



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

**D3.4.3 System Integration Progress Report**

Due date: 1/4/2017  
Submission Date: 7/4/2017

Start date of project: **01/04/2014**

Duration: **54 months**

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

Responsible Person: **E. Billing**

Revision: **1**

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



## Contents

<b>Executive Summary</b>	<b>3</b>
<b>Principal Contributors</b>	<b>4</b>
<b>Revision History</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
<b>2 Integration meetings</b>	<b>5</b>
2.1 Developers meeting at HIS . . . . .	5
2.2 Integration week at PORT . . . . .	5
2.3 Integration follow-up at UBB . . . . .	6
<b>3 Integrated Software</b>	<b>6</b>
<b>4 DREAM system architecture</b>	<b>7</b>
4.1 Reorganisation of the DREAM folder hierarchy . . . . .	7
4.2 Changes in the DREAM component architecture . . . . .	9
<b>5 Utilities</b>	<b>9</b>
5.1 DREAM Boxology . . . . .	9
5.2 Script Generator . . . . .	11
5.3 User Data Export . . . . .	11
5.4 User Model Creator . . . . .	11
<b>Appendices</b>	<b>13</b>
<b>A Notes from the DREAM Developers Meeting</b>	<b>13</b>
<b>B Notes from the UBB integration week</b>	<b>13</b>
<b>C Integration Reports</b>	<b>13</b>



### Executive Summary

Deliverable D3.4 is an annual progress report on the integration of the software developed in work packages WP4, WP5, and WP6. This is the Month 36 progress report. The third year of the project has been a critical period in terms of software integration. From leaving year two with a skeleton for the complete system and a smaller proportion of all required software components integrated, we reached a complete system working in a clinical setting with ASD children by the end of February 2017. As such, this past year has been the time where many pieces came together, including the organization of three larger developer meetings.

The QA procedure and software standards developed during the first two years have to a large degree remained unchanged during this period. Two significant architectural changes have however been issued. The system folder hierarchy has been updated to clearly separate system components, being part of the final DREAM system, from test and utility software. Furthermore, the component organization within WP4 (sensory analysis subsystem) has been updated. Four additional utility applications has been developed, supporting both developers and therapists in working with the DREAM system. *DREAM Boxology* is an application for creating, modifying, and visualizing system configurations. *Script Generator* and *User Model Creator* are tools for generating and modifying intervention scripts and the participant database, respectively. Finally, *User Data Export* allows exporting of participant information into Excel format, as used by therapists.



## Principal Contributors

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

Erik Billing, University of Skövde  
Robert Homewood, University of Skövde  
James Kennedy, Plymouth University  
Emmanuel Senft, Plymouth University  
David Vernon, University of Skövde

## Revision History

Version 1.0 (P.G. 28-03-2017)

First draft.

Version 1.1 (P.G. 06-04-2017)

Second draft including all sections.

Version 1.2 (P.G. 06-04-2017)

Fixed page reference issues in Appendix C.

Version 1.3 (P.G. 06-04-2017)

Updated with clarification in relation to UBB meeting.

Version 2.0 (P.G. 07-04-2017)

Proofed version for submission.

## 1 Introduction

The third year of the project has been the time when the work conducted within WP3 has been put into practice. During this period, fifteen new system components have been integrated, undergoing the QA and integration procedure developed within the project [6]. This led up to a complete running DREAM system used for conducting therapy interventions during February and March, 2017.

Sec. 2 summarizes three developer meetings held during the past year, facilitating integration of system components, as discussed in Sec. 3. During the first meeting, two significant changes to the DREAM architecture were discussed and later implemented. These changes are presented in Sec. 4. Sec. 5 presents four new utility applications developed in order to facilitate maintenance and therapist's work with the DREAM system.

The past year also meant changes in the staff working within WP3. As David Vernon, previously responsible for WP3, left his position with the University of Skövde, he unfortunately also had to leave the work in DREAM. Erik Billing, previously active both within WP3 and WP5, has taken over David's role as research director and responsible for WP3, from January 1<sup>st</sup> 2017.

## 2 Integration meetings

During the past year, three physical meetings with all developers have played a key role for integrating and aligning software development within the project. In addition, weekly interactions over Skype and email, as well as a couple of 1-1 partner meetings, have continued to strengthen interaction in the interim of the three larger meetings listed below.

### 2.1 Developers meeting at HIS

June 7-8, 2016, a developers planning meeting was held at the University of Skövde. The meeting was initiated to strengthen interaction between the software developing parties in WP3, WP4, and WP5 (Recommendation 5) but, with the presence of UBB, also served to clarify details of system functionality in relation to WP2. Following our efforts to facilitate the integration process from a developers' point of view, discussed in D3.4.2, the meeting comprised a walk-through of quality assurance (D.3.3) and corresponding integration procedure updates Spring 2016, c.f. the Dream Wiki [5]. The tools `yarpGenerator` and `componentChecker` (c.f., D3.4.2), were also presented and discussed.

The meeting resulted in a number of concretizations of details in the component specifications and an agreed deadline (November 18) for submitting sixteen out of eighteen components necessary for running the complete DREAM system in a clinical setting, as specified by the roll-out plan [4]. The most significant impact was within WP5, where the exact roles of these components were clarified. Please refer to Appendix A for details.

### 2.2 Integration week at PORT

January, 11-13, 2017, a second developers meeting was held at the University of Portsmouth. The meeting comprised a similar set of people as the former HIS meeting, but with a more direct focus on software integration and testing, with the specific target of getting the complete DREAM system running.

During the meeting, several integration issues were identified and some of them were also resolved. Thanks to the modular architecture and the extensive set of unit tests provided with each individual system component, different subsections of the DREAM architecture could be executed and tested

individually. Large parts of the complete architecture were successfully integrated during this meeting, but due to new, previously unseen problems, the complete system did not run. The progress during and after the meeting is documented using the Redmine Issue tracker [3] and the specific problems preventing execution of the complete system are documented as Bug #46. These problems were later resolved and allowed testing of the DREAM system during the follow-up week in Romania.

### 2.3 Integration follow-up at UBB

In order to prepare for interventions conducted as part of WP2 (reported in D2.2 and D2.3), a follow-up meeting for developers was arranged at UBB February 13-17, 2017. Intense work by the different developing partners and continuous tests of the complete system at HIS before this meeting meant that the team achieved a running system, and four trials with typically developing children were conducted during this week. The meeting also comprised a demo for the press<sup>1</sup>, February 16<sup>th</sup>. Technical notes from the meeting are included as Appendix B.

While the system did run successfully at the end of the meeting, a number of bugs and issues remained, listed in Sec. 3 of Appendix B. The work has continued after the meeting and at the time of writing, most of these issues have been resolved. A number of less critical issues and occasional crashes remain at the present time. Updates to resolve these problems will continue along with development of new functionality.

## 3 Integrated Software

By the end of year two (D3.4.2), eight software components, *actuationSimulator*, *eyeBlinking*, *fallingReaction*, *socialReaction*, *naoInterface*, *reactiveSystemGui*, *sensoryInterpretationLogger*, and *componentChecker*, were integrated into the release version of the DREAM system, following the guide for submitting software for integration available at the DREAM Wiki [5]. *actuationSimulator*, *reactiveSystemGui*, *sensoryInterpretationLogger* and *componentChecker*, were created for testing and support purposes, and are therefore not part of the roll-out plan [4]. Furthermore, following the roll-out plan, *eyeBlinking*, *socialReaction*, and *fallingReaction*, were merged into the new component *attentionReactionSubsystem*.

During year three, the following fifteen new system components have been integrated into the DREAM architecture, following the roll-out plan and QA procedure:

- *actuationSubsystem* (WP6)
- *assessChildEngagement* (WP5)
- *assessChildPerformance* (WP5)
- *attentionReactionSubsystem* (WP6)
- *cameraSelection* (WP4)
- *deliberativeSubsystem* (WP6)
- *kinectSource* (WP4)

---

<sup>1</sup>The press demo held February 16<sup>th</sup> resulted in a newspaper article: <https://evonews.com/tech-science/2017/feb/18/exclusive-how-robots-are-changing-therapy-scientists-closing-in-on-a-breakthrough-ethical-questions-arise>.

- sandtrayEvent (WP6)
- sandtrayServer (WP6)
- scriptManager (WP6)
- selfMonitoringSubsystem (WP6)
- sensoryAnalysis (WP4)
- systemGUI (WP6)
- usbCameraSource (WP4)
- userModel (WP6)

Documentation for all integrated components can be found in the Component Reference Manual [1].

To keep track of component integration in relation to the roll-out plan, a page for the integration status was created in the DREAM Wiki [2]. This page was continuously updated as new components were integrated into the release version of the system.

As examples of the QA procedure, the integration process for two components, *systemGUI* and *assessChildPerformance*, are documented in Appendix C. In addition to the fifteen components listed here, approximately twenty components were created as part of the unit tests, constituting drivers and stubs for execution of the individual system components.

Two components from the roll-out plan have not been submitted for integration: *mapFromPerceptsToBehaviour* (WP5) and *proboInterface* (WP6). During the HIS developers meeting (Sec. 2.1), the exact role for the child behavior classifications (WP5) in relation to cognitive control (WP6) was discussed and clarified. We realized that the basic functionality provided by *mapFromPerceptsToBehaviour*, as required by WP6 components, was already implemented in the sensory interpretation subsystem (WP4). Development of *mapFromPerceptsToBehaviour* was therefore postponed in order to prioritize implementation of *assessChildEngagement* and *assessChildPerformance*, both critical for WP6 and complete system functionality. See Sec. 7 in Appendix A for details. The *proboInterface* was also postponed as it remains unused in the system evaluation conducted within WP2, using the Nao-robot. Both components are scheduled for integration during the final year of the project.

## 4 DREAM system architecture

Over the past year, two significant architectural changes have been made, affecting the structure of the DREAM system and the roll-out plan.

### 4.1 Reorganisation of the DREAM folder hierarchy

Following the integration meeting in Skövde, June 2016 (Sec. 2.1), we decided separate *system components*, listed in the roll-out plan, from *test components* that were not in the roll-out plan and merely created as part of the QA procedure. This change was implemented in the DREAM folder hierarchy on the SVN by adding a new folder *release/test* containing test components, while keeping only system components in the original *release/components* folder. An illustration of the updated folder hierarchy is visible in Fig. 1. Please refer to Appendix A for further details.

Additionally, as listed in D3.4.2, two utility programs had been developed and to avoid convolution with other parts of the DREAM software, these utilities were placed in a new folder *release/tools*. During the past year, an additional four utilities have been developed, presented shortly in Sec. 5.

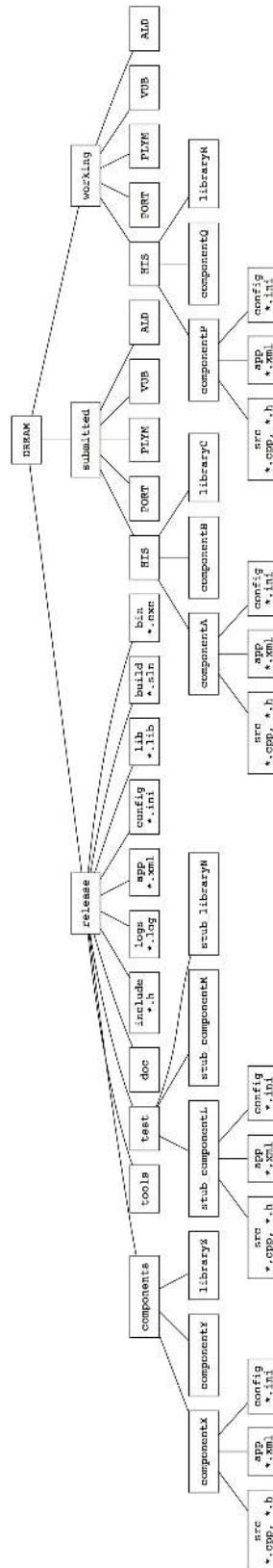


Figure 1: The updated directory structure separating test and utility components from pure system components.



## 4.2 Changes in the DREAM component architecture

As presented in Sec. 3, all sixteen components specified for integration up until year three have been integrated according to the roll-out plan [4], leaving only two components yet to be integrated. However, for technical reasons, one architectural change within WP4 was issued during this period. The roll-out plan specifies four components within WP4: *cameraSelection*, *kinectSource*, *usbCameraSource*, and *sensoryAnalysis*. These four components communicate over data-intensive connections transmitting several parallel video streams, depth data, and motion information. While YARP does a good job handling port communication efficiently, these data-intensive connections did add a significant overhead to the already CPU intensive algorithms for sensory analysis created within WP4. Port communication also requires synchronization of parallel data streams, adding an extra challenge to the time sensitive fusion of RGB and RGBD data from usb cameras and Kinects. For these reasons, the four components were merged into a single *sensoryAnalysis* component. *cameraSelection*, *kinectSource*, and *usbCameraSource* are however still kept and maintained as parts of the system although they are currently not used in the main system configuration. Please refer to D4.3.2 for further details.

While this reorganization of software within the Sensory Interpretation subsystem has been large, the interface to the other two subsystems, Child Behaviour Classification and Cognitive Control, has remained unchanged. As a result, this reorganization has been kept within WP3 and WP4, not affecting other work packages.

## 5 Utilities

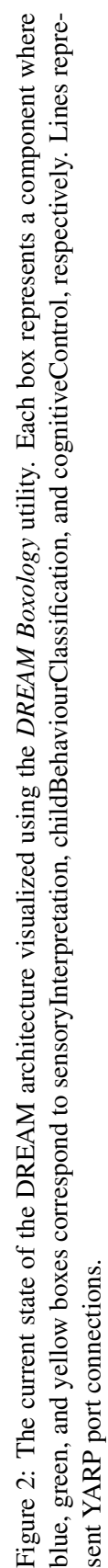
D3.4.2 presents two utility applications, *componentChecker* and *yarpGenerator*. During the last year, four additional utilities have been developed supporting both developers and therapists in working with and maintaining the DREAM system.

### 5.1 DREAM Boxology

The complete DREAM system comprises a large set of interconnected components, following Component-Based Software Engineering (CBSE) and the YARP component-port-connector model, c.f., D3.1. Each specific configuration of connected components is referred to as an *application* and is defined as an XML-file specifying all components and connections constituting the specific system configuration.

As the number of components and connections grew, maintaining different application files by manually editing XML became increasingly difficult. In order to handle this problem and to better visualize the system architecture, *DREAM Boxology* was created. This utility allows instantiation and re-wiring of YARP components using a box-and-arrow interface. When a specific system configuration has been created using *DREAM Boxology*, the corresponding XML application file can automatically be generated for execution using YARP.

A visualization generated using *DREAM Boxology* is presented in Fig. 2. While the labels for components and connections are too small to be readable in this figure, the GUI provides an effective visualization to zoom in and out in order to switch between overview and system details. The GUI also provides direct editing by dragging and dropping new components and connections. This visualization can be seen as a successor to Fig. 1 from D3.1.



## 5.2 Script Generator

Each intervention is defined as a script specifying the robot's actions and expected child behavior. These scripts can be seen as formalizations of the intervention definitions originally presented in D1.1. *Script Generator* is a tool for the developers and therapists to create and modify intervention scripts. It provides a graphical user interface for modifying and creating scripts and has been used to create all scripts active in interventions conducted with ASD children in spring 2017. Please refer to D6.3.3 for further details.

## 5.3 User Data Export

During the integration meeting at UBB (Sec. 2.3), the need to export information logged during interventions was identified. This included information about the child, their performance and engagement during the intervention, and other script-specific aspects of the intervention. This information is stored by the DREAM system in an XML-based user model and text-based log files, not directly readable by the therapists. The *userDataExport* tool extracts relevant information from the system storage formats to a single Excel sheet, in a format requested by the therapists.

## 5.4 User Model Creator

The *userModelCreator* is used by the therapists to create and modify user model XML files for each child involved in the study. The utility automatically updates the list of participants that will be displayed when running the DREAM system, and also allows the therapist to specify diagnosis information for each child, visualized during interactions.

## References

- [1] DREAM. Component reference manual. [https://dreamproject.aldebaran.com/projects/dream/wiki/Component\\_Reference\\_Manual](https://dreamproject.aldebaran.com/projects/dream/wiki/Component_Reference_Manual), 2017.
- [2] DREAM. Integration status. [https://dreamproject.aldebaran.com/projects/dream/wiki/Integration\\_Status](https://dreamproject.aldebaran.com/projects/dream/wiki/Integration_Status), 2017.
- [3] DREAM. Redmine issue tracker. <https://dreamproject.aldebaran.com/projects/dream/issues>, 2017.
- [4] DREAM. Roll-out plan. [https://dreamproject.aldebaran.com/projects/dream/wiki/Software\\_Rollout\\_Plan](https://dreamproject.aldebaran.com/projects/dream/wiki/Software_Rollout_Plan), 2017.
- [5] DREAM. Wiki. <https://dreamproject.aldebaran.com/projects/dream/wiki>, 2017.
- [6] D. Vernon, E. Billing, P. Hemeren, S. Thill, and T. Ziemke. An Architecture-oriented Approach to System Integration in Collaborative Robotics Research Projects - An Experience Report. *Journal of Software Engineering for Robotics*, 6(1):15–32, 2015.



**Appendix A Notes from the DREAM Developers Meeting**

**Appendix B Notes from the UBB integration week**

**Appendix C Integration Reports**



## Notes from the DREAM Developers Meeting

Version 3.0

7-8 June

University of Skövde

### Participants

Erik Billing, Haibin Cai, Hoang-Long Cao, Cristina Costescu, Karl Drejing, Pablo Gómez, Paul Hemeren, Rob Homewood, Bangli Liu, Honghai Liu, James Kennedy, Alexandre Maxel, Emmanuel Senft, David Vernon.

### 1. Matters arising from the ad hoc developers meeting 13 May

For the benefit of those not present at the ad hoc developers meeting, we summarized and discussed the main issues.

#### 1.1 Structure of release directory

The release directory will be reorganized, as follows.

- Two additional subdirectories will be introduced: *tools* and *test*
- The complete DREAM system comprises 17<sup>1</sup> components as specified in the rollout plan. These 17 components will be placed in the existing *components* subdirectory. This subdirectory will have a flat structure, i.e. components will not be organized into sub-subdirectories according to system architecture subsystem (sensoryInterpretation, childBehaviourClassification, and cognitiveControl).
- The *components* subdirectory will also be used for library sources, e.g. *guiUtilities*, that need to be compiled and linked when creating component executables.
- Utilities such as *yarpGenerator* will be placed in the *tools* subdirectory.
- The *componentChecker* component will also be placed in the *tools* directory and compiled along with the other test components.
- Additional components that are needed for running unit tests and systems tests will be placed in the *test* subdirectory.
- The four placeholder components *sensoryInterpretation*, *childBehaviourClassification*, *cognitiveControl*, and *systemArchitectureGUI* will be moved from the *components* subdirectory to the *test* subdirectory. The *sensoryInterpretationLogger* component will be placed here too.
- The *cmake* for *release* should compile all components in *components*, *test*, and *tools* subdirectories. Some utilities in *tools* may be precompiled or compiled separately.

These changes will be implemented over the coming weeks.

---

<sup>1</sup> The number of components was revised upwards to 19 later in the meeting to reflect the need for a new *sandtrayServer* component, to be developed by Plymouth University, and a new *identifyVoice* component, to be developed by Aldebaran. However, the number of components in the run-time system remains at 17 in the initial version: the *improveAssessment* component will probably be removed from the *childBehaviourClassification* subsystem and *mapFromPerceptsToBehaviour* will be used in a later version of the run-time system. Both of these are still relevant for D2.2.

## 1.2 Guidelines for naming ports

We will adopt a new convention for naming ports to avoid name pollution but still provide unique port names that as short as possible.

- The new standard format will be `/<component_name>/<port_name>:<ilo>`
- This means all the system architecture port names have to be renamed since they have adopted the format `/<subsystem_name>/<port_name>`
- The subsystem names will only be used for logical grouping of components, not for physical naming and organization in directories.
- The `<component_name>` part of the port names will be derived from the 17 components in the new system architecture that was developed during the meeting (see below).
- The `<port_name>` part of the port names often reflects the software primitives defined in the system architecture.
- Since ports are used to connect two components, there will always be at least two ports with the same `<port_name>`, one for the producer and one for the consumer of data on the connector.
  - The producer writes to the port and so its port has an `<:o>` appended.
  - The consumer reads from the port and so has an `<:i>` appended.

For example, the `getEyes(eyeL_x, y, z, eyeR_x, y, z)` primitive is implemented in the `sensoryAnalysis` component and the information it produces is read, for instance, by the `mapFromPerceptsToBehaviour` component. Consequently, the `sensoryAnalysis` component implements this port

```
/sensoryAnalysis/getEyes:o
```

and the `mapFromPerceptsToBehaviour` component implements this port

```
/mapFromPerceptsToBehaviour/getEyes:i
```

- However, some primitives read parameter values before producing data. In this case, the component implementing the primitive has both an `<:i>` port and a `<:o>` port: the `<:i>` port is used to read the parameter values and the `<:o>` is used to write out the data produced by the primitive. Similarly, the component that consumes the data also has an `<:i>` port and a `<:o>` port: the `<:o>` port is used to write out the parameter values and the `<:i>` is used to read the data produced by the primitive.

For example, the `getEyeGaze(eye, x, y, z)` primitive is implemented in the `sensoryAnalysis` component and the information it produces is read, for instance, by the `mapFromPerceptsToBehaviour` component. It reads the `<eye>` parameter and writes the `<x, y, z>` data. Consequently, the `sensoryAnalysis` component implements these two ports

```
/sensoryAnalysis/getEyeGaze:i // read the <eye> parameter
/sensoryAnalysis/getEyeGaze:o // write the <x, y, z> data
```

and the `mapFromPerceptsToBehaviour` component implements these two ports

```

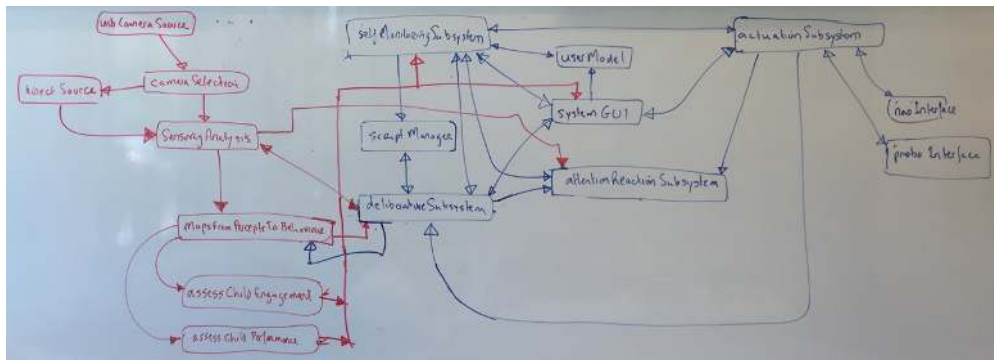
/mapFromPerceptsToBehaviour/getEyeGaze:o // write the <eye> parameter
/mapFromPerceptsToBehaviour/getEyeGaze:i // read the <x, y, z> data

```

While this naming convention should now be clear, some work may remain to ensure that parameter setting messages and subsequently produced data are in synch.

### 1.3 System Architecture Diagram

A complete system architecture diagram with all 17 components was created, showing consolidated connections between components rather than individual ports-to-port connections.



This architecture has to be updated to include a new sandtrayServer component, to be developed by Plymouth University, and a new identifyVoice component, to be developed by Aldebaran.

The connection to and from the three components associated with the childBehaviourClassification subsystem – mapFromPerceptsToBehaviour, getChildPerformance, getChildEngagement – need to be rationalized to reflect the amendments of their specifications agreed later in the meeting (see below) and the removal of mapFromPerceptsToBehaviour in the current version of the run-time system.

A second version showing stubs and drivers used for unit tests and system tests should be produced in due course.

## 2. Walkthrough of the installation of the yarpGenerator

The resources for the yarpGenerator utility is currently located in

C:\DREAM\working\PLYM\yarpGenerator\

These resources, i.e. all subdirectories and contents, should be relocated to

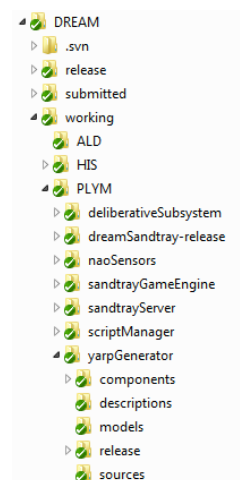
C:\DREAM\release\tools\yarpGenerator

The executable is located in

C:\DREAM\working\PLYM\yarpGenerator\release

Support documentation is located in

C:\DREAM Documents\techreports\WP6\2015-05 yarpGenerator documentation

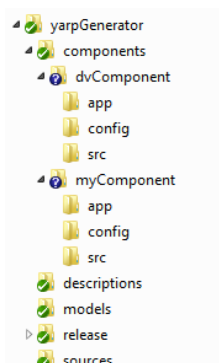




### 3. Walkthrough of the development of a simple component with yarpGenerator

Components that are generated by yarpGenerator are placed in ~\yarpGenerator\components

For example:



### 4. Submission of a real component for integration, ideally from the rollout plan

Leading up to and during the meeting, the University of Portsmouth submitted four components for integration.

### 5. Walkthrough of the integration procedure, including use of componentChecker

The componentChecker utility provides a first level check on compliance with the revised standards set out on the wiki. The wiki checklist remains the definitive validation test.

We agreed that if a component fails any of the checks it will be referred back to the developer for action and removed from the submitted subdirectory. In general, the person responsible for integration tests (Rob Homewood) will not attempt to fix any errors that are spotted during the integration procedure.

Erik Billing agreed to amend the component checker so that it emits error messages that refer to the checklist numbering on the wiki.

### 6. Review of “integration readiness” of components from the rollout plan

This item led to a discussion of the timeline for development. While most developers had nominated submission dates in the rollout plan that are related to the Description of Work and the schedule for deliverables, it quickly became clear that these dates won't work if we are to have a system ready in time for the next WP2 test interventions and for a demonstration for the next review.

We agreed the following milestones.

- 18 November 2016: all 17 components will have been submitted for integration.
- Early December: integration meeting in Plymouth.
- 1 January: 1<sup>st</sup> version of complete system operational.
- 1 February: UBB tests begin.
- 1 April: review demo ready.

This is a necessarily ambitious schedule but it is one we have to meet if the project is to succeed.

## 7. Modifications to the intervention tasks and implications for WPs 4, 5, and 6

The aim of this agenda item was to determine whether intervention tasks needed to be modified to ensure that the DREAM system can be completed in time for the D2.2 tests scheduled for February 2017.

This was one of the most important discussions because it brought together several other items on the agenda.

Beginning at the start of the second day, we walked through the interventions as defined in Deliverable D1.1, beginning with joint attention, to check that all the necessary sensory and motor primitives would be available. This was done using Table 3 (D1.2) *Correspondence between baseline robot actions and action primitives* and Table 3 (D1.3): *Correspondence between robot perceptions and perception primitives*, linking these correspondences back to the intervention definitions in Section 3 of D1.1.

After walking through several interventions it became clear that all of the necessary primitives from WP4 and WP6 have either been implemented or are at an advanced stage of development and that these will be ready for the complete system integration on November 18. At the same time, some changes to the intervention specification (as originally documented in D1.1) are needed to ensure that they reflect what is actually required of the cognitiveControl subsystem. For example, some steps in some interventions are no longer needed and the joint attention intervention will use the sandtray exclusively and there is no need to use physical objects.

However, it was unclear how the WP5 primitives in the rollout plan would be used and whether they provided any useful information for the run-time system. This is primarily due to the fact that the WP5 primitives were not referred to when writing Deliverables D1.1, D1.2, and D1.3. They were defined in Deliverable D3.1 on the system architecture but the links to the WP6 cognitiveControl subsystem were never defined. The links to WP4 sensoryInterpretation subsystem were defined in a rather shallow manner, essentially saying little more than all WP4 data was potentially of use. Furthermore, there is also a lack of clarity regarding the link between the work in T4.4 and the work in WP5.

### 7.1 getChildBehaviour primitive vs. identifyTrajectory primitive

It emerged that there is a significant difference between what functionality was actually needed for the execution of the interventions from work in WP5 and what had been planned. It also emerged that there is an overlap between the functionality being provided by WP4 through the identifyTrajectory primitive and the behaviour classification functionality to be provided by WP5 through the getChildBehaviour primitive.

The situation regarding WP5 prior to the meeting was that WP5 would implement three primitives:

```
getChildBehaviour()  
getChildMotivation()  
getChildPerformance()
```

These are to be implemented in three components, respectively:

```
mapFromPerceptsToBehaviour  
assessChildEngagement  
assessChildPerformance
```

However, there seems to have been some confusion regarding a difference between the “behaviours” that are identified in the WP5 getChildBehaviour() primitive and the “actions” (or gestures) that actually need to be recognized in the execution of the intervention scripts. The behaviours seem to be more aligned with the work in WP2 (Deliverable D2.2) than what is required for WP6 cognitiveControl. It became evident that the functionality provided by the WP4 primitive identifyTrajectory() is much closer to what is needed in WP6 and the University of Portsmouth kindly agreed to expand the functionality of this primitive to cater for the ten actions that need to be recognized during the interventions, as follows.

1. Hand wave
2. Hands covering eyes
3. Hands on head
4. Fly (arms extended horizontally)
5. Drive car (straight horizontal hand gesture)
6. Drink / Smell (straight vertical hand gesture, bringing hand to the mouth)
7. New complex trajectory number 1
8. New complex trajectory number 2
9. New complex trajectory number 3
10. New complex trajectory number 4

UBB will provide video examples each trajectory, i.e. gesture, so that PORT can generate prototypical classes.

As a consequence, identifyTrajectory() effectively replaces getChildBehaviour() in the run-time system. The latter, and its corresponding component mapFromPerceptsToBehaviour, will still be needed for work related to WP2 and it will probably be part of the runtime system at a later stage. The information derived from identifyTrajectory and other WP4 primitives will now be captured in a revised version of the getChildPerformance primitive implemented in the assessChildPerformance component. This information will be used in the WP6 cognitiveControl subsystem.

## 7.2 identifyTrajectory primitive

The specification of identifyTrajectory() has to change slightly to accommodate its new more general use. Specifically, the format for the inputs and outputs need to be adapted. The current specification involves a vector of doubles comprising a sequence of 4-tuples (x, y, z, t) and writing a vector with one double indicating the identity of the trajectory:

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

The new specification must allow for the input to be a sequence of skeleton configurations and the output should be a vector of 10 doubles.

Element  $x$  of the output vector identifies the probability that the input is trajectory (i.e. action / gesture) number  $x$ . If it proves difficult to compute a probability value, element  $i$  of the output vector will be set to 1 if the input is trajectory (i.e. action / gesture) number  $i$ , otherwise it will be set to 0.

At the meeting, we did not decide on a specific format for the input sequence of skeleton configurations that are used to identify the trajectory. None of the sensoryAnalysis primitives currently expose the skeleton data and it seems wasteful to collect this data in the childBehaviourClassification subsystem (specifically in the assessChildPerformance component; see below) only to send it back to sensoryAnalysis.

*Instead, it is proposed here that the input to identifyTrajectory is simply a trigger to start the recognition process. All the skeleton data can then be processed locally in the sensoryAnalysis component. This proposal has yet to be agreed.*

```
identifyTrajectory(startTrigger, vectorOfActions)
/sensoryAnalysis/identifyTrajectory:i BufferedPort<int>
/sensoryAnalysis/identifyTrajectory:o BufferedPort<VectorOf<double>>
```

### 7.3 Interface between childBehaviourClassification and cognitiveControl subsystems

The manner in which the childBehaviourClassification subsystem interfaces with the cognitiveControl subsystem was also discussed and amended. This resulted in changes to the getChildMotivation and getChildBehaviour primitives and the assessChildEngagement and assessChildBehaviour components, respectively.

### 7.4 getChildPerformance primitive and assessChildPerformance component

The getChildPerformance primitive, implemented in the assessChildPerformance component, is currently defined as follows (see Deliverable D3.1).

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

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.

It was agreed to simplify this functionality significantly in order to provide the information required by the cognitiveControl subsystem, as follows.

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

The getChildPerformance() primitive determines whether or not the child successfully performs a required action at a given point in the intervention.

These actions comprise the ten already mentioned above:

1. Hand wave
2. Hand covering eyes
3. Hands on head
4. Fly (arms extended horizontally gesture?)
5. Drive car (straight horizontal hand gesture)
6. Drink / Smell (straight vertical hand gesture, bringing hand to the mouth)
7. New complex trajectory number 1
8. New complex trajectory number 2
9. New complex trajectory number 3
10. New complex trajectory number 4

In addition, it includes the following actions (to be confirmed):

11. Look left (in joint attention intervention)
12. Look right (in joint attention intervention)

13. Point left (in joint attention and turn-taking interventions)
14. Point right (in joint attention and turn-taking interventions)
15. No movement (i.e. waiting in turn-taking intervention)
16. Child speaks

Whether or not the child speaks will be determined using the identifyVoice primitive to be developed by Aldebaran (see below).

The expected action to be performed is determined from the information provided by the getInterventionStatus primitive (see below).

Ideally, the degree\_of\_performance output identifies an estimate of the degree to which the action has been performed by the child, bounded by 0 and 1. If this is not possible, then the output will be either 0 or 1 to indicate non-performance and perfect performance, respectively.

### 7.5 getChildMotivation / getChildEngagement primitive

The getChildMotivation primitive, implemented in the assessChildEngagement component, is currently defined as follows (see Deliverable D3.1).

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.

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

It was agreed to simplify very significantly the functionality of this primitive (now to be renamed getChildEngagement). It will be replaced by a primitive defined as follows.

```
getChildEngagement(engagement_flag)
/getChildEngagement/engagementFlag:o BufferedPort<VectorOf<double>>
```

It will be computed on the basis of three binary variables indicating whether or not:

- The child is smiling (derived from identifyFaceExpression)
- The child is making eye contact with robot (derived from checkMutualGaze)
- The child is positioned in front of the robot (derived from getBody)

The output is a simple binary flag derived from a truth table that has yet to be determined with Cristina's help.

Smiling	Eye Contact	In Front	Engaged
F	F	F	?
F	F	T	?
F	T	F	?
F	T	T	?
T	F	F	?
T	F	T	?
T	T	F	?
T	T	T	?

## 7.6 improveAssessment component

A fourth component – improveAssessment – does not implement any primitive and apparently has no direct role to play in the system architecture. This was noticed after the meeting. We need to check to see if this component should be removed from the rollout plan.

## 7.7 getInterventionStatus primitive

The getInterventionStatus primitive is currently defined as follows.

```
getInterventionStatus(interventionDescriptor, stateDescriptor,
cognitiveModeDescriptor) /cognitiveControl/getInterventionStatus:o
BufferedPort<VectorOf<int>>
```

The exact format of the information to be exposed by the deliberativeSubsystem component in the cognitiveControl subsystem has yet to be defined. This information is used in the assessChildPerformance component to determine what action the child should be attempting to perform, based on the intervention definition.

Furthermore, the exact form of the intervention definition has yet to be defined so that the intervention status data is sufficient to extract the required information.

## 7.8 Joint Attention Intervention

It was decided that the joint attention intervention would not use objects and would instead use the sandtray.

## 8. Walkthrough of cognitiveControl subsystem

Having already walked through the operation of the full system architecture, it was agreed that it was no further discussion of this item was necessary.

## 9. Walkthrough of operation of a system primitive

Having already walked through the guidelines for naming ports (see above) and manner in which these ports would expose the data produced by the perception and action primitives in D1.2 and D1.3, it was agreed that further discussion of this item was unnecessary.

## 10. Implementation of interventions under supervised autonomy

Having already walked through the interventions, it was agreed that it was no further discussion of this item was necessary at this time.

## 11. Walkthrough of video annotation and implications for performance measurement

This item was dealt with off-line in a separate meeting.

## 12. Discussion of the rollout schedule

Developers agreed to update the submission dates in the wiki software rollout plan to reflect the new agreed timeline (see above).

## 13. Discussion of new WP8 development plans

Several issues were discussed. For the purposes of immediate development, the main decision was that Aldebaran will implement the identifyVoice primitive as a standalone component identifyVoice.



#### 14. Actions To Be Taken (responsible person in bold)

1. Implement the new directory structure in release, relocate the components as necessary, and update the Cmake files [**Erik**].
2. Copy yarpGenerator resources to C:\DREAM\release\tools\yarpGenerator [**Erik**].
3. Add information on yarpGenerator to the wiki [**David**].
4. Add componentChecker to integration procedure and amend componentChecker to emit error messages that refer to the checklist numbering on the wiki [**Erik**].
5. Implement the new port naming convention for all 17 components in the wiki software rollout plan (see WP6 components for examples) [**All developers**].
6. Amend the component delivery dates for all 17 components in the wiki software rollout plan to align them with the November 18 deadline (see WP6 components for examples) [**All developers**].
7. Update the port name convention in the integration procedure [**Erik**].
8. Add a definition of the sandtrayServer component to the wiki software rollout plan [**James**].
9. Add a definition of the identifyVoice component to the wiki rollout plan [**Alexandre**].
10. Draw the system architecture with all 17 components [**James**].
  - a. Update the connections to reflect the changed specifications for the child behaviour classification subsystem, reflecting the removal of the mapFromPerceptsToBehaviour component and the operation of the revised assessChildPerformance and assessChildEngagement components.
  - b. Update for the new sandtray component. Update for the new identifyVoice component.
11. Define port protocols by assigning parameter values for all ports in the wiki software rollout plan [**All developers**].
12. Agree the new specification of the identifyTrajectory primitive: input is simply a trigger to start the recognition process; output is a vector of 10 action/gesture probabilities: either 0 or 1, or ideally a number in the range 0-1 [Haibin, Honghai, **Serge**, Yinfeng].
13. Provide video examples each of the 10 actions, including the four new complex actions for the identifyTrajectory primitive [**Cristina**].
14. Update specification of getChildPerformance primitive and assessChildPerformance component on the wiki software rollout plan [**Serge**].
15. Update specification of getChildMotivation primitive (renaming to getChildEngagement) and assessChildEngagement component on the wiki software rollout plan [**Serge**].
16. Removed improveAssessment component from the wiki software rollout plan [**Serge**].



17. Define the exact format of the information to be exposed by the deliberativeSubsystem component in the cognitiveControl subsystem, i.e. the form and content of the information produced by the getInterventionStatus primitive [James, **Emmanuel**].
18. Define the exact form of the intervention definition so that the intervention status data is sufficient to extract any required information, e.g. by the assessChildPerformance component [James, **Emmanuel**, Serge]
19. Agree dates for the integration developers meeting in the University of Plymouth in December [**All developers**].
20. Update specifications of all interventions to ensure that they reflect what is actually required [**Cristina**].





## Notes from the UBB integration week

James Kennedy

February 13-17, 2017

### 1 System state

- The system is integrated with components from WP4, WP5 and WP6 (PORT, HIS, PLYM, VUB), and it runs.
- Cristina confirmed that it is smooth enough for the needs of the therapists to start the study. However, it is not perfect and has some bugs that we should aim to fix (see further below in this message).
- Unfortunately, no one from the integration team could attend, so I acted as the integrator for the week. During this week, the full QA procedure was temporarily set aside with the aim of getting the system running. This means that some unit tests are outdated as system components were updated, and these unit tests may not yet work. The software rollout plan on the wiki is also outdated as some components and ports have changed. There will not be SVN history of submission for many changes as I either integrated directly from working, or modified code directly in release, but the log messages should be clear enough.
- Some code may be newer in release than in working; before doing further development in working, it is your responsibility to ensure that any changes in release are ported to working (otherwise we may be undoing bug fixes on re-submission).

### 2 Changed software versions

- We are now using Windows 10 as the PCs in Romania had automatically updated, and given that most of us have been using Windows 10 without problems, it was quicker to stick with it than roll back to Windows 7.
- The Romanian TTS was not developed for the project agreed NAO version (v2.1.2.17) by ALD/SBRE. This made the robot take a long time to boot (50+ mins) and have connection issues. No fix was forthcoming, so we therefore had to change the NAO robot version and corresponding C++ SDK to v2.1.4.13 (if you upgrade, remember that zlib1.dll still needs to

be replaced). The wiki page<sup>1</sup> also needs updating as v2.1.2.17 is regularly stated as the required version.

- CMake has been changed to version 3.3.2 as the old version caused problems with WP4 stability. Wiki instructions have already been updated for this.
- YARP, FLTK, and VC compiler versions remain as originally specified.

### 3 Known bugs/work to do:

- The system will occasionally freeze in the GUI. The buttons still call code, but nothing updates visually. This is infrequent (about once a day) and we cannot find a cause. We may have to live with this one.
- The system can get stuck within the WP6 action suggestion loop (again, this is infrequent). This is an issue with WP6 having many ports that need to do something and also forward information. If the read does not complete, then the callback port can get blocked. We have an idea for a solution, but Erik has also asked on robotology for further inspiration (c.f., <https://github.com/robotology/QA/issues/188>).
- WP6: Logging needs to be refined to ensure we are capturing the performance data needed (and to make the writing of the logs thread safe - this is likely what currently causes the issue above). I've discussed a possible solution with Erik.
- WP6: The logs need to be exported into Excel for use by the therapists (I'm currently writing a program to handle this).
- System start/exit instructions for the therapists (I have made a start on this already).

---

<sup>1</sup>Wiki page for robot setup: [https://dreamproject.aldebaran.com/projects/dream/wiki/Nao\\_software](https://dreamproject.aldebaran.com/projects/dream/wiki/Nao_software)

## D3.4.3 DREAM Integration Report

University of Skövde

April 6, 2017

### 1 Introduction

The following is a selection of emails chronicling the integration process for two components, *systemGUI* (WP6) and *assessChildPerformance* (WP5) integrated during 2016.

### 2 Component Integration Cases

#### 2.1 systemGUI

Hoang-Long Cao  
C:\DREAM\submitted\VUB\systemGUI  
To: Dream Integration Team, Cc: James Kennedy, Emmanuel Senft, Pablo Gómez  
18 November 2016 at 16:53  
Details HC

New contact info found in this email: Hoang-Long Cao hoancao@vub.ac.be add...

Dear DREAM Integrators,

I would like to submit the systemGUI component.

The name of the component being submitted for integration  
>> systemGUI

The name(s) of any stub or driver components used in the unit test  
>> test/systemGUIDriver


The names of any system architecture placeholder components used in the unit test  
>> N/A

The names of any files access by any of the components (including the component being integrated, the driver and stub components)  
>> systemGUI/src/buttonsLibrary.h  
>> systemGUI/config/data/buttons.dream config file  
>> systemGUI/config/icons config folder with icon files  
>> systemGUI/config/logo.png and systemGUI/config/dream-eu-logo.png files  
>> include/scriptManagerLibrary.h, include/userListLibrary.h, include/userDataLibrary.h, include/scriptMessages.h, include/actionMessages.h, and include/actionOrigin.h shared headers  
>> userModel/config/userdata/user\_1.dream and userModel/config/userdata/userlist.dream config files  
>> scriptManager/config/scripts.dream config file

Kind regards,  
Long

--  
CAO Hoang-Long  
PhD candidate - Vrije Universiteit Brussel - Belgium  
Member of Brubotics - The Brussels Human Centered Robotic research center  
[T] +32 4 85 60 40 66  
[A] Pleinlaan 2, Room ZW105, 1050, Brussels, Belgium  
[W] <http://www.brubotics.eu>  
[W] <http://mech.vub.ac.be/multibody/members/caohaolang.htm>

Figure 1: The developer submits the component to the repository under the 'submitted' folder and emails the integration with a brief report about the submitted component

Hoang-Long Cao  
 Re: C:\DREAM\submitted\VUB\systemGUI  
 To: Dream Integration Team, Cc: James Kennedy, Emmanuel Senft, Pablo Gómez  
 18 November 2016 at 17:01  
[Details](#) 

Dear DREAM Integrators,

This systemGUI version uses a newer version of scriptManager/config/scripts.dream config file than the current version in the release folder. The new version of scripts.dream will be submitted in C:\DREAM\submitted\PLYM\scriptManager.

Best regards,  
 Long

[See More from Hoang-Long Cao](#)

—  
 CAO Hoang-Long  
 PhD candidate - Vrije Universiteit Brussel - Belgium  
 Member of Brubotics - The Brussels Human Centered Robotic research center  
 [T] +32 4 85 60 40 66  
 [A] Pleinlaan 2, Room ZW105, 1050, Brussels, Belgium  
 [W] <http://www.brubotics.eu>  
 [W] <http://mech.vub.ac.be/multibody/members/caohoanglong.htm>

Figure 2: The developer submits a second email highlighting a change to one of the dependencies necessary to test the component

Dream Integration Team  
 Re: C:\DREAM\submitted\VUB\systemGUI  
 To: Hoang-Long Cao, Cc: James Kennedy, Emmanuel Senft, Pablo Gómez  
 1 December 2016 at 19:36  
[Details](#) 

Hi Long,

The systemGUI integration has failed on two minor points. The first I have fixed for you but the second is outside of my range. It should be quite quick to solve however and then the integration will be complete.

The first issue is with point 9. Of the integration checklist:

```
3.1 [ ] /** FILE <componentName>.h <one line to identify the nature of the file>
```

While the component's name is there, the line identifying the nature of the file is missing. This is the one that I have fixed for you (indicated by the orange tick on the report attached), but I did notice that this same mistake has been made on the selfMonitoringSubsystem component so it may just be something to look out for in future components. Its an easy one to fix, if you take a look at the protoComponent you should be able to copy, paste and modify it from there.

The second issue I unfortunately wasn't able to resolve for you but it shouldn't take too long to do:

```
4.2.4 [ ] Do the instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration Load file are altered?
```


The readme file was actually quite nicely laid out, it just lacked this one piece. If you could please update the file we will be ready to commit it to the release folder. This is also an issue for the selfMonitoringSubsystem component too (I will test this with the actionMessages.h file and get back to you with the integration report for this too).

Many Thanks,  
 Floo

  
 SystemGUI  
 Integre...list.pdf

[See More from Hoang-Long Cao](#)

Figure 3: The Integrator responds with an integration report (Sec. 3.1) and a list of issues required be fixed before the component can be integrated

Hoang-Long Cao  
 Re: C:\DREAM\submitted\VUB\systemGUI  
 To: Dream Integration Team, Cc: James Kennedy, Emmanuel Senft, Pablo Gómez  
 2 December 2016 at 10:18  
[Details](#) 

Hello Rob,

Thank you. I have updated the systemGUI.h and README.txt files.  
 Hope that the component can now pass all requirements.

```
9. [] /** @file <componentName>.h <one line to identify the nature of the file>
```

The minor issue comes from the yarpGenerator and it can be fixed easily. Some components currently in the release and submitted folders have the same problems.

Best,  
 Long

[See More from Dream Integration Team](#)

..

CAO Hoang-Long  
 PhD candidate - Vrije Universiteit Brussel - Belgium  
 Member of Brubotics - The Brussels Human Centered Robotic research center  
 [T] +32 4 85 60 40 66  
 [A] Pleinlaan 2, Room ZW105, 1050, Brussels, Belgium  
 [W] <http://www.brubotics.eu>  
 [W] <http://mech.vub.ac.be/multibody/members/caohoanglong.htm>

Figure 4: The developer responds with confirmation that the necessary changes have been made to the component and reports back on an issue related to the component generator tool

Dream Integration Team  
 Re: C:\DREAM\submitted\VUB\systemGUI  
 To: Hoang-Long Cao, Cc: James Kennedy, Emmanuel Senft, Pablo Gómez  
 1 December 2016 at 19:36  
[Details](#) 

Hi Long,

The systemGUI integration has failed on two minor points. The first I have fixed for you but the second is outside of my range. It should be quite quick to solve however and then the integration will be complete.

The first issue is with point 9. Of the integration checklist:

```
9. [] /** @file <componentName>.h <one line to identify the nature of the file>
```

While the component's name is there, the line identifying the nature of the file is missing. This is the one that I have fixed for you (indicated by the orange tick on the report attached), but I did notice that this same mistake has been made on the selfMonitoringSubsystem component so it may just be something to look out for in future components. Its an easy one to fix, if you take a look at the protoComponent you should be able to copy, paste and modify it from there.

The second issue I unfortunately wasn't able to resolve for you but it shouldn't take too long to do:

```
42 4. [] Do the instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration (.ini) file are altered?
```

The readme file was actually quite nicely laid out, it just lacked this one piece. If you could please update the file we will be ready to commit it to the release folder. This is also an issue for the selfMonitoringSubsystem component too (I will test this with the actionMessages.h file and get back to you with the integration report for this too).

Many Thanks,  
 Rob

  
 SystemGUI  
 Integra...list.pdf

[See More from Hoang-Long Cao](#)

Figure 5: The Integrator responds letting the developer know that the systemGUI component now meets all the integration criteria and has been successfully integrated. The integrator attaches an integration report (Sec. 3.2)

## 2.2 assessChildPerformance

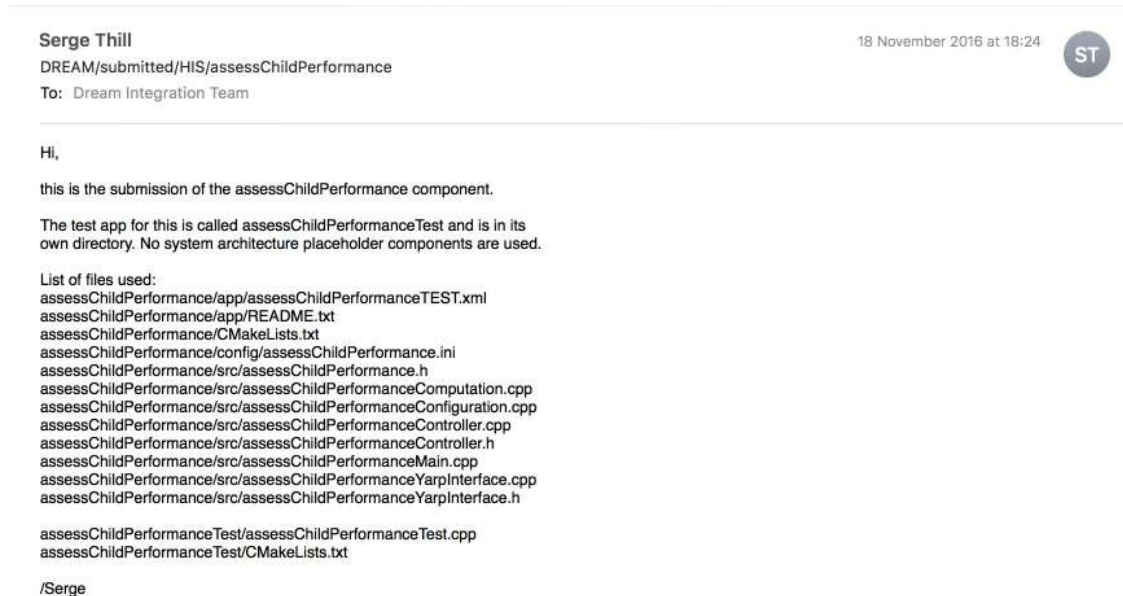


Figure 6: The developer submits the component to the repository under the 'submitted' folder and emails the integration with a brief report about the submitted component

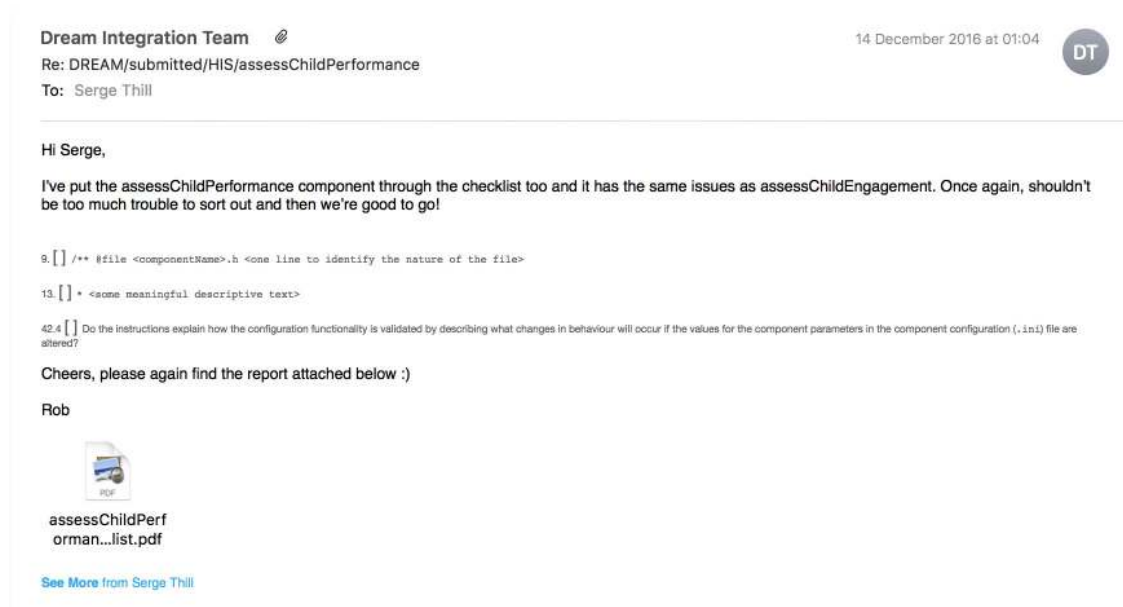


Figure 7: The Integrator responds with an integration report (Sec. 3.3) and a list of issues required be fixed before the component can be integrated

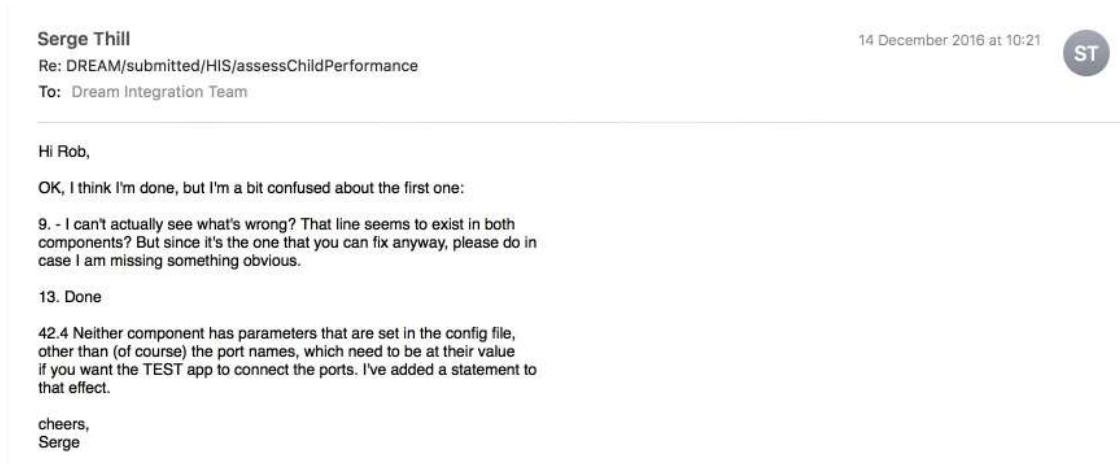


Figure 8: The developer responds with confirmation that the necessary changes have been made to the component and asks for clarity on point 9

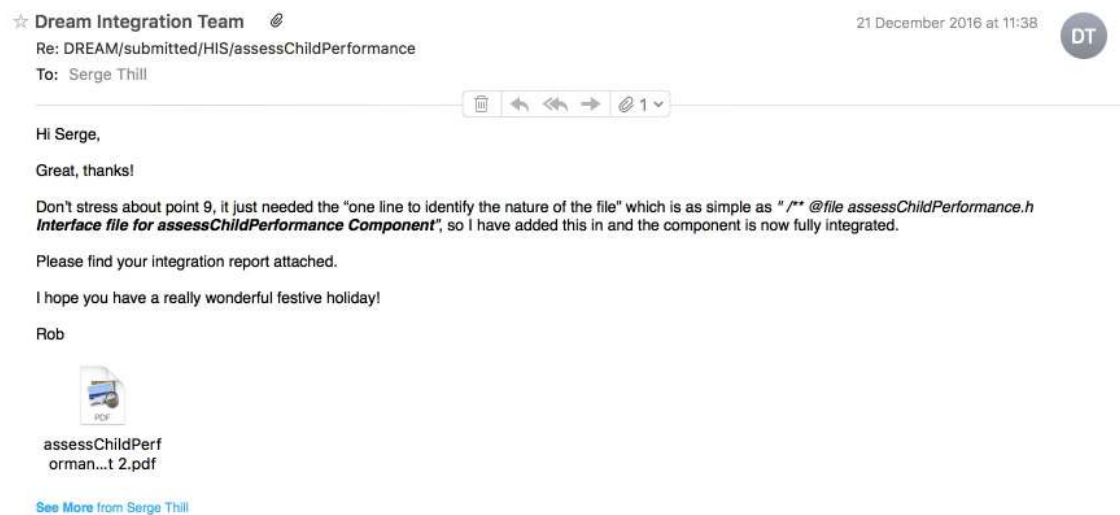


Figure 9: The Integrator responds letting the developer know that they have fixed the minor issue with point 9 and the assessChildPerformance component now meets all the integration criteria and has been successfully integrated. The integrator attaches an integration report (Sec. 3.4)



## 3 Integration reports

### 3.1 systemGUI Report - Incomplete

#### INTEGRATION CHECKLIST

##### FILES AND DIRECTORIES

1. ☒ Are files for a single component stored in a directory with the same name as the component? For the purposes of this checklist, we will use `<componentName>` to stand for the component name. For example, if we were integrating a component named `myComponent`, then we would use `myComponent` everywhere we find `<componentName>` in the following. Note that the leading letter is in lowercase.

2. ☒ Does this directory have three sub-directories: `src`, `app`, and `config`?

Does the `src` directory contain one header file and three source files, named as follows.

3.1 ☒ `<componentName>.h`

3.2 ☒ `<componentName>Main.cpp`

3.3 ☒ `<componentName>Configuration.cpp`

3.4 ☒ `<componentName>Computation.cpp`

4. ☒ Does the `app` directory contain an XML application file named after the component but with the suffix `TEST`: `<componentName>TEST.xml`?

5. ☒ Does the `app` directory contain a `README.txt`.

6. ☒ Does the `config` directory contain a `<componentName>.ini` configuration file?

7. ☒ Does the configuration file contain the key-value pairs that set the component parameters?

8. ☒ Is each key-value pair written on a separate line?

Note that the instructions in the `README.txt` file should identify all system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`).

##### INTERNAL SOURCE CODE DOCUMENTATION

Does the `<componentName>.h` file contain a documentation comment with the following sections and accompanying text:

9. ☒ `/** @file <componentName>.h <one line to identify the nature of the file>`

10. ☒ `* <version information>`

11. ☒ `* <date>`

12. ☒ `* \section component_description Component Description`

13. ☒ `* <some meaningful descriptive text>`

14. ☒ `* \section lib_sec Libraries`

15. ☒ `* \section parameters_sec Parameters`

16. ☒ `* <b>Command-line Parameters </b>`

17. ☒ `* <b>Configuration File Parameters </b>`

18. ☒ `* \section portsa_sec Ports Accessed`

19. ☒ `* \section portsc_sec Ports Created`

20. ☒ `* <b>Input ports</b>`

21. ☒ `* <b>Output ports</b>`

Component: SystemGUI

Integrator: Rob Homewood

Date: 01/12/16



22. [✓] \* <b>Port types </b>
23. [✓] \* \section in\_files\_sec Input Data Files
24. [✓] \* \section out\_data\_sec Output Data Files
25. [✓] \* \section conf\_file\_sec Configuration Files
26. [✓] \* \section example\_sec Example Instantiation of the Component
27. [✓] \* \author

\* <forename> <surname>

Do all source files contain a block comment that gives the copyright notice, as follows.

```
/*
 * Copyright (C) 2014 DREAM Consortium
 * FP7 Project 611391 co-funded by the European Commission
 *
 * Author: <name of author>, <author institute>
 * Email: <preferred email address>
 * Website: www.dream2022.eu
 *
 * This program comes with ABSOLUTELY NO WARRANTY.
 */
```

- 28.1 [✓] <componentName>.h
- 28.2 [✓] <componentName>Main.cpp
- 28.3 [✓] <componentName>Configuration.cpp
- 28.4 [✓] <componentName>Computation.cpp

#### COMPONENT FUNCTIONALITY

29. [✓] Does <componentName>.h contain a declaration of a class derived from `yarp::os::RFModule`?

```
class <ComponentName> : public RFModule {} // first char in uppercase
```

30. [✓] Is a ResourceFinder class, e.g. `ResourceFinder rf`, instantiated in <componentName>Main.cpp?
31. [✓] Does the component set the default configuration filename, named after the component with a .ini extension, in <componentName>Main.cpp?

```
rf.setDefaultConfigFile(<"<componentName>.ini">);
```

32. [✓] Does the component set the default path (context) in <componentName>Main.cpp?

```
rf.setDefaultContext(<"components/<componentName>/config">);
```

33. [✓] Does the component read all its key-value parameters from either a <componentName>.ini configuration file or from the list of command line arguments using the ResourceFinder `check()` method, called from within the `configure()` method in <componentName>Configuration.cpp?

```
<parameterValue> = rf.check(<"key">, // parameter key
                          Value(<number>), // default value
                          "key value {int}", asInt()); // key value type
```

34. [✓] Does the component allow the port names to be set and overridden using the port name key-value parameters in the <componentName>.ini configuration file?
35. [✓] Do all port names have a leading /?
36. [✓] Do all input and output port names have a trailing : or :o, respectively?

37. [✓] Does the component allow the default name of the component to be set and overridden with the `--name` parameter?

```
module_name = yr.check("name",
                      Value("<componentName>"),
                      "module_name (required).assertion()");
setName(module_name, n.attr());
```

38. [ ] **Optional:** Does the component allow commands to be issued on a special port with the same name as the component by overloading the `respond()` method in the resource finder `RModule` class in `<componentName>Configuration.cpp`?

### COMPONENT COMMUNICATION

39. [✓] Are all *input* ports named according to the port naming convention: `/<component_name>/<port_name>;i`

40. [✓] Are all *output* ports named according to the port naming convention: `/<component_name>/<port_name>;o`

### COMPONENT UNIT TESTING

41. [✓] Is a unit test application named `<componentName>TEST.xml` provided in the app directory?

42. [✓] Are unit test instructions provided in a file named `README.txt` in the app directory?

42.1 [✓] Do the instructions identify the system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`)?

42.2 [✓] Do the instructions identify the resources required to run the test, including source (input) and sink (output) data files, driver and stub components, and libraries?

42.3 [✓] Do the instructions explain how the communication and computation functionality are validated by describing the (sink) output data that will be produced from the (source) input data?

42.4 [✗] Do the instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration (`.ini`) file are altered?

42.5 [ ] Do the instructions explain how the coordination functionality is validated by describing what changes in behaviour will occur when commands are issued interactively by the user to the component using the port named after the component itself (**optional**)?

43. [✓] Does the test application launch the component being tested on a YARP run servers called `dream1` using the `<node> </node>` construct?

44. [✓] Does the test application connect the component through its ports to a data source and a data sink (linked either to files or driver/stub components)?

45. [✓] Are the data source and sink file resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

46. [✓] Are the library resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

47. [✓] Are the driver and stub components provided as separate components in a distinct component directory, just like the one being submitted for integration (where necessary and as indicated in the `README.txt` file)?



## 3.2 systemGUI Report - Complete

### INTEGRATION CHECKLIST

#### FILES AND DIRECTORIES

1. ☒ Are files for a single component stored in a directory with the same name as the component? For the purposes of this checklist, we will use `<componentName>` to stand for the component name. For example, if we were integrating a component named `myComponent`, then we would use `myComponent` everywhere we find `<componentName>` in the following. Note that the leading letter is in lowercase.

2. ☒ Does this directory have three sub-directories: `src`, `app`, and `config`?

Does the `src` directory contain one header file and three source files, named as follows.

3.1 ☒ `<componentName>.h`

3.2 ☒ `<componentName>Main.cpp`

3.3 ☒ `<componentName>Configuration.cpp`

3.4 ☒ `<componentName>Computation.cpp`

4. ☒ Does the `app` directory contain an XML application file named after the component but with the suffix `TEST`: `<componentName>TEST.xml`?

5. ☒ Does the `app` directory contain a `README.txt`.

6. ☒ Does the `config` directory contain a `<componentName>.ini` configuration file?

7. ☒ Does the configuration file contain the key-value pairs that set the component parameters?

8. ☒ Is each key-value pair written on a separate line?

Note that the instructions in the `README.txt` file should identify all system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`).

#### INTERNAL SOURCE CODE DOCUMENTATION

Does the `<componentName>.h` file contain a documentation comment with the following sections and accompanying text:

9. ☒ `/** @file <componentName>.h <one line to identify the nature of the file>`

10. ☒ `* <version information>`

11. ☒ `* <date>`

12. ☒ `* \section component_description Component Description`

13. ☒ `* <some meaningful descriptive text>`

14. ☒ `* \section lib_sec Libraries`

15. ☒ `* \section parameters_sec Parameters`

16. ☒ `* <b>Command-line Parameters </b>`

17. ☒ `* <b>Configuration File Parameters </b>`

18. ☒ `* \section portsa_sec Ports Accessed`

19. ☒ `* \section portsc_sec Ports Created`

20. ☒ `* <b>Input ports</b>`

21. ☒ `* <b>Output ports</b>`

Component: SystemGUI

Integrator: Rob Homewood

Date: 06/12/16

22. [✓] \* <b>Port types </b>
23. [✓] \* \section in\_files\_sec Input Data Files
24. [✓] \* \section out\_data\_sec Output Data Files
25. [✓] \* \section conf\_file\_sec Configuration Files
26. [✓] \* \section example\_sec Example Instantiation of the Component
27. [✓] \* \author

\* <forename> <surname>

Do all source files contain a block comment that gives the copyright notice, as follows.

```
/*
 * Copyright (C) 2014 DREAM Consortium
 * FP7 Project 611391 co-funded by the European Commission
 *
 * Author: <name of author>, <author institute>
 * Email: <preferred email address>
 * Website: www.dream2022.eu
 *
 * This program comes with ABSOLUTELY NO WARRANTY.
 */
```

- 28.1 [✓] <componentName>.h
- 28.2 [✓] <componentName>Main.cpp
- 28.3 [✓] <componentName>Configuration.cpp
- 28.4 [✓] <componentName>Computation.cpp

#### COMPONENT FUNCTIONALITY

29. [✓] Does <componentName>.h contain a declaration of a class derived from `yarp::os::RFModule`?

```
class <ComponentName> : public RFModule {} // first char in uppercase
```

30. [✓] Is a ResourceFinder class, e.g. `ResourceFinder rf`, instantiated in <componentName>Main.cpp?
31. [✓] Does the component set the default configuration filename, named after the component with a .ini extension, in <componentName>Main.cpp?

```
rf.setDefaultConfigFile(<"<componentName>.ini">);
```

32. [✓] Does the component set the default path (context) in <componentName>Main.cpp?

```
rf.setDefaultContext(<"components/<componentName>/config">);
```

33. [✓] Does the component read all its key-value parameters from either a <componentName>.ini configuration file or from the list of command line arguments using the ResourceFinder `check()` method, called from within the `configure()` method in <componentName>Configuration.cpp?

```
<parameterValue> = rf.check(<"key">, // parameter key
                          Value(<number>), // default value
                          "key value {int}", asInt()); // key value type
```

34. [✓] Does the component allow the port names to be set and overridden using the port name key-value parameters in the <componentName>.ini configuration file?
35. [✓] Do all port names have a leading /?
36. [✓] Do all input and output port names have a trailing : or :o, respectively?

37. [✓] Does the component allow the default name of the component to be set and overridden with the `--name` parameter?

```
module_name = yr.check("name",
                      Value("<componentName>"),
                      "module_name (required).assertion()");
setName(module_name, n.attr());
```

38. [ ] **Optional:** Does the component allow commands to be issued on a special port with the same name as the component by overloading the `respond()` method in the resource finder `RModule` class in `<componentName>Configuration.cpp`?

### COMPONENT COMMUNICATION

39. [✓] Are all *input* ports named according to the port naming convention: `/<component_name>/<port_name>:i`

40. [✓] Are all *output* ports named according to the port naming convention: `/<component_name>/<port_name>:o`

### COMPONENT UNIT TESTING

41. [✓] Is a unit test application named `<componentName>TEST.xml` provided in the app directory?

42. [✓] Are unit test instructions provided in a file named `README.txt` in the app directory?

42.1 [✓] Do the instructions identify the system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`)?

42.2 [✓] Do the instructions identify the resources required to run the test, including source (input) and sink (output) data files, driver and stub components, and libraries?

42.3 [✓] Do the instructions explain how the communication and computation functionality are validated by describing the (sink) output data that will be produced from the (source) input data?

42.4 [✓] Do the instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration `(.ini)` file are altered?

42.5 [ ] Do the instructions explain how the coordination functionality is validated by describing what changes in behaviour will occur when commands are issued interactively by the user to the component using the port named after the component itself (**optional**)?

43. [✓] Does the test application launch the component being tested on a YARP run servers called `dream1` using the `<node> </node>` construct?

44. [✓] Does the test application connect the component through its ports to a data source and a data sink (linked either to files or driver/stub components)?

45. [✓] Are the data source and sink file resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

46. [✓] Are the library resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

47. [✓] Are the driver and stub components provided as separate components in a distinct component directory, just like the one being submitted for integration (where necessary and as indicated in the `README.txt` file)?



### 3.3 assessChildPerformance Report - Incomplete

#### INTEGRATION CHECKLIST

##### FILES AND DIRECTORIES

1. ☒ Are files for a single component stored in a directory with the same name as the component? For the purposes of this checklist, we will use `<componentName>` to stand for the component name. For example, if we were integrating a component named `myComponent`, then we would use `myComponent` everywhere we find `<componentName>` in the following. Note that the leading letter is in lowercase.

2. ☒ Does this directory have three sub-directories: `src`, `app`, and `config`?

Does the `src` directory contain one header file and three source files, named as follows.

3.1 ☒ `<componentName>.h`

3.2 ☒ `<componentName>Main.cpp`

3.3 ☒ `<componentName>Configuration.cpp`

3.4 ☒ `<componentName>Computation.cpp`

4. ☒ Does the `app` directory contain an XML application file named after the component but with the suffix `TEST`: `<componentName>TEST.xml`?

5. ☒ Does the `app` directory contain a `README.txt`?

6. ☒ Does the `config` directory contain a `<componentName>.ini` configuration file?

7. ☒ Does the configuration file contain the key-value pairs that set the component parameters?

8. ☒ Is each key-value pair written on a separate line?

Note that the instructions in the `README.txt` file should identify all system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`).

##### INTERNAL SOURCE CODE DOCUMENTATION

Does the `<componentName>.h` file contain a documentation comment with the following sections and accompanying text:

9. ☒ `/** @file <componentName>.h <one line to identify the nature of the file>`

10. ☒ `* <version information>`

11. ☒ `* <date>`

12. ☒ `* \section component_description Component Description`

13. ☒ `* <some meaningful descriptive text>`

14. ☒ `* \section lib_sec Libraries`

15. ☒ `* \section parameters_sec Parameters`

16. ☒ `* <b>Command-line Parameters </b>`

17. ☒ `* <b>Configuration File Parameters </b>`

18. ☒ `* \section portsa_sec Ports Accessed`

19. ☒ `* \section portsc_sec Ports Created`

20. ☒ `* <b>Input ports</b>`

21. ☒ `* <b>Output ports</b>`

Component: assessChildPerformance

Integrator: Rob Homewood

Date: 13/12/16

22. ☒ \* <b>Port types </b>
23. ☒ \* \section in\_files\_sec Input Data Files
24. ☒ \* \section out\_data\_sec Output Data Files
25. ☒ \* \section conf\_file\_sec Configuration Files
26. ☒ \* \section example\_sec Example Instantiation of the Component
27. ☒ \* \author

\* <forename> <surname>

Do all source files contain a block comment that gives the copyright notice, as follows.

```
/*
 * Copyright (C) 2014 DREAM Consortium
 * FP7 Project 611391 co-funded by the European Commission
 *
 * Author: <name of author>, <author institute>
 * Email: <preferred email address>
 * Website: www.dream2022.eu
 *
 * This program comes with ABSOLUTELY NO WARRANTY.
 */
```

- 28.1 ☒ <componentName>.h
- 28.2 ☒ <componentName>Main.cpp
- 28.3 ☒ <componentName>Configuration.cpp
- 28.4 ☒ <componentName>Computation.cpp

## COMPONENