DR 5.1: PAL Technical Architecture and Software Architecture

Mario Fumagalli, Bert Bierman, Bernd Kiefer, Joost Broekens, Yiannis Demiris, Mark Neerincx

Mixel, Italy; Produxi, The Netherlands; DFKI, Germany; Delft University of Technology, The Netherlands; Imperial College, United Kingdom; TNO, The Netherlands

⟨`m.fumagalli@mixelweb.it`⟩

Project, project Id:	EU H2020 PAL / PHC-643783
Project start date:	March 1 2013 (48 months)
Due date of deliverable:	March 1 2016
Actual submission date:	March 8, 2016
Lead partner:	Mixel
Revision:	final
Dissemination level:	PU

Executive Summary

This document represents the first deliverable of WP5 and describes the PAL architecture of the IT infrastructure and of its main software modules. The software development is driven by the user requirements analyses of WP1 package. The challenging goals of the PAL project, described and defined in the deliverable DR 1.2, require a sophisticated and complex mechanism of **several interacting software modules**, developed by a multidisciplinary team working in research centers located in **different European countries**.

The main goals of WP5 have been:

1. **Design of the overall IT architecture** capable to fulfill the user requirements
2. Identification of **the software modules** that compose the PAL IT infrastructure
3. **Design of the main data flows** among the software modules
4. **Identification of the teams and leaders in charge** of the development of each software module of the PAL project (i.e., allocation of responsibilities).

This information is provided in the deliverable DR 5.1. Following this step, the design of each software module has been addressed. Since the modules have tight and frequent interactions and several interdependencies, periods of integration have been planned and executed, in which the development teams have been working closely together, in the same physical site, quickly testing ideas and different approaches. To provide the AI based, high level PAL-support and human-machine interactions, a solution based on a sophisticated reasoning mechanism has been defined and implemented by the identification of three main components interacting with each other:

- **Interaction Manager**: a module, based on a dialogue model and ontology, to establish the child-PAL interactions;
- **Child model Agent**: a module tailoring the possible actions based on the analysis of his estimated emotions and sentiments;
- **Adaptive Action Selection Model**: a module, based on HAMMER able to select an appropriate action to propose to the child or to use to interact with the child.

Since the PAL Architecture requires several hardware devices and software modules connected, one with the other, and frequently exchanging data, an **ONLINE** and **REAL-TIME** solution has been adopted. To exchange data among the several software modules a "message dispatcher" approach has been chosen.

1 Tasks, objectives, results

1.1 Actual work performed

In the PAL project a sophisticated reasoning mechanism is being developed that consists of three components: an action proposing component, an action prioritising component and an action selection component. This is a solution to the difficult problem of having a smart agent architecture that needs to cope with different types of data and different types of reasoning to interact with a child in a flexible manner. It is also a solution to the practical problem of having three work packages (WP2, 3, 4) work on action selection at different levels of abstraction. Instead of separating responsibility over the three work packages based on levels of abstraction for the actions (e.g., strategic actions related to goal setting in WP2, low level actions related to behaviours in WP3 and medium level actions related to dialogs in WP 4), we decided that a better way to integrate work in these work packages was to focus on the type of information and reasoning that went on in the technologies developed in these packages. As such, work package 2 focusses on high-level modelling of the childs cognitive-affective processes so that model-based predictions can be made regarding what the child wants and feels. These predictions are used to prioritise action selection developed in work package 3. Further, work package 3 can optimize action selection based on data gathered over time regarding the effectiveness of proposed actions in terms of child sentiment and learning goals (e.g. play a quiz when the kid is at home and feeling good). The actual actions are proposed in work package 4, since this work package focusses on the dialog with the child and has control over what actions should be possible for the robot and its avatar to select at any moment in time.

1.2 Description of logical architecture

The logical architecture needs to enable the following things (see also Figure 1):

- Control of both the Real NAO (i.e. the robot) and the Virtual NAO (i.e. the avatar) through a common interface that enables mood modulation in the same way. This is needed to establish consistency in the behavior of the NAO as perceived by the children (a secondary benefit is that it facilitates testing applications on a virtual and real platform as well as facilitates reuse of games and apps). This means the virtual NAO needs to implement a physical NAO motor controller simulation layer so that the BehaviorManager can also be connected to the virtual NAO (A NOAConnector was developed to connect to Real NAO). NAOs are modules just like all other apps. A NAO registers

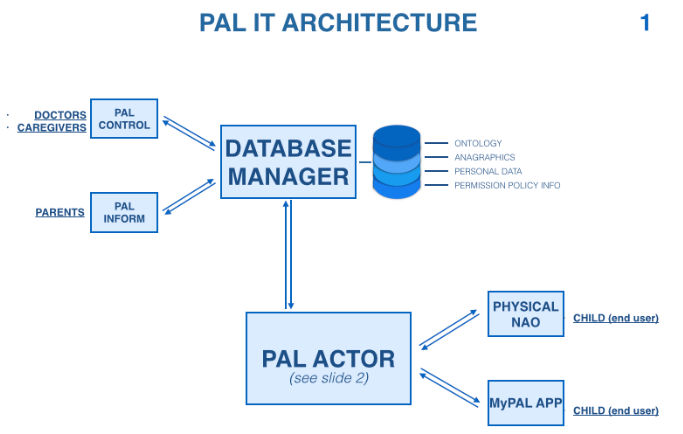


Figure 1: PAL's IT Architecture. Note that the MyPal App contains a virtual NAO module to visualize robot behavior when the physical NAO is absent.

itself and can then be controlled by the system. For simplicity we for now assume only one NAO is present at one time.

- A common action/percept interface that allows modules to register the actions and percepts they have available (such as diary_start, diary_end, diary_goto, NAO_wave, NAO_look_at, NAO_say, etc. for actions and e.g. multiple_choice_input, speech_to_text_input, text_input, cam_input, as inputs). Speech input should also be an input module that registers particular percept types. Modules can access input and output without involvement of the PAL agent logic but need to do so through the central message passing mechanisms (the nexus), so that a databse agent can listen to all messages and stores these in a database. Further, the nexus could notify the PAL agent logic of changes/data available. Modules can't access the database by themselves. So in essence by registering modules a PAL environment is created that can be seen by the different modules (including the PAL logic agent) simply as a set of available actions/percepts that can be used (i.e., the system has a changing "morphology"). An ontology for this environment/morphology is being developed. Modules should be able to get for example a list of action types they need implemented before being able to run (e.g., a card-game needs NAO_point_at, NAO_look_at, NAO_say, NAO_happy, NAO_sad from the PAL environment, if not, it can't run)

The current PAL environment consists of the following modules (see also Figure 2): Nexus, ActionSelector, InteractionManager, ChildModelAgent, BehaviorManager, NAOConnector, PALControl, QuestionGeneratorQuiz-

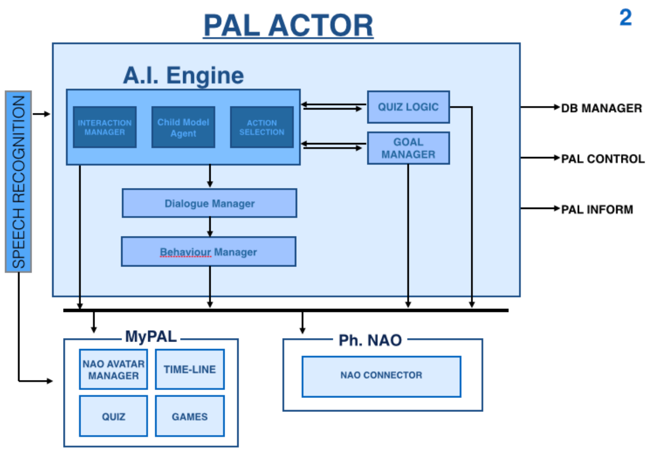


Figure 2: PALs IT Architecture. Note that the MyPal App contains a virtual NAO module to visualize robot behavior when the physical NAO is absent.

Logic and MyPALApplication. Note that as QuizLogic could be interrupted by a dialog with the kid. The QuizLogic is seen as part of the overall AI system. It is true that the quizlogic itself could be run without the 3 AI modules for the PalActor, but it needs to be able to cope with interruptions, and thus is managed by the Pal Actor.

2 List of PAL Software modules

Table 1: PAL software modules.

SW Code	**NAME**	**Device**	**Developer**	**Person in Charge**	**Work Package**
SW01	PAL Control	PC (Browser)	TUD	Joost Broekens	WP2
SW02	PAL Inform	PC (Browser)	TUD	Joost Broekens	WP2
SW03	DB Manager	PAL Server	DFKI	Bernd Kiefer	WP5
SW04	Interaction Manager	PAL Server	DFKI	Bernd Kiefer	WP2
SW05	Child Agent Model	PAL Server	TUD	Joost Broekens	WP2
SW06	Action Selection	PAL Server	IMP	Yiannis Demiris	WP3
SW07	Question Generator	PAL Server	PROD	Bert Bierman	WP5
SW08	Goal Manager	PAL Server	TUD	Joost Broekens	WP2
SW09	Behaviour Manager	PAL Server	PROD	Bert Bierman	WP5
SW10	NAO Connector	NAO PC Controller	PROD	Bert Bierman	WP5
SW11	MyPAL App	Tablet/Smartphone /PC (Browser)	MXL	Diego Fumagalli	WP5
SW12	NAO Avatar Manager	Tablet/Smartphone /PC (Browser)	MXL	Diego Fumagalli	WP5
SW13	Time-Line	Tablet/Smartphone /PC (Browser)	MXL	Diego Fumagalli	WP5
SW14	Quiz	Tablet/Smartphone /PC (Browser)	MXL	Diego Fumagalli	WP5
SW15	Games	Tablet/Smartphone /PC (Browser)	MXL	Diego Fumagalli	WP5
SW16	Message Dispatcher	PAL Server	PROD	Bert Bierman	WP5
SW17	Speech Recognition	External Service	DFKI	Bernd Kiefer	WP4

3 Description of the software modules

3.1 PAL Control

This SW module is the application that will be used by doctors and caregivers to manage the:

- Personal data of each child
- Read all the timeline data generated by each child by using the MyPalApp
- Insert the personal goals of each child
- Review progress and goals performed by each child
- Interact with the DB Manager and PAL Actor module

A detailed description of this module is provided in the WP2 deliverables.

3.2 PAL Inform

This SW module is the application that will be used by parents to overview the status and the (allowed) data of their child. More in particular the parents will be able to:

- Review some of the personal data of their child
- Read some of the timeline data generated by their child by using the MyPalApp
- Review progress and goals performed by their child

A detailed description of this module is provided in the WP2 deliverables.

3.3 DB Manager

We decided to use a custom solution as Database Manager, namely an RDF [1] Store with Forward Chaining Reasoner build at DFKI. It provides its services through the TECS infrastructure, which is used throughout the system, and provides several ways of access, i.e., as a web service and directly as remote procedure call (RPC) service. An extension to make it available via the event passing mechanism, which most of the modules currently use, is planned.

The advantage of using an RDF store is that it not only provides database functionality, but also a convenient way to represent and store the *form* of the data content as an OWL [5] ontology. In this way, the database can also serve as a central reference point for the data representation of content shared between modules.

A more detailed description of the underlying module (HFC) [3] can be found in the D4.1 report of work package 4.

3.4 Interaction Manager

The multimodal interaction manager analyses natural language coming from the user, and generates natural language and gestures for the robot resp. its virtual replacement, the avatar. The generation is based on incoming stimuli, like speech or text input, or high-level action requests coming from some strategic planning component, or any other sensor input, if available. The goal is to create engaging interactions with the users that support the currently active high-level goals.

3.4.1 Interaction with overall system

The interaction manager will get several input types from the nexus, the ones currently foreseen are: input from automatic speech recognition (ASR) or typed natural input, user parameters, like name, age, hobbies, etc. but also more dynamic ones like mood or health data, and also triggers from high-level planning. All these inputs are stored as RDF data, based on an ontology developed as part of the interaction manager, and available to all other partners as a data format specification. When new data is added, a set of declaratively specified reactive rules will propose dialogue moves or other actions and send these proposals to the action selection mechanism that is provided by WP3. The selection mechanism eventually selects one of the proposed actions and sends it back into the nexus. If the proposed action results in dialogue acts, these are turned into verbal output and gestures with the help of a multimodal generation component, which also retrieves parameters from the RDF database to adapt the generation to the users likings and also the PAL systems needs.

3.4.2 Internal Structure

As shown in Figure 3, the interaction manager consists of the RDF store, which also contains the functionality to store incoming data in the format specified by the ontology, thereby making it readily accessible for other components. The second major component is the rule processor for the dialogue management rules, which generates proposals for actions when new incoming data arrives. The rules not only use the new data, but also the interaction history stored in the RDF database to take its decisions. The last two parts are a robust natural language interpretation module (not explicitly shown in the picture), which turns spoken or written utterances into dialogue acts, possibly with an intermediate step that involves a more elaborate semantic format, and a multimodal generation component, which turns outgoing dialogue acts into natural language utterances and gestures. The linguistic resources that are necessary for these components are also maintained by WP4.

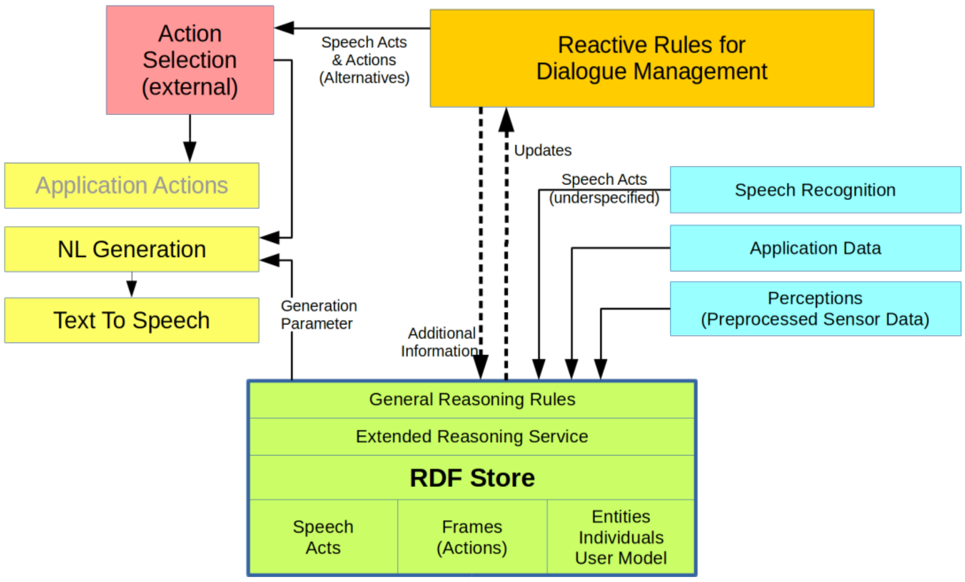

Action Selection (external)
Speech Acts & Actions (Alternatives)
Reactive Rules for Dialogue Management
Application Actions
NL Generation
Text To Speech
Updates
Speech Acts (underspecified)
Speech Recognition
Application Data
Perceptions (Preprocessed Sensor Data)
Generation Parameter
Additional Information
General Reasoning Rules
Extended Reasoning Service
RDF Store
Speech Acts
Frames (Actions)
Entities Individuals User Model

Figure 3: Interaction Manager structure.

3.5 Child Agent Model

The main task of the child model is to provide predicted child state information when this information is not (yet) available from the child. Currently the child model focusses on predicting the current emotional state of the child. The cognitive reasoning simulation of the child model is implemented in the agent programming language GOAL [2]. The emotion simulation is based on the belief-desire theory of emotion (BDTE) [4], a cognitive emotion theory that simulates 7 basic emotions, namely, happy, unhappy, hope, fear, surprise, disappointment and relief. An implementation of BDTE has been made that can be used to simulate emotions in a GOAL agent program.

In this first prototype the child model only listens to the quiz responses of the child. More specifically the model stores how often the child provided correct or incorrect responses. Based on this information the model makes an estimate. It assumes the child will find performing well on the quiz to be desirable. This is based on the presumed desire of the child to be knowledgeable. Therefore, the better the child performs on the quiz, the more positive the resulting emotional responses will be. The model allows for other goals, such as learning goals related to self-management, to be included in the reasoning mechanism, but for this first prototype that had not yet been incorporated (see also Figure 4).

For this first prototype the main goal was to have a complete implementation based on a psychological theory, and to have this implementation interface with the rest of the PAL agent system as provided by work package 3 and 4. Currently the child model publishes user state information to the

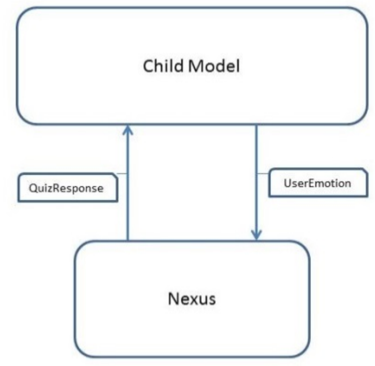


Figure 4: Data Flow between the ChildModel and the Nexus .

nexus and this information is picked up by the action selection mechanisms developed in work package 3 that can then use it do decide upon. Concretely, the user emotional state information is used to determine whether the program should continue the quiz. If the emotional state is positive then the activity is continued, otherwise the action selection mechanism will suggest to stop the quiz. The main task of the child model is, as mentioned, is to provide predicted user state information when this information is not (yet) available from the child.

Finding enough input data and validation mechanisms to improve the child model is a part of future work. An example method to gather additional input data for verification of the models predictions is to use the information of a sentiment mining module or the emotions reported by the child in the diary.

3.6 Action Selection

The Action Selection Module of the PAL Project is based on the HAMMER architecture, which is a versatile, parallel, distributed, and hierarchical architecture for action selection developed at Imperial College London. This architecture executes concurrently, on a multi-threaded infra-structure, several models that assess the quality of the different potential actions suggested by the Natural Multimodal Interaction Module (WP4) and with respect to the childs cognitive-affective state provided by the Child Model (WP2).

The main advantage of running concurrently several models is, first, to combine the abilities of the different models like in a council of expert, which is known to improve the average quality of the decisions and, second, to allow the architecture to explore simultaneously several alternatives and thus to rapidly select the ones that fit the best to the user preferences. Moreover, by using models with online learning features, the architecture is able to follow the changes in the users preferences. Each interaction with the child will allows the PAL system to gather data, which can be used to progressively improve the models employed in the Action Selection Module.

Thanks to these different properties, the Action Selection module will be able to quickly select actions, and in the same time to personalize and to adapt these decisions according to the user preferences and its evolution over the time (see also Figure 5).

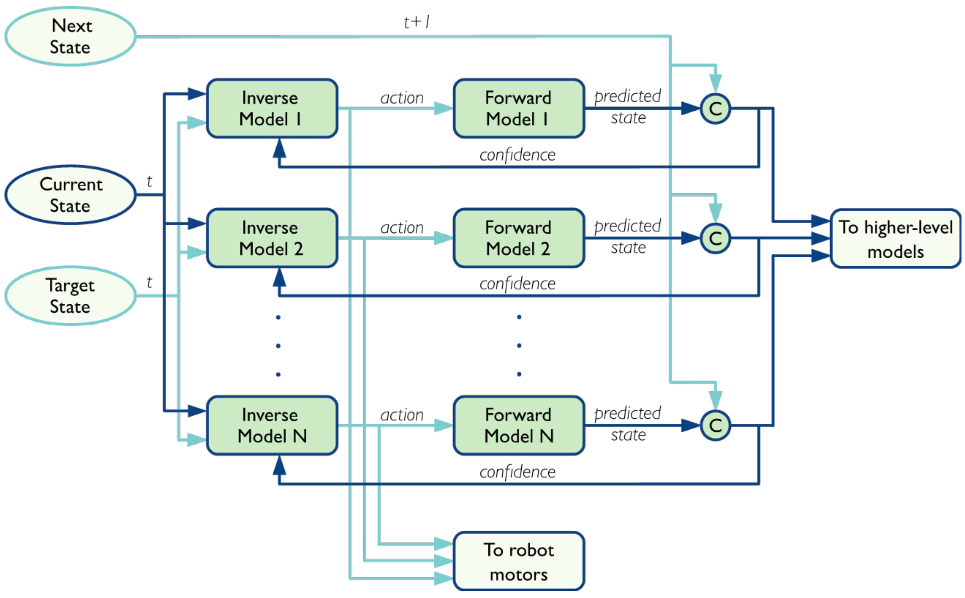


Figure 5: Action selection flow.

3.7 Question Generator

Based on content in a xml files this module creates a database of questions and answers available for a Quiz based on the language of the child. This module is subscribed to the QuizCommand and QuizResponse messages (coming from the Action Selector) and produces QuizQuestion messages. If the QuizCommand contains the Question command a question is selected taken into account:

- if the child already is asked this question before
- the topic of the question
- the goal of the child

If a QuizResponse is received the results are saved into de database. This functionality could be changed in a later stage, e.g., when quiz questions are embedded in the dialog manager, or when a quiz module has more advanced quizzes (e.g., including the difficulty level of the question, etc..)

3.8 Goal Manager

The specific goals to be reached by each child will be identified by his personal doctor/caregiver and input into the DB manager throught the PAL control module. The goal manager constantly checks the data provided by the MyPALApp and by NAO connector to verify if some specific goal has been reached. Goals reached will be stored in the DB Managed and messages will be sent to MyPalApp and to NAO Connector to provide real time feedback of the child.

3.9 Behavior Manager

The BehaviorManager module is responsible for the generation of the movement of the (virtual) NAO, and it subscribes to the messages form the Nexus which contain the multimodal utterance. The utterance consists of the name of the gestures to execute and the text to be spoken as well as the emotional state of the child. From this content it constructs the necessary values to move the joints of the (virtual) NAO which are then published to the Nexus. The task of the BehaviorManager can be divided into two parts i) make the (virtual) NAO look lively and ii) make the make the (virtual) NAO execute the multimodal utterance.

3.9.1 Make the (virtual) NAO look lively

The activity of this part is to realize a (virtual) NAO which is looking lively (do autonomous moves). When the (virtual) NAO is looking alive it will be more engaging for the child. These autonomous moves activity needs to be carefully combined with the deliberate movements resulting from Behavior messages from the PAL system. Currently the autonomous movements are sending randomly modulated signals to the head, the arms and the hips/legs, as well as blinking with the eyes.

3.9.2 The gesture generation

The Behavior Manager (see Figure 6) receives multimodal utterance which contains a name of a gesture, the text to be spoken and the emotional state of the child. From the gesture name the BehaviorManager needs to calculate the position of the joints of the (virtual) NAO and the time available for the joints to reach that position. These conversions are defined in three different ways: i) fixed behaviors, ii) mood behaviors and iii) pre-recorded behaviors.

i) Fixed behaviours. Fixed behaviors are defined in a file which contains values for joints over time for various gestures. The BehaviorManager reads the values and publish them. The NAOConnector listens to that messages and acts upon them.

ii) Mood behaviors. Mood-behaviors are defined using both joint over time values and a procedure to modulate the values by an emotion parameter in terms of positiveness and activeness [7]. The message contains a value for the actual emotional state of the child derived by the ChildModel. For each gesture a sequence of postures is defined. Based on the emotional parameter the postures are adapted with regards to the amplitude of the posture and the speed with which the sequence is executed. The resulting joint and time values are published and received by the NAOConnector. These behaviors originate from the DUT RoboTutor. This mechanism has been tested for correct recognition of affect in the gestures as well as effects on observers in several usage scenarios [6, 8]. Since the RoboTutor is created using C#, a programming language no longer supported by the NAO, the source code has been ported to Java. The rough conversion is done with the CS2J tool from Twiglet Software after which the remaining changes where done by hand. After that the resulting code is integrated into the workflow of the already available BehaviorManager.

iii) Pre-Recorded behaviors. These behaviors are situated on the NAO and mostly contain complex gestures like stand-up and sit-down. The Behavior Manager publishes the name of the gesture to be executed and the NAOConnector processes that message.

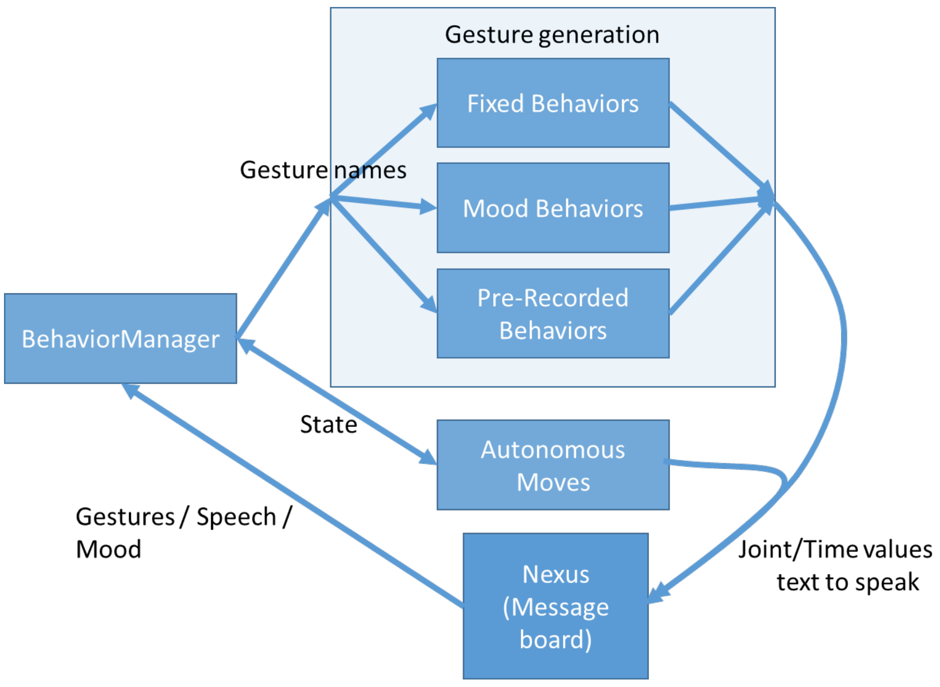


Figure 6: graphical representation of the various components of the BehaviorManager.

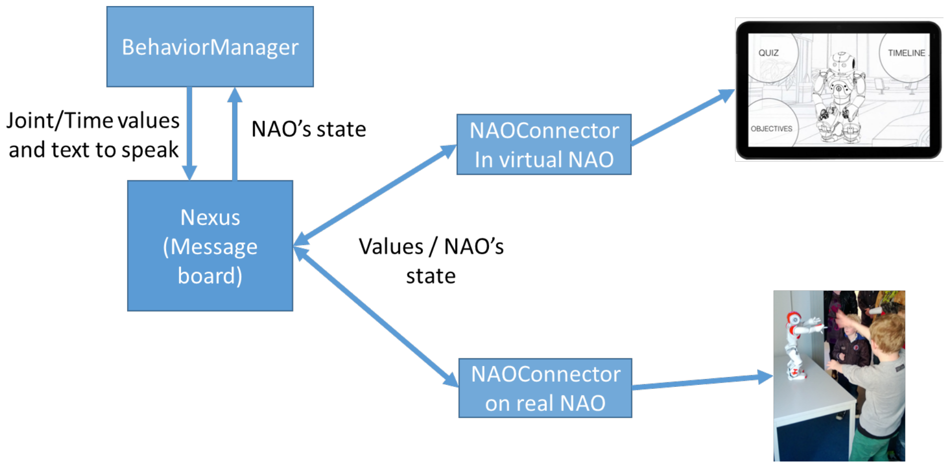


Figure 7: Description of how the different parts are connected.

3.10 NAO Connector

This module is responsible to have the NAO move and/or speak. It subscribes to message containing the actual values of the joint positions and the time to reach the position, and the text to be spoken (published by the BehaviorManager). The information received is converted into actual commands of the NAOs internal system (see Figure 7).

This module has two different implementations:

- An implementation which communicates with the real NAO
- An implementation which is running on the tablet to communicate with the virtual NAO

Both are using the same text-to-speech engine to assure the voices sounds identical. In case of the Italian Mary TTS is used but because there is no support for the Dutch language (yet) eSpeak is applied. Components are created on both NAOs to be able to handle the text-to-speech conversion.

The implementation for the real NAO is a wrapper which converts the data received (from the BehaviorManager into the commands which can be send to the NAOs own execution system. The majority of the messages originate from the autonomous move system. The main issue to solve is handling all messages in parallel without blocking the processes. The module also monitors if an execution and/or speech is finished, if its sensors are touched and processes the sound localization messages. This information is published to the PAL system.

The implementation for the virtual NAO is embedded in the Unity environment in which the MyPAL application is developed and the task is to execute the moving of the joints of the model according to the values received.

If needed the NAOConnector of the real NAO can also be run on the NAO. It simplifies the registration process during the experiment, but then the NAO needs to be equipped with the Java run time environment, which is not installed by default.

3.11 MyPAL App

MyPalApp is the application and one of the front-end parts of PAL System. The children can interact with the PAL System through the application or the Real NAO. All the analysis and the design is focused on usability for children that are 7-15 years old. This application can help children to keep track of their meal, activity, and physical state, such as Glycaemia or emotional state. One of the main component in the App is the NAO Avatar, who helps children during the usage of the application and increase their desire to use it. The application contains several parts that communicate with the entire system through Nexus. MyPalApp is connected with a server and the children must login in to use it.

The application is composed by 4 sections (see also Figure 8):

- Main section
- Objectives section
- Quiz section
- Timeline section

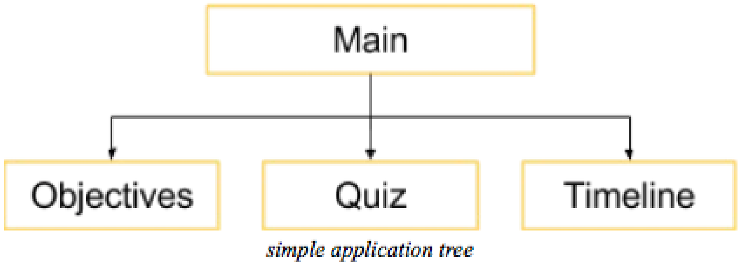


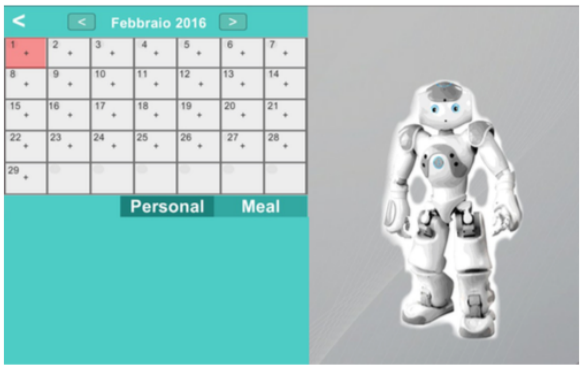
Figure 8: Simple application tree.

3.12 Main Section

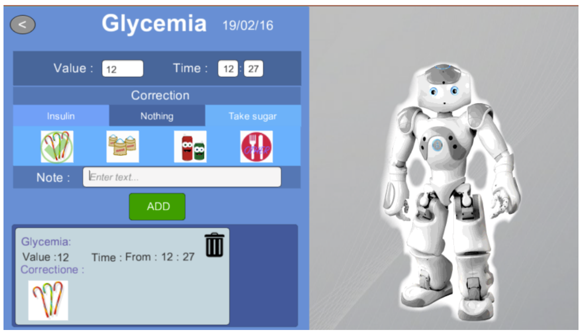
In the Main Section there will be mainly the PAL Avatar (3D Model of NAO) and it is focused on the Child-Avatar (NAO) interaction. The Interaction Manager (PAL Agent Module) will decide the interactions that can be suggested by the system, the possible questions to be asked, useful information to be gained through dialogue, etc.

3.13 Time Line

In the Timeline section the child will able to insert, in a diary frame, everything s/he does during the day (Figure 9a); Child can insert about his meal (Figure 9d), activity (Figure 9c), Glycemia (Figure 9b) or emotional state (Figure 9c) and the NAO Avatar interacts with the child for supporting him/her during the App utilization. Information will be accessible for the caregiver in charge (through the PAL Control panel), because the application will send the information to the PAL database (policies for information exchange are being developed).



(a) Time Line start



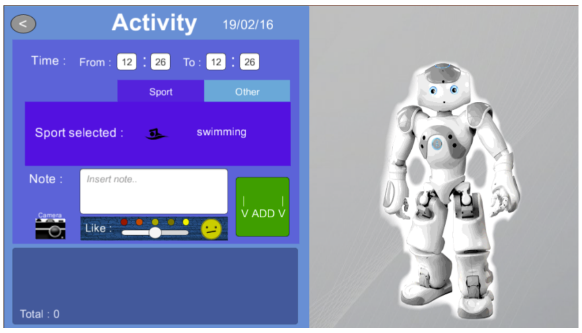
Glycemia 19/02/16

Value : 12 Time : 12 27

Note : Enter text...

Glycemia:
Value :12 Time : From : 12 : 27
Correctione :

(b) Time Line Glycemia



Activity 19/02/16

Time : From : 12 : 26 To : 12 : 26

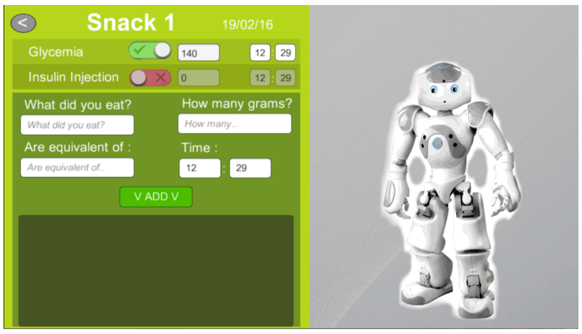
Sport selected : swimming

Note : Insert note..

Like :

Total : 0

(c) Time Line Activity



Snack 1 19/02/16

Glycemia 140 12 29

Insulin Injection 0 12 29

What did you eat? What did you eat?

How many grams? How many..

Are equivalent of : Are equivalent of..

Time : 12 : 29

(d) Time Line Food

Figure 9: Different views of the timeline section

3.14 Quiz

In the Quiz Section the child will have the possibility to play a quiz game with the PAL Avatar. The QuizManager (PAL agent's component) will generate all the questions and answers.

The child is allowed to select his/her answer only by touching the correct one, and s/he can cut off the play at any time s/he wants.

Quiz will be classified by difficulty levels, in such a way to provide an always challenging experience to the player.

3.15 Games

Serious games will be delveloped inside the MyPalApp application to offer to the child the oppurtunity to learn and to review important concepts about his desiease and the correct lifestyles and a potentially fun way.

The games will be designed in such a way to generate a long term engagement about the MyPalApp mbile application. The presence of the NAO avatar will probably be used in each game to provide a family feeling whitin the whole application.

Game benefits and rewards will be provided to the player if specific achievements will be reached,by using the MyPalApp. Examples of goals that will generate rewards are:

- daily use of the application,
- number and type of timeline data inserted,
- number and persentage of correct quiz answered

Examples of rewards that will be provided are:

- more special powers for the games
- new accessories of the NAO avatar (for example: new hats, clothes, glasses, etc)
- new animations for the NAO avatars
- new levels and features of the games

3.16 Message Dispatcher

The Nexus is the module which acts like the message board. It is a publish/subscribe or remote procedure call mechanism through which messages are send throughout the system.

The current implementation is based on the TECS-infrastructure (Thrift Eventbased Communication Service) created by one of the partners (DFKI). The services provide both messaging based on the publish/subscribe mechanism (PS) as well as based on remote procedure calls (RPC). The PS type is used generally and for specific database access the RPC method is used.

This TECS environment provides Thrift-based-tools to create class implementation of messages just by defining the message in thrift syntax. This being the primary reason why this system is selected.

Each module can register to the message-types it needs and produce message-types it is supposed to. Appendix 4.1 provides the full description of the message.

3.17 Speech Recognition

As a preliminary solution, we developed an adapter to the Google Speech API which provides its results as UserTextFeedback events. Since the use of automatic speech recognition will be minor in the first system, this should suffice for the current needs, but will have to be replaced in the long run.

References

[1] Patrick Hayes. RDF semantics. Technical report, W3C, 2004.

[2] Koen V Hindriks and J-J Ch Meyer. Toward a programming theory for rational agents. *Autonomous Agents and Multi-Agent Systems*, 19(1):4–29, 2009.

[3] Hans-Ulrich Krieger. Ontologies and reasoning architecture for PAL. Technical report, DFKI GmbH, 2015.

[4] Rainer Reisenzein. Emotions as metarepresentational states of mind: Naturalizing the belief–desire theory of emotion. *Cognitive Systems Research*, 10(1):6–20, 2009.

[5] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3:79–115, 2005.

[6] Junchao Xu, Joost Broekens, Koen Hindriks, and Mark A Neerincx. Bodily mood expression: Recognize moods from functional behaviors of humanoid robots. In *ICSR 2013*. Springer, 2013.

[7] Junchao Xu, Joost Broekens, Koen Hindriks, and Mark A Neerincx. Mood expression through parameterized functional behavior of robots. In *RO-MAN, 2013 IEEE*, pages 533–540. IEEE, 2013.

[8] Junchao Xu, Joost Broekens, Koen Hindriks, and Mark A Neerincx. Robot mood is contagious: effects of robot body language in the imitation game. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 973–980. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

4 Annexes

4.1 Message Description

Abstract The full description of the message definition.

Availablity Unrestricted.

Message description

The full description of the message defined is:

```
# Names of the various messages #
#################################

Const string PALToAll = ".*";
const string UserTextFeedbackMsg = "UserTextFeedback"
const string UserEmotionMsg = "UserEmotion"
const string QuizResponseMsg = "QuizResponse"
const string QuizCommandMsg = "QuizCommand"
const string QuizQuestionMsg = "QuizQuestion"
const string UserPictureMsg = "UserPicture"
const string DiaryFreeTextMsg = "DiaryFreeText"
const string UserSentimentMsg = "UserSentiment"
const string IntentionListMsg = "IntentionList"
const string IntentionMsg = "Intention"
const string DialogCommandMsg = "DialogCommand"
const string BehaviourMsg = "Behaviour"
const string LowLevelNaoCommandMsg = "LowLevelNaoCommand"
const string LowLevelExecuteCommandMsg = "LowLevelExecuteCommand"

# Names of the various modules #
################################
const string ActionSelector = "ActivitySelector";
const string NAOConnector = "NAOConnector";
const string ChildSimulator = "ChildSimulator";
const string BehaviorManager = "BehaviorManager";
const string Database = "Database";


# Messages related to perception #
###################################

# this message contains any direct textual input coming from the child
either through the SR or the interface
struct UserTextFeedback {
        1:required i32 id;
        2:required string content;
}

# this message contains emotional state information of the child.
struct UserEmotion {
        1:required i32 id;
        2:required string content;
}

# this message contains the response of a child to a question
struct QuizResponse {
        1:required i32 id;
        2:required string category;
        3:required string questionText;
        4:required i32 response;  # 0-based!
        5:required bool correct;
}
```

```
struct UserPicture {
        1:required i32 id;
        2:required string content;
}

# this message contains a valence, arousal description of the child.
struct UserSentiment {
        1:required i32 id;
        2:required string content;
}

struct DiaryFreeText {
        1:required i32 id;
        2:required string content;
}

# Messages related to reasoning cycle #
########################################

# this message contains a list of intentions which can be executed.
struct IntentionList {
        1:required i32 id;
        2:required string content;
}

# this message contains the intention selected from the list
struct Intention {
        1:required i32 id;
        2:required string content;
}

# Messages related to actions #
################################

struct DialogCommand {
        1:required i32 id;
        2:required string content;
}

# this message contains command about the quiz
# [stop, start, question]
struct QuizCommand {
        1:required i32 id;
        2:required string command;
        3:required list<string> topics;
        4:required i32 level;
        5:required string asker;        # [child|robot]
}

# this message contains the data about a question
struct QuizQuestion {
        1:required i32 id;
        2:required string questionText;
        3:required list<string> answers;
        4:required string topic;
        5:required i32 whichCorrect; # 0-based
        6:required string asker;        # [child|robot]
}

# this message contains the information needed to generate the behavior of
the (virtual) NAO
struct Behaviour {
        1:required i32 id;
        2:required string gesture;
        3:required string textToSpeak;
        4:optional string type;
        5:optional double mood;
}

# this message contains commands send to the NAO
# [startbehavior, stopbehavior, getposture, gotoposture]
struct LowLevelNaoCommand {
        1:required i32 id;
        2:required string command;
}


# this message contains the data for the joints values over time
struct LowLevelExecuteCommand {
        1:required i32 id;
        2:required list<string> joints;
        3:required list<list<double>> angles;
        4:required list<list<double>> times;
        5:required string textToSpeak
}
```

Example scenario explaining message flow:
PAL agent/NAO presents itself

- InteractionManager posts **IntentionList** that includes **"**greeting" intention, ChildModel posts **UserEmotion**.
- ActionSelector posts **Intention** "greeting"
- InteractionManager posts **Behavior** as implementation of the "greeting" intention.
- BehaviorManager posts **LowLevelExecuteCommand** (containing the greeting implementation).
- NAOConnector execute motion and speech on real or virtual NAO and posts **LowLevelNAOCommand** when finished execution

PAL agent asks a question to get to know the name OR a closed question

- InteractionManager posts **IntentionList** that includes **"**ask_usermodel_question" intention,
- ChildModel posts **userEmotion**.
- ActionSelector posts **Intention** "ask_usermodel_question"
- InteractionManager posts **DialogCommand** implementation of question "ask_usermodel_question" intention.
- InteractionManager posts **Behavior** implementation. "ask_usermodel_question" intention.
- BehaviorManager posts **LowLevelExecuteCommand** (containing the "ask_usermodel_question" implementation).
- NAOConnector execute motion and speech on real or virtual NAO and posts **LowLevelNAOCommand** when finished execution
- DialogRealizer posts **SRCommand** containing start.

- SRModule posts **UserTextFeedback** containing possible text spoken by the child
- DialogRealizer posts **UserTextFeedback** based on buttonclicks

<off topic talk can be repeated….>

Not going to the same level of detail, the rest of the sample scenario is as follows:
NAO suggest playing a quiz
    Reasoning cycle posts a quizCommand (start) command
QUIZ runs (repeated)
    The QuizEngine posts a next question as a quizCommand (question)
The QuizApp (MyPal) posts a quizResponse
The InteractionManager proposes to quit the quiz with that action in the intentionList
QUIZ ends and NAO praises the child on his/her performance (customized to quiz result)
    Reasoning cycle ends the quiz with a quizCommand(stop) message
    Reasoning cycle proposes and executes a Behavior with customized text