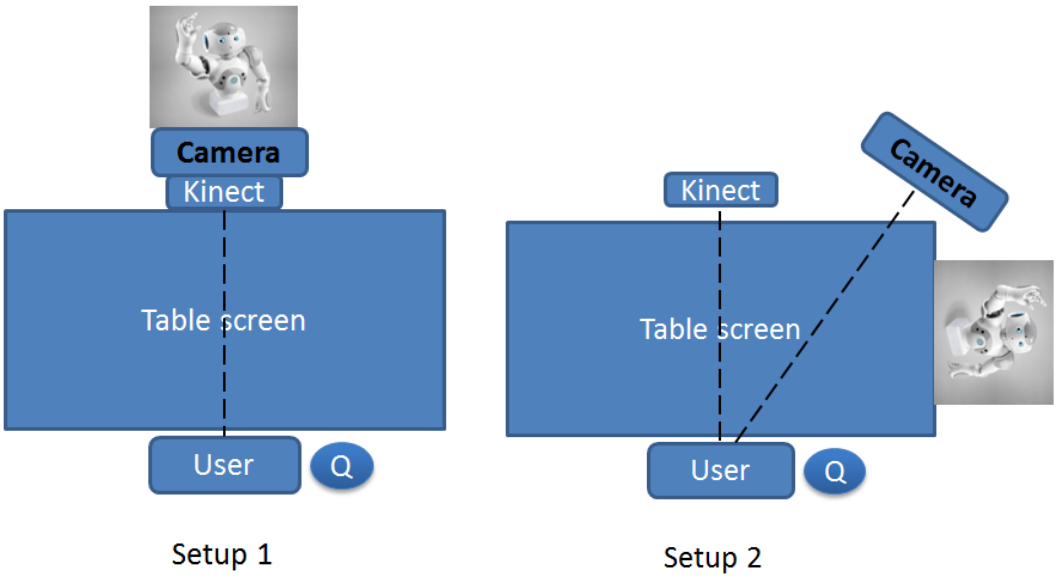

Figure 3: Sensor setup on the table

YDreams focus on the experiments around novel design paradigms for multi-touch tables that support the learning goals as defined in the learning scenarios. The recent version of Kinect for Windows SDK 1.8 and its capabilities are been researched and tested for EMOTE. Considering the predefined set, with the multi-touch table and the Kinect, several experiments for seamless and composed interaction fusing both devices have been sketched and will be

[1] OMRON OKAO SDK, http://www.omron.com/r_d/coretech/vision/okao.html

assessed. A strong focus has been applied to hand detection with the Kinect sensor. Some examples are:

- Hand detection above the multi-touch table: simulation of grabbing and releasing objects (instead of using other less natural types of menu based grabbing simulations usually used on multi-touch panels);
- Cloud or other aerial direct interactive elements (rain effect according to cloud/hand distance to the table – the user hand is the cloud). Other possible applications in the scenarios will be studied).

The goal of this task is to seamlessly extend the interaction space from the touching surface to the area above it, so that new interaction design paradigms can be created for this setup. During the next year, YDreams, together with the EMOTE consortium, intends to further explore these options and integrate the interaction features in the general project architecture.

Requirements Analysis

Given the above requirements for hardware required to develop EMOTE scenarios, we investigated the use of existing architectures, including that from the FP7 LIREC (www.lirec.eu) project (based on the INESC-ID ION framework). However the EMOTE project does not involve mobile robots, and the functionality required from the architecture point of view is mainly to establish communication between the sensors, robot, touch table and modules developed with WPs 2-5. We concluded that EMOTE architecture should be functionally distributed in design in contrast to the standard 3-layer architecture platforms commonly used in robotics. The proposed architecture (described in more detail in Architecture section in this document) will be using Thalamus as middleware which provides a suitable functionality required for EMOTE scenarios as well as supports migration from robot to handheld devices. The architecture also has to meet the requirements of the two showcases being developed in the project (more details on showcases can be found in D2.1).

Showcases

Showcase 1: Map Activity (Treasure hunt)

This scenario will be based on a map activity implemented as a treasure hunt task. During the interaction, the robot tutor will provide clues/hints to the learner in order to navigate through various locations on the map. The learner's map reading skills will be tested through this activity.
HWU developed an initial prototype to study the setup requirements to run the map reading scenario using Google maps; this was primarily to test the requirements for scenario 1. This prototype had a map application installed on the touch table and simple dialogue manager providing clues to the user for solving the treasure hunt on the map. (video link [Emote Prototype 2013])

Map application
The University of Birmingham is currently working on map application development. Based on the review of the curriculum for the age group of students, review of existing interactive map activities and consultation with teachers a map application has been designed. This is documented in detail in deliverable D2.1

In this task, the user interacts with the map via a web application that displays a map using OpenlayersToolkit (an open source javascript library for displaying and interacting with map data, http://openlayers.org/) and, additional controls are added using html and javascript. The web application backend is written in java, which handles communication to the rest of the system via a thalamus bridge. Map data for the activity will be stored and served locally using GeoServer (also open source), this means that we do not require an internet connection to run the application.

Scenario 2: Game Activity (Enercities)

In order to vary the range of showcase types, it was decided to consider a second showcase in game-based learning. The Enercities game (http://www.enercities.eu/) was developed in the European Commission programme Intelligent Energy Europe (programme area: Intelligent energy education initiative) in the period 2009-2011 and won the 2010 title of "Best Learning Game 2010" in the European platform "ENGAGE Quality Awards". The code has been licenced from Paladin Studios, the Dutch game company that now distributes Enercities.

This single-user game has been used as the basis for an EMOTE multi-player collaborative learning game, in which the robot acts both as a tutor and a player, while collaborating with two students on the development of a virtual city with focus on keeping environmental issues in mind. The game is developed in Unity3D/C#, and runs as a full-screen application on the multi-touch table. It communicates with the rest of the system using a Thalamus Client in C# (Thalamus middleware is described in more detail under Architecture->Thalamus section). As a Thalamus Client, it will publish messages regarding game events, and also some specific and well-defined user input, such as answers to yes/no questions. It will subscribe to game actions including multi-touch navigation such as panning and zooming, in order for the tutor to be able to share the same kind of interaction with the table as the students, but without physically touching it. This scenario is documented in detail in deliverable D2.1.

Architecture

The architecture design will take an iterative prototyping approach linking closely with the user-centred design work of WP2. In the first year, we established initial requirements, will refine these through iterative prototyping and develop an architecture design that can meet the requirements. The proposed architecture in figure 4 gives a conceptual overview of the various functional components currently being developed to meet the requirements of the artificial tutor and scenarios discussed in the previous section. The architecture is a generic one and will support both scenarios at the abstract design level, although the actual implementation will differ for the two showcases given their functional differences.

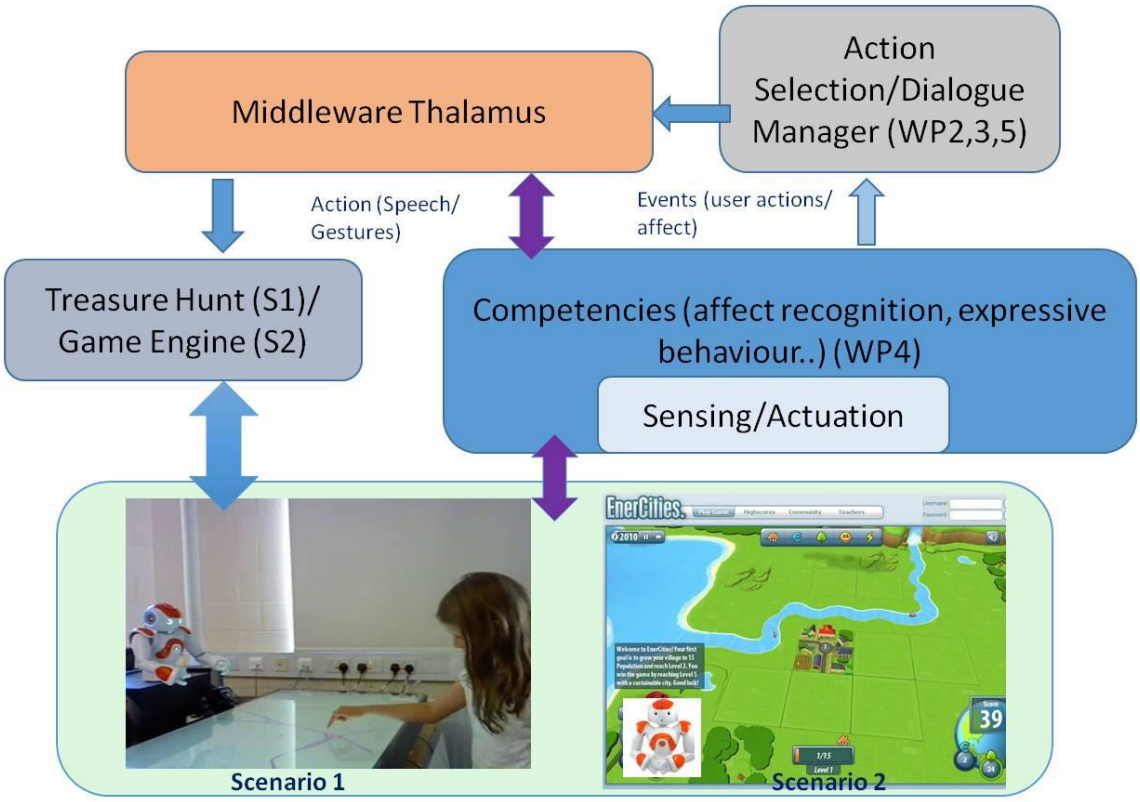

Figure 4: Architecture Design

Architecture Components:

This section describes each component in the architecture and how they can interact with each other. The learners' touch inputs, map/game state can be communicated from the touch table through middleware to action selection module which can respond with corresponding action to produce a robot action or map/game action. The sensing competencies can also communicate the learners' affect to action selection module in the same way via middleware.

Thalamus:

Thalamus will act as an interface/middleware connecting all the architecture components together and allowing them to communicate with each other. Thalamus is a framework developed in C# for creating body-independent modular interactive characters [Ribeiro, Vala & Paiva, 2012]. It was initially developed to provide a flexible framework support multi-modal

behaviour in robotic characters with feedback from the real physical environment. It has since then evolved into making it possible to create interactive characters out of a set of different independent modules, each of which has a designated role in the behaviour of the character. Unlike most the other interactive character frameworks, there is no specific module to act as the Mind, neither as the Body. It is based on the Censys framework for interactive embodied characters [Ribeiro, Vala & Paiva, 2013a].

Instead, there can be, for example, a module that deals with text-to-speech, another dealing with recognition of humans using a Microsoft® Kinect®, another acting as a Dialogue Manager, and still other modules controlling the different parts of the character's body (virtual or robotic) or sensors (camera, speech recognition, etc.).

The main advantage of using Thalamus is that by creating these modules once, we can later re-use them with different characters in different scenarios with similar requirements. It also makes it easier to share modules with other colleagues so that all our work can be useful to other people even if they did not use the exact same character, or the exact same scenario, dialogues and interactive goals.

The Thalamus Character

A Character in Thalamus is a runtime environment in which Thalamus Modules (or *Clients*) reside and interact. The collection of Modules that compose the Character defines how the Character behaves. We can think of a Character as a closed environment, as all the messages that travel between Modules are internal to the Character.

However, a Thalamus Character is made to interact with an external environment. In order to achieve this, some of the Modules should have some means of communicating with the external environment, and translating all the information into a Thalamus message (*Fig. 5*).

Although we think of a Thalamus Character as being just an abstract, open-space, in fact, each Character contains a master node, which we call a Thalamus Master. This node will be in standby for Modules to connect with it. By default, whenever a Module is created, it automatically searches for a Master in its own sub-network (via an UDP Broadcasting mechanism) and connects to it (*Fig. 6*).

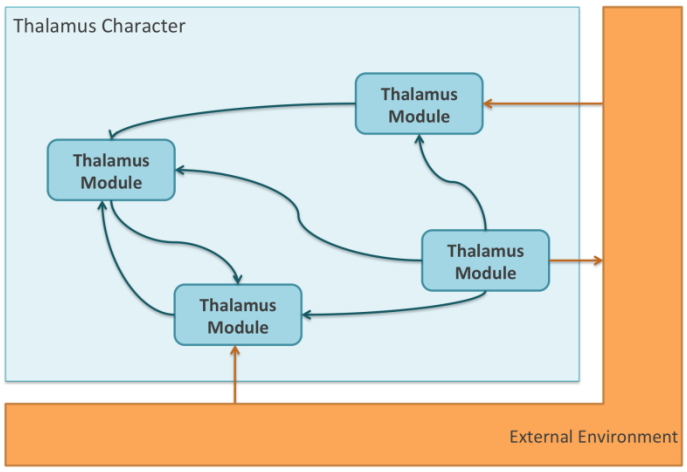

Figure 5. Some Modules receive or send information from the external environment (orange arrows) and then transmit them to other Modules as Thalamus Messages (blue arrows).

All the messages that travel between **Modules** thus actually go through the **Master**. That is important in order to guarantee, in the future, Message/Parameter remapping, or conflict resolution.

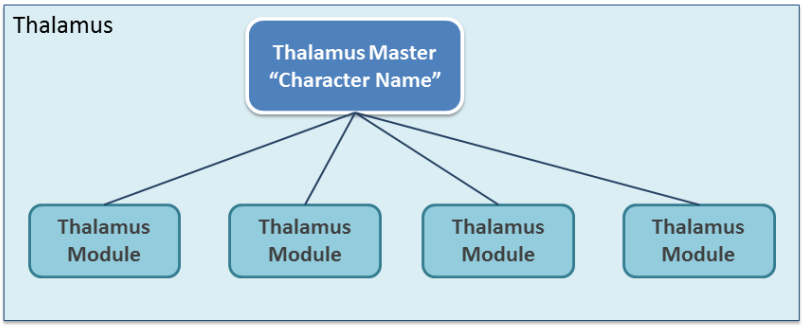

Figure 6. Thalamus Modules automatically search for and connect to the Thalamus Master (which represents the Character). This connection process is completely transparent to the Module programmer.

Thalamus Modules (Clients)

A Module in Thalamus is a program or a piece of a program that we can think of having Sensors and Effectors for Thalamus Messages. Technically it follows a Publication/Subscription mechanism, so a Sensor actually corresponds to the subscription (and reception) of a set of Messages, while an Effector corresponds to the announcement (and publication) of a set of Messages. It is important to distinguish these four concepts:

Announcement When the *Module* connects to the *Master*, it announces the *Messages* (both *Perceptions* and *Actions*) that it will be later publishing.

Subscription When the *Module* connects, it subscribes to certain *Messages* that it should later receive.

Publication At any point of its execution, the *Module* can publish any of the *Messages* it announced, so that other *Modules* who subscribed to the same *Message* will receive it.

Reception Whenever another *Module* publishes a *Message* we subscribed to, we will receive it.

A Thalamus Message can be a Perception or an Action. Perceptions are implemented as a notification of occurrence of an event, and an Action as a request for a module to do something (*Fig. 7*).

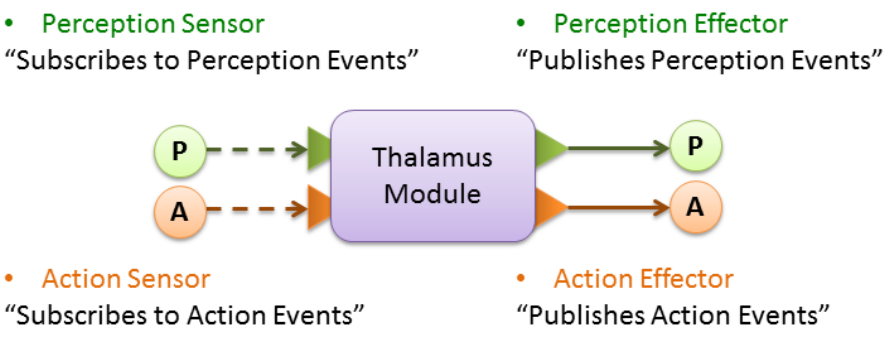

Figure 7. A Module can both subscribe to Perceptions and Actions, and publish Perceptions and Actions.

Both types of messages are defined by a message Type and a set of named parameters. The message Type does not directly state whether the message is a Perception or an Action. Instead, it discriminates via the conceptual and structural type of data it contains. As an example, a message that acts as a perception representing whether the tutor currently sees anyone, could have a Type *PersonInSight* and a True/False parameter. Thalamus also allows messages be sent to the Robot in BML (Behaviour Markup Language) format which gives more flexibility to the Decision/Action selection module. The use of BML will make the architecture more scalable to other robot platforms as well as virtual agents on mobile devices which will be addressed in year 2/3 migration aspect of the project since it defines a hardware-independent interface to body-level commands.

Behaviour Markup Language (BML):

Our work builds on the SAIBA framework [Kopp et al., 2006], shown in **Figure BML1**. SAIBA is a representational framework for unified multimodal behaviour generation. One of the cores of SAIBA is the Behaviour Modelling Language (BML).

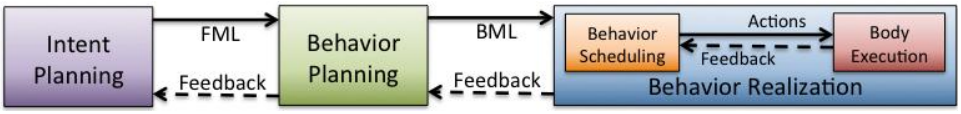

Figure BML1: The SAIBA framework.

In the figure we can see an Intention planning phase which produces Functional Markup Language (FML). This information is a high-level description of the behaviour that the character is intended to do, without specifying yet how exactly it should be performed in detail. A Behaviour Planner receives this FML and then generates BML which is more specific, containing discrete animation names, and specific timing values for the intended behaviour to be realized. The last phase of SAIBA is to send the BML to a Behaviour Realizer, which will actually perform the behaviour on the characters.

In EMOTE, we expect to have different modules being able to produce BML code that is then scheduled by Thalamus and executed on the robotic tutor by using some realizer that supports the robot. In order to switch to a virtual version of the robot for migration, we would just either need to switch the realizer from a robotic one to a virtual one, or use a realizer that would already support both types of embodiments.

Thalamus is, in its core, largely based on the way BML works, so it is important to understand the core mechanisms of its functionality. First, a BML block can be composed of several BML behaviours as seen in **Figure BML2**. Note that this figure does not represent the whole nor accurate specification of a block, but is instead used only as a reference to what a BML block looks like.

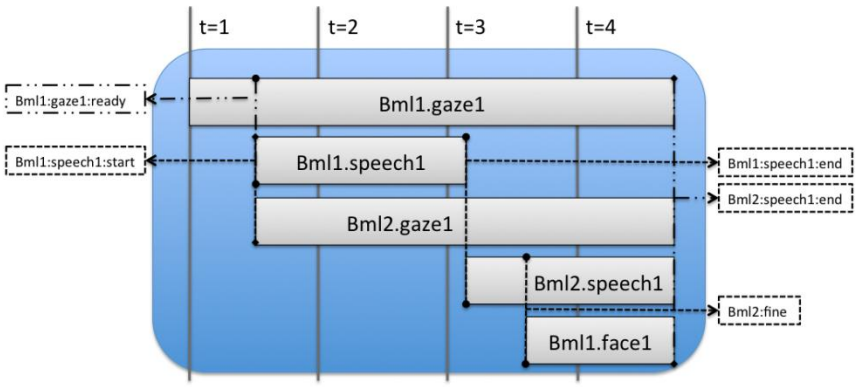

Figure BML2: An example BML block containing several synchronized actions.

Behaviour actions can be of type Face, Gaze, Gesture, Head, Locomotion, Posture or Speech. Each behaviour is also defined by a unique ID, and synchronisation points (SyncPoints). These SyncPoints are what makes it possible for a character to perform synchronized multimodal behaviour, and also to synchronize its behaviour with other characters. All behaviours have a start and end SyncPoint, and some may also have other intermediate SyncPoints, to define when a certain instant of the action is happening. Please refer to Appendix I for a detailed example.

We currently run all the actions directly using NAO's own API. However, in the future our plan is to try to integrate a more flexible animation engine that is capable of dealing with asynchronous events. The partners from INESC-ID are developing Nutty Tracks [Ribeiro, Dooley & Paiva, 2013b] which is an animation engine that supports mixing pre-designed and procedural animation, all driven by asynchronous events, and is also body-independent, so the same animation program can be used with different robots, in both physical and virtual forms (see D5.1). We expect that the BML functionality of Thalamus combined with the features of Nutty Tracks will further aid the architecture to be extended across handheld devices and thereby support migration from a robot to a hand-held device; this will be addressed in more detail in year 2 of the project.

Dialogue Manager (DM):

Dialogue management is the task of managing the conversation between the system and its users, deciding what to say when. This task is carried out by the dialogue manager (DM) module. In this project, we aim to design and implement a dialogue manager that will manage pedagogical conversations between the system (playing the role of a tutor) and one or more users. We plan to do this using two showcases: the Map-based Treasure Hunt and Enercities. The dialogue manager will take inputs from other modules, such as the application module (Treasure Hunt application/Enercities game), the learner model module, and the affect recognition module. Through the Behaviour Generation module, the embodied robot will present its output gestures and utterances including deictic references to the application module (i.e. the touch table).

DM State

The DM state defines the current context. Based on this, the DM decides which dialogue moves to make and when to make them. This includes the following information:

a. Task state (Enercities game/Treasure Hunt) -
For the Treasure Hunt, this includes information such as: How much of the task has been completed? What is the current sub-task? Is the user's answer/response correct, incorrect or partially correct? For the Enercities game, this will include the state of the game, environment, economy and citizen scores, etc.

b. Affective state of the user(s) - This state informs the system about the emotional state of the user(s): i.e. happy, sad, confused, etc.

c. Pedagogical state of the user(s) - This state informs the system what skills/concepts the user(s) might already know/have already learned.

d. Engagement level of the user(s) - This state informs the system the user(s) level of engagement.

e. State of the dialogue - This state contains information about the conversation in general like current speaker, turn holder, current addressee(s), etc.

DM Actions

The dialogue manager is a decision making component which decides what the system should do next in response to the dialogue context update. We divide the set of decisions the DM should be capable of into two sets which are described in further detail in D5.1:

Task moves: Task moves are actions related to the underlying tasks. If the system is engaged in the underlying task, it will have to generate task moves. In scenario 1, the system does not participate in the task (i.e. Treasure Hunt activity) but plays the role of an observer and a tutor. But, in scenario 2, the system plays two roles: player and tutor. As a player, it will generate task moves which are equivalent to a user's game moves such as building a new structure or upgrading an existing structure, etc.

Dialogue moves: In response to the dialogue context, the system communicates with the user(s) using dialogue moves. Dialogue moves are processes that create dialogue actions. Dialogue actions (DA) abstractly represent the content and form of what a dialogue participant wants to present. Dialogue actions include *pedagogical actions* such as questioning, prompting, hinting, *affect management actions* such as encouraging the user, affective feedback to users moves, task actions like explaining current game move, *time management actions* such as stalling, *turn management actions* such as requesting a user to play his/her turn, and *communication management actions* such as repeating, rephrasing, confirmation, etc.

Behaviour Generation

Dialogue actions from the DM are semantic representations of what the system intends to tell the user(s). These can be unpacked and realised simultaneously in a number of complementing modalities: head/body gestures, spoken utterances and deictic references on the application. In order to do this, the behaviour generation module should be able to generate gestures such

as head nods, raising arms, pointing to objects on the map, speaking utterances and making deictic references to objects on the map such as highlighting them, drawing a boundary around them, etc. To generate spoken utterances, we will first generate textual utterances that will then be sent to the TTS engine to be converted into speech. Utterances will be generated by an Utterance Generation module that will use a template-based generation approach (traditionally known as Natural Language Generation). These utterances will be embedded with symbolic names for body/head gestures and deictic references if applicable.

An example would be to translate a pedagogical action such as hinting into a spoken utterance such as "See if you can find the castle over here" and complementing such an utterance with a deictic head/body gesture where the robot turns its head and points to the region on the map where the symbol of a castle is present along with a large circle around the symbol to restrict the space in which the learner has to look for a castle.

Perception Capabilities/Competencies:

The perception competencies consist of three different modules: Kinect, OKAO and Q Sensor (Figure 8). Each module handles the low level communication between the hardware (sensor) and the connected clients using the Thalamus middleware.

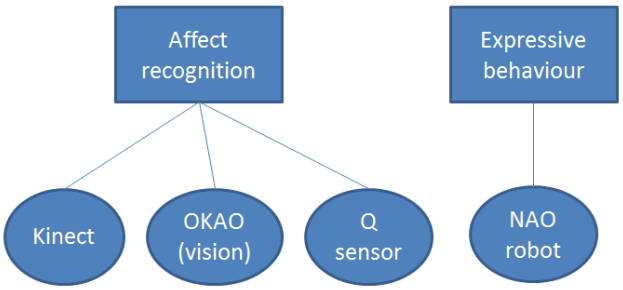

Figure 8: Sensing/Actuating competencies

The outputs interfaces of the perception modules are:

Kinect module - Thalamus perceptions

Head direction (X,Y parameters)
Facial action units (6 facial parameters)
Depth – Body lean angle (1 parameter)
Face Detection (Boolean parameter)

OKAO module - Thalamus perceptions

Smile detection (Smile degree and confidence)
Eye gaze direction (X,Y parameters)
Expression estimation (Expression and confidence)

Q sensor module - Thalamus perceptions

Accelerometer (X,Y,Z parameters)
Skin temperature (1 parameter)
Electro-dermal activity (1 parameter)

The perception modules above will publish the parameters to the connected clients in a predefined transmission rate close to 10Hz. The frequency will be informed and defined accurately after the initial pilot studies.

The affect recognition competency will use as input data derived from the perception modules in order to evaluate the current affective state of the participant. The design, build and evaluation of the Affect recognition competency will be explored in detail in deliverable 4.1.

Source code Repository:

In order to provide a collaborative platform for developing the scenarios, we are using source code repositories running the Mercurial revision control tool. Different repositories were set up for different needs throughout the phases of development of the project.
The structure of repositories is presented in the following list:

MapReadingEMOTE
This repository contains the work-in-progress development regarding the Map Activity / Treasure Hunt application for Scenario 1. It allows all partners to share and have access to intermediate versions of the application while also providing a common space for different partners to collaborate by developing different parts of the application.
Currently all partners have read access to this repository, however, HWU was set as the admin of the repository so they are the ones who define write-access to it.

EnercitiesEMOTE
This repository contains the work-in-progress development regarding the Enercities application for Scenario 2. Its purpose and usage is the same as the MapReadingEMOTE repository, but for the Enercities application. Currently all partners have read access to this repository, however, INESC-ID was set as the admin of the repository so they are the ones who define write access to it.

Enercities-3-Players
This repository contains a version of Enercities which was modified to correctly support 3 human players instead of a robot. This version is the game is used for mock-up studies in school without the robot. Currently all partners have read access to this repository, however, INESC-ID was set as the admin of the repository so they are the ones who define write access to it.

EMOTE-Modules-WIP
This repository contains all the work-in-progress (WIP) and tests regarding individual Thalamus modules that all the different partners may be developing. It allows the partners to develop and share tests, partial modules, and WIP (not working) modules on thing that may even not end up being used in the final scenarios. It serves as a collaborative sandbox for the development of Thalamus modules, which also allows partners to help each other out by debugging the same code. Having all the developed modules in the same repository also helps on sharing and reusing code between different Modules, and also to test interaction between different Modules developed by different partners that use the same types of messages.
All partners have read and write access to this repository.

EMOTE-Modules
This repository contains *release* versions of the modules developed by the partners. The purpose of having this repository is to make it easier to find and user final and correctly working versions of the modules, instead of having to find them in the WIP repository. The partners should upload their modules to this repository only when they have completed a solid version of a module that provides some useful functionality for other partners to use.
All partners have read and write access to this repository.

EMOTE-Scenario1
This repository will hold intermediate *release* versions of the full Scenario 1, and in the end, the final version of the scenario. As such, each version of Scenario 1 uploaded into this repository should contain everything needed to run Scenario 1. That includes a working version of the Map Reading/Treasure Hunt application along with the working versions of the modules used by the scenario, and also any other library, drivers or applications needed to run the scenario, along with a guide on how to install and run it. Its purpose is to provide a central point from which any partner can, at any point, grab a working version of the scenario for use in tests, studies or demonstrations. Currently all partners have read access to this repository, however, HWU was set as the admin of the repository so they are the ones who define write-access to it.

EMOTE-Scenario2
This repository will hold intermediate *release* versions of the full Scenario 2, and in the end, the final version of the scenario. Its purpose and usage is the same as the EMOTE-Scenario1 repository, but for Scenario 2. Currently all partners have read access to this repository, however, INESC-ID was set as the admin of the repository so they are the ones who define write access to it.

Thalamus
This repository is held at the public web-based source code repository Sourceforge [Source code repository]. The Thalamus Framework is developed by INESC-ID, and was made publicly available online, so they are the admins of the repository. All the other partners have public read access to this repository.

Conclusion

In this deliverable we described the physical setup (table stand, hardware, sensors) and the two showcases we will be developing in the project including the current state of development. We proposed an approach for developing an architecture that is required to support the scenarios in the EMOTE project. The architecture section provided a detailed description of the various components in the architecture and how they will work with each other. The source repository has been setup to support software development and promote collaboration between partners.

Future Work

In year 2 of the project, the architecture will be designed as proposed in this deliverable. The architecture will focus on integrating multiple-level (non-symbolic and symbolic) components, including robot interaction with user and multi-touch table and migration facilities. The robot will use speech to interact with the learner; the learner however will provide explicit input from the touch table given that speech recognition is very unlikely to work at the required level in the current state-of-the-art (see D2.1 for discussion).

In year 2 we will focus on integrating the dialogue manager (WP5), perception competencies (WP4), Map application/Enercities and inputs from WPs 2 and 3, based both on learner requirements and on the social-emotional bonding requirements. This will include the expressive behaviour repertoire of the robot (gesture, posture, facial output) and the user-interface both with the table and with the robot. Additionally, the issues related to supporting migration from a robot to a hand-held device will be addressed in the coming year.

References:

Touch table specs, http://multitouch.s3.amazonaws.com/multitaction/downloads/spec/SpecSheet_MT550W8_2013_Jan_a4_webprint.pdf

Touch table video, http://www.youtube.com/watch?v=kIwpc1PDQWY

Aldeberan SDK, http://www.aldebaran-robotics.com/documentation/ref/python-api.html

Source code repository: http://sourceforge.net/p/thalamus/code/ci/default/tree/

Emote Prototype 2013, http://www.macs.hw.ac.uk/~amol/emote/emoteprototype.mp4

Ribeiro, T.; Vala, M.; Paiva, A. Thalamus: Closing the Mind-Body Loop in Interactive Embodied Characters, 12th International Conference on Intelligent Virtual Agents, Santa Cruz, CA, EUA, 2012.

Ribeiro, T.; Vala, M.; Paiva, A. Censys: A Model for Distributed Embodied Cognition. 13th International Conference on Intelligent Virtual Agents, Edinburgh, 2013a.

Kopp, S., Krenn, B., Marsella, S., Marshall, A. N., Pelachaud, C., Pirker, H., et al. (2006). Towards a common framework for multimodal generation: The behavior markup language. In Intelligent virtual agents (IVA)

Reidsma, D. and Welbergen, H. V. BML 1.0 Standard. http://www.mindmakers.org/projects/bml-1-0/wiki/Wiki.

Ribeiro, T.; Dooley, D; Paiva, A. Nutty tracks: symbolic animation pipeline for expressive robotics. In *ACM SIGGRAPH 2013 Posters* (SIGGRAPH '13). ACM, New York, NY, USA, 2013b.

Appendix I -

Let's take a look at the two example BML blocks in **Figure BML3**. In line 2, we see a Gaze behavior, which should start immediately (at time *t=0)*, and finish when the action *speech1* ends. The agent will gaze towards the target referenced as *Character2*. In this example, *start="0"* is an absolute SyncPoint, and *end="speech1:end"* is a relative SyncPoint.

Custom SyncPoints can also be specified, for example, in the middle of the speech text in line 12, by using a *Sync* tag. We can see a reference to this SyncPoint in *start="fine"* in the faceLexeme action on line 6. Please refer to the BML 1.0 Standard specification (Reidsma & Welbergen) for a more detailed description of the mechanisms underlying BML and each of its specific behaviors.

```
1  <bml id="Bml1">
2       <gaze id="gaze1" start="0" end="speech1:end" target="Character2">
3       <speech id="speech1" start="gaze1:ready">
4               <text>How are you feeling today?</text>
5       </speech>
6       <faceLexeme id="face1" lexeme="RAISE_BROWS" start="fine" end="gaze1:end">
7  </bml>
8
9  <bml id="Bml2">
10      <gaze id="gaze1" start="speech1:start" end="speech1:end" target="Character1">
11      <speech id="speech1" start="speech1:end">
12              <text>I'm feeling fine <sync id="fine" /> and sassy.</text>
13      </speech>
14 </bml>
```

Figure BML3: Two example BML blocks of code.

The whole block serves as input to a BML scheduler, which will place them in a plan. This plan will execute each of the behaviours within its defined timing and constraints. By using Thalamus, we can actually have this realization distributed throughout different Realization Modules. In our case, we can have one Thalamus Client subscribing to the *Speech* actions, taking care of only realizing those. We can have another Thalamus Client receiving the *Gaze*, *FaceLexeme* or *Posture* actions in order to blend them and render them as animation on the robot.