Development of Robot-enhanced Therapy for Children with Autism Spectrum Disorders

Project No. 611391

DREAM
Development of Robot-enhanced Therapy for Children with Autism Spectrum Disorders

Grant Agreement Type: Collaborative Project
Grant Agreement Number: 611391

D3.1 System Architecture

Due date: **1/10/2014**
Submission Date: **1/10/2014**

Start date of project: **01/04/2014** Duration: **54 months**

Organisation name of lead contractor for this deliverable: **University of Skövde**

Responsible Person: **D. Vernon** Revision: **2.0**

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including the Commission Service)	
RE	Restricted to a group specified by the consortium (including the Commission Service)	
CO	Confidential, only for members of the consortium (including the Commission Service)	

Contents

Executive Summary

Deliverable D3.1 defines the DREAM system architecture, i.e. the top-level functional decomposition of the DREAM system, including a definition of the data flows between these functional elements. Specifically, the deliverable provides a functional specification of the three principal sub-systems to be developed in WP4: robot sensing and interpretation; WP5: child behaviour assessment; and WP6: cognitive robot behaviour control, together with a definition of the data connectivity between these sub-systems.

The functional specifications are defined in terms of the action primitives and perception primitives described in Deliverables D1.2 and D1.3, corresponding to the functionality of WP6 and WP4, respectively. The functional specification of the WP5 sub-system is defined in terms of child behaviour classification primitives defined in this deliverable because they have not yet been documented in a WP5 deliverable.

The specifications are grounded in the implementation framework adopted by the DREAM project — the Component-Based Software Engineering (CBSE) component-port-connector model implemented using YARP modules and ports — and therefore this deliverable should be read in tandem with the DREAM Software Engineering Standards described in Deliverable D3.2.

It is envisaged that each primitive (and hence each functional element) will eventually be implemented as a distinct component in the DREAM architecture. However, for the purpose of this top-level definition, we collect them together in three component placeholder components

1. *sensoryInterpretation*
2. *childBehaviourClassification*
3. *cognitiveControl*

corresponding to WP4, WP5, and WP6, respectively.

Nevertheless, the data and control flow associated with each distinct primitive (and corresponding component) is defined here, i.e. the input and output ports that are exposed by each primitive component over which input and output function data and control data can be exchanged with these components. This effectively defines the sub-system interconnectivity.

Where appropriate, the persistent data sources and sinks associated with each sub-system are also identified. Since the DREAM software engineering standards (Deliverable D3.2) require that all component application programming interfaces (APIs) be effected by configurable port interfaces, the identification of all the data interfaces exposed on the ports associated with each primitive component effectively defines the sub-system API.

All data entities are specified by both information content and representation. The content is derived from the primitive specifications in Deliverables D1.2 and D1.3, and Section 4 below while the representations are defined in terms of YARP port protocols.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Paul Baxter, University of Plymouth
Tony Belpaeme, University of Plymouth
Erik Billing, University of Skövde
Pablo Gómez, Vrije Universiteit Brussel
Zhaojie Ju, University of Portsmouth
Corentin Le Molgat, Aldebaran Robotics
Honghai Liu, University of Portsmouth
Hoang Long Cao,Vrije Universiteit Brussel
Serge Thill, University of Skövde
David Vernon, University of Skövde
Hui Yu, University of Portsmouth
Tom Ziemke, University of Skövde

Revision History

Version 1.0 (DV 31-08-2014)
First draft.

Version 1.1 (DV 05-09-2014)
Implementation of the WP6 changes identified at the Amsterdam workshop.

Version 1.2 (DV 23-09-2014)
Removed empty references section.

Version 2.0 (DV 01-10-2014)
Implementation of several suggestions arising from pre-submission review by the DREAM team.

1 Introduction

The DREAM software system comprises three main sub-systems, corresponding to WP4 (Sensing and Interpretation), WP5 (Child Behaviour Analysis) and WP6 (Robot Behaviour). Initially, these three sub-systems are implemented by three place-holder components, as follows.

1. *sensoryInterpretation*
2. *childBehaviourClassification*
3. *cognitiveControl*

In addition, there is one other utility component in the system architecture. This is a Graphic User Interface (GUI) to facilitate external control of the robot by a user (either a therapist or a software developer) and to provide the user with an easily-to-understand view on the current status of the robot control. It will also provide a graphic rendering of the child's behavioural state, degree of engagement, and degree of performance in the current intervention.

The functionality of each sub-system will be developed incrementally as the project progresses and as new components that implement part of the functionality encapsulated in the place-holder components are developed and integrated into the system. During integration, white-box testing will be performed on a system-level by removing the driver and stub functions that simulate the output and input of data in the top-level system architecture, i.e. in one of the three components above, allowing that source and sink functionality to be provided instead by the component being integrated.

The functionality of *sensoryInterpretation* is specified completely by the 25 perception primitives defined in Section 2 of Deliverable D1.3 (Child Behaviour Specification).

The functionality of *cognitiveControl* is specified partially by the seven action primitives defined in Section 2 of Deliverable D1.2 (Robot Behaviour Specification). This is only a partial specification because the basis for invoking each of these action primitives has not yet been defined (whereas, in the case of *sensoryInterpretation*, all of the primitives are continually invoked to monitor the status of the robot's environment). Specifically, these seven action primitives only reflect the functionality required to carry out the robot behaviours that are required to follow the scripted interventions described in Deliverable D1.1. Inevitably, additional functionality will be required to achieve supervised autonomy on the part of the robot when there are even slight deviations from this pattern of behaviour. Consequently, we augment this functional definition in Section 5 to reflect more cognitive behaviour.

The functionality of *childBehaviourClassification* is encapsulated by three primitives: `getChildBehaviour()`, `getChildMotivation()`, and `getChildPerformance()`. These primitives are defined in Section 4.

2 Primitives and Ports

The parameters of every primitive in the three sub-systems are explosed by two dedicated ports, one for input and one for output, with the arguments encapsulated in a YARP vector or bottle, whichever is more appropriate.

The general naming convention for the two ports is `/<primitive name>:i` for input and `/<primitive name>:o` for output.

3 The *sensoryInterpretation* Component

The following are the primitives, associated input and output ports, and port types in the *sensoryInterpretation* component. Not all primitives have input parameters. The components for those that do are stateful, i.e. once the associated argument values are set, they remain persistently in that state until reset by another input.

```
checkMutualGaze()
   /sensoryInterpretation/checkMutualGaze:o
   BufferedPort<VectorOf<int>>

getArmAngle(left_azimuth, elevation, right_azimuth, elevation)
   /sensoryInterpretation/getArmAngle:o
   BufferedPort<VectorOf<double>>

getBody(body_x, y, z)
   /sensoryInterpretation/getBody:o
   BufferedPort<VectorOf<double>>

getBodyPose(<joint_i>)
   /sensoryInterpretation/getBodyPose:o
   BufferedPort<VectorOf<double>>

getEyeGaze(eye, x, y, z)
   /sensoryInterpretation/getEyeGaze:i
   /sensoryInterpretation/getEyeGaze:o
   BufferedPort<VectorOf<double>>

getEyes(eyeL_x, y, z, eyeR_x, y, z)
   /sensoryInterpretation/getEyes:o
   BufferedPort<VectorOf<double>>

getFaces(<x, y, z>)
   /sensoryInterpretation/getFaces:o
   BufferedPort<VectorOf<double>>

getGripLocation(object_x, y, z, grip_x, y, z)
   /sensoryInterpretation/getGripLocation:i
   /sensoryInterpretation/getGripLocation:o
   BufferedPort<VectorOf<double>>

getHands(<x, y, z>)
   /sensoryInterpretation/getHands:o
   BufferedPort<VectorOf<double>>

getHead(head_x, y, z)
   /sensoryInterpretation/getHead:o
   BufferedPort<VectorOf<double>>

getHeadGaze(<plane_x, y, z>, x, y, z)
   /sensoryInterpretation/getHeadGaze:i
   /sensoryInterpretation/getHeadGaze:o
   BufferedPort<VectorOf<double>>
```

```
getHeadGaze(x, y, z)
   /sensoryInterpretation/getHeadGaze:o
   BufferedPort<VectorOf<double>>

getObjects(<x, y, z>)
   /sensoryInterpretation/getObjects:o
   BufferedPort<VectorOf<double>>

getObjects(centre_x, y, z, radius, <x, y, z>)
   /sensoryInterpretation/getObjects:i
   /sensoryInterpretation/getObjects:o
   BufferedPort<VectorOf<double>>

getObjectTableDistance(object_x, y, z, vertical_distance)
   /sensoryInterpretation/getObjectTableDistance:i
   /sensoryInterpretation/getObjectTableDistance:o
   BufferedPort<VectorOf<double>>

getSoundDirection(threshold, azimuth, elevation)
   /sensoryInterpretation/getSoundDirection:i
   /sensoryInterpretation/getSoundDirection:o
   BufferedPort<VectorOf<double>>

identifyFace(x, y, z, face_id)
   /sensoryInterpretation/identifyFace:i
   /sensoryInterpretation/identifyFace:o
   BufferedPort<VectorOf<double>>

identifyFaceExpression(x, y, z, expression_id)
   /sensoryInterpretation/identifyFaceExpression:i
   /sensoryInterpretation/identifyFaceExpression:o
   BufferedPort<VectorOf<double>>

identifyObject(x, y, z, object_id)
   /sensoryInterpretation/identifyObject:i
   /sensoryInterpretation/identifyObject:o
   BufferedPort<VectorOf<double>>

identifyTrajectory(<x, y, z, t>, trajectory_descriptor)
   /sensoryInterpretation/identifyTrajectory:i
   /sensoryInterpretation/identifyTrajectory:o
   BufferedPort<VectorOf<double>>

identifyVoice(voice_descriptor)
   /sensoryInterpretation/identifyVoice:o
   BufferedPort<VectorOf<int>>

recognizeSpeech(text)
   /sensoryInterpretation/recognizeSpeech:o
   BufferedPort<Bottle>

trackFace(seed_x, y, z, time_interval, projected_x, y, z)
```

```
   /sensoryInterpretation/trackFace:i
   /sensoryInterpretation/trackFace:o
   BufferedPort<VectorOf<double>>

trackHand(seed_x, y, z, time_interval, projected_x, y, z)
   /sensoryInterpretation/trackFace:i
   /sensoryInterpretation/trackFace:o
   BufferedPort<VectorOf<double>>

trackObject(objectDescriptor, seed_x, y, z, time_interval, projected_x, y, z)
   /sensoryInterpretation/trackObject:i
   /sensoryInterpretation/trackObject:o
   BufferedPort<VectorOf<double>>
```

4 The *childBehaviourClassification* Component

The functionality of the *childBehaviourClassification* component is encapsulated by three primitives, as follows.

1. `getChildBehaviour()`
2. `getChildMotivation()`
3. `getChildPerformance()`

The `getChildBehaviour()` primitive classifies the child's behaviour on the basis of current percepts. It produces a set of number pairs where the first element of each pair represents a child state and the second element the likelihood that the child is in that state. Thus, the primitive effectively produces a discrete probability distribution across the space of child states.

The `getChildMotivation()` primitive determines the degree of motivation and engagement on the basis of the temporal sequence of child behaviour states, quantifying the extent the children are motivated to participate in the tasks with the robot and detect in particular when their attention is lost. It produces two numbers, the first representing an estimate of the degree of engagement and the second representing an indication of confidence in that estimate.

The `getChildPerformance()` primitive determines the degree of performance of the child on the basis of a temporal sequence of child behaviour states, quantifying the performance of the children in the therapeutic sessions. It produces two numbers, the first representing an estimate of the degree of performance and the second representing an indication of confidence in that estimate.

In summary, the following are the primitives, associated ports, and port types for the *childBehaviourClassification* component.

```
getChildBehaviour(<state, probability>)
   /childBehaviourClassification/getChildBehaviour:o
   BufferedPort<VectorOf<double>>

getChildMotivation(degree_of_engagement, confidence)
   /childBehaviourClassification/getChildMotivation:o
   BufferedPort<VectorOf<double>>

getChildPerformance(degree_of_performance, confidence)
   /childBehaviourClassification/getChildPerformance:o
   BufferedPort<VectorOf<double>>
```

5 The *cognitiveControl* Component

As noted already, the functionality of the *cognitiveControl* placeholder component is specified partially by the seven action primitives defined in Section 2 of Deliverable D1.2 (Robot Behaviour Specification). These seven action primitives reflect the functionality required to carry out the robot behaviours that are required to follow the scripted interventions described in Deliverable D1.1. However, by itself this is not sufficient because there are two additional considerations for effective robot control in the context of supervised autonomy.

First, the robot control is focussed on interaction with the child. Even when everything goes according to plan and the intervention scripts can be followed exactly, there is a need to adapt the robot behaviour to ensure its actions are in sync with those of the child. In a sense, the robot's behaviour, in an ideal situation, is entrained by the child's (perhaps it would be better to say that the child's and the robot's actions are mutually entrained). This means that the robot control isn't just a simple case of effecting playback of scripted motions and there has to be an element of adaptivity in the robot behaviour, even in the case of following the scripted interventions defined in Deliverable D1.1).

Second, individual children child are likely to deviate from the generalized expectations underlying the scripted interventions and the stereotyped standard behaviours they describe. Even slight deviations call for adaptivity on the part of the robot. WP5 — Child Behaviour Analysis — provides crucial information on the extent of these deviation in the guise of the engagement and performance indicators. As these deviate from acceptable norms, the robot controller has to adapt, either to suspend the interaction and hand over control of the situation to the therapist (this is the default scenario envisaged in Deliverable D1.1) or to select some action — autonomously — that will re-engage the child and hopefully improve his or her performance. This adaptivity requires more cognitive control. However, that said, it is not possible to specify exactly what kind of behaviour is required because we don't know the extent to which autonomous interaction by the robot is required or useful when dealing with children with ASD. To a large extent, this is exactly the goal of Work Package 2. At this juncture, all we can specify is that the cognitive control will have to involve reactive (i.e. reflexive) and life-like involuntary behaviours, attentive behaviours, expressive behaviours, and deliberative (i.e. anticipatory) behaviours. These behaviours will be effected through the implementation of an appropriate cognitive architecture (to be designed in Work Package 6) with action selection being guided by the constraints imposed by the parameters of desirable interaction to be defined in Work Package 2.

However, while the *cognitiveControl* placeholder component cognitive functionality will be developed in line with the constraints and insights gained in Work Package 2, we can identify here the manner in which the component will expose the current status of its control behaviour so that this can be used by the other two placeholder components. Specifically, the `cognitiveControl` placeholder component will have a `getInterventionState()` primitive and an associated output port that will identify the intervention that is currently being enacted and the current phase within that intervention. To facilitate this, each intervention will be defined as a finite state automaton (equivalently, a state transition diagram) with uniquely-labelled states to provide a common reference for all other components.

While a `getInterventionState()` primitive handles the situation where the robot control is following the prescribed intervention, even when the robot is adaptively entrained with the child's behaviour, it cannot capture the cognitive behaviour of the robot when dealing with situations that deviate from this script because of diminished performance or engagement on the part of the child. This is true not only because the required robot behaviours have not yet been identified (as discussed above)

but also because cognition and autonomy — even supervised autonomy — by definition precludes *a priori* prescriptive behavioural description by an external agent, including a software designer. To compensate for this, the `getInterventionState()` primitive will have a third output parameter to flag situations when the controller is (cognitively) handling an unexpected deviation from a given intervention state (typcially as a result of the child's behaviour exhibiting diminished engagement and/or performance in the current intervention). This will allow the other components, and the user (either a therapist or a software developer), to be informed about what is going on in the cognitive controller, at least in general.

The following are the primitives and associated ports and port types in the *cognitiveControl* component. The first seven are derived directly from Deliverable D1.2. The eight and ninth are addtional utility primitives to enable and disable the robot. The tenth is the `getInterventionState()` primitive described above.

```
grip()
   /cognitiveControl/grip:i
   BufferedPort<VectorOf<int>>

moveHand(handDescriptor, x, y, z, roll)
   /cognitiveControl/moveHand:i
   BufferedPort<VectorOf<double>>

moveHead (x, y, z)
   /cognitiveControl/moveHead:i
   BufferedPort<VectorOf<double>>

moveSequence(sequenceDescriptor)
   /cognitiveControl/moveSequence:i
   BufferedPort<VectorOf<int>>

moveTorso (x, y, z)
   /cognitiveControl/moveTorso:i
   BufferedPort<VectorOf<double>>

release()
   /cognitiveControl/release:i
   BufferedPort<VectorOf<int>>

say(text, tone)
   /cognitiveControl/say:i
   BufferedPort<Bottle>

enableRobot();
   /cognitiveControl/enableRobot:i
   BufferedPort<VectorOf<int>>

disableRobot();
   /cognitiveControl/disableRobot:i
   BufferedPort<VectorOf<int>>

getInterventionStatus(interventionDescriptor, stateDescriptor,
                      cognitiveModeDescriptor)
   /cognitiveControl/getInterventionStatus:o
```

```
BufferedPort<VectorOf<int>>
```

6 Inter-connectivity between the components

Any component that needs to access the information exposed on the ports associated with a primitive has to have equivalent ports of its own (so that the two ports can be connected) but reversing the input/output designation. Thus, for example, one would connect

`/cognitiveController/identifyObject:o` to `/sensoryInterpretation/identifyObject:i`
`/sensoryInterpretation/identifyObject:o` to `/cognitiveController/identifyObject:i`.

This would allow *cognitiveController* to send the x, y, and z location of the object to be identified to *sensoryInterpretation* and then to receive the identification number of that object from *sensoryInterpretation* (see definition of `identifyObject()` in Deliverable D1.3).

Regarding the connectivity between the six components, the following principles apply.

- Each *sensoryInterpretation* output port is connected to the counterpart input port in the *cognitiveController* and *childBehaviourClassification* components
- Each *sensoryInterpretation* input port is connected to the counterpart output port in the *cognitiveController* component (but not the *childBehaviourClassification* component).
- Each *childBehaviourClassification* output port is connected to the counterpart input port in the *cognitiveController* component.
- Each *cognitiveController* input port is, typically, *not* connected to any counterpart output port in either the *sensoryInterpretation* or *childBehaviourClassification* components since these ports will be typically be used only internally within the components that will constitute the *cognitiveController* as it is developed.
- All of the *cognitiveController* input and output ports will be connected to a Graphic User Interface (GUI) to facilitate external control of the robot by a user (either a therapist or a software developer) and to provide the user with an easily-comprehended view on the current status of the robot control. Furthermore, the child state, degree of engagement, and degree of performance output ports in the *childBehaviourClassification* component will also be connected to the GUI and graphically rendered in a suitable manner.

The complete system architecture is shown in Figure 1.

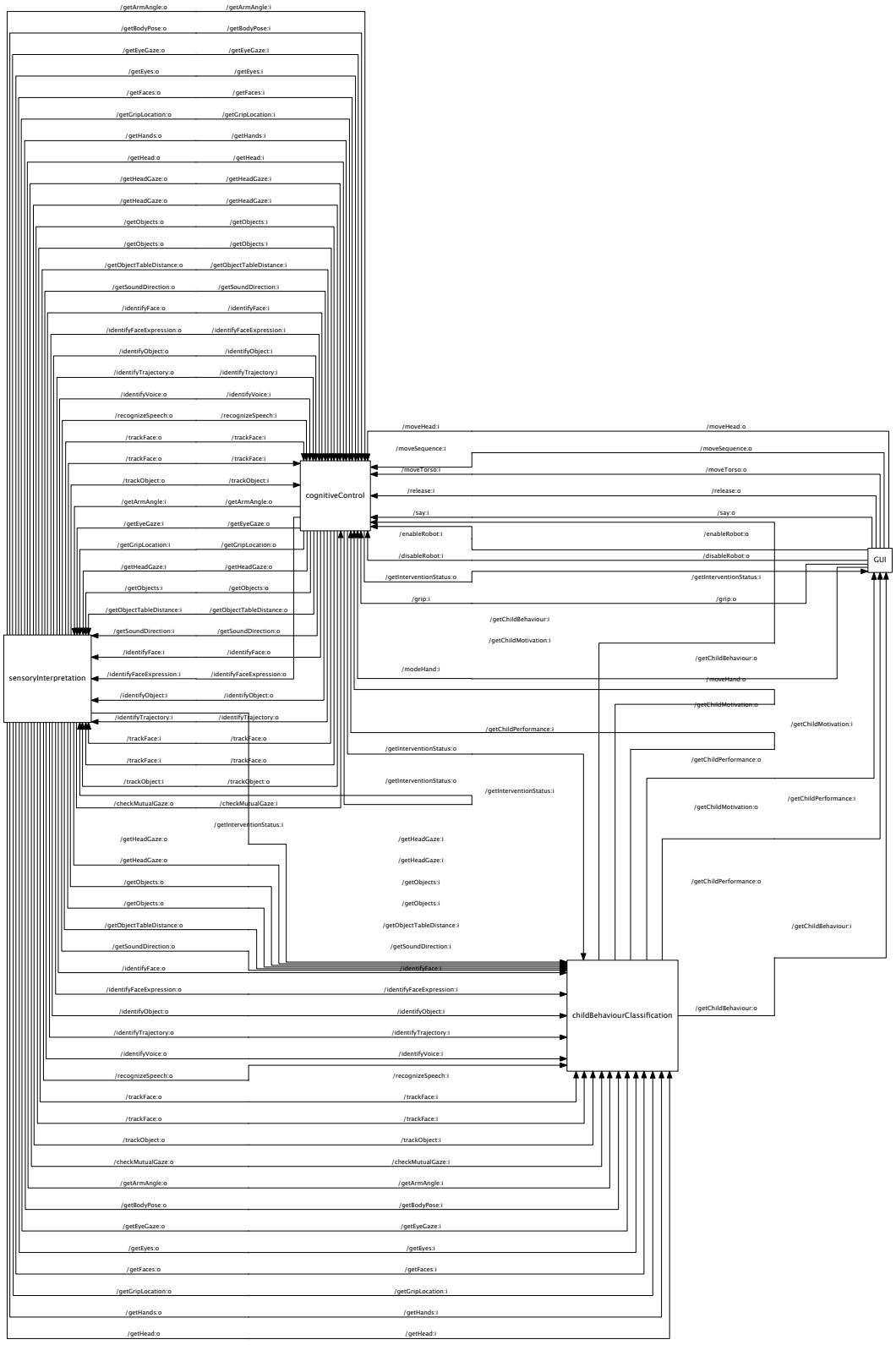

Figure 1: The DREAM system architecture.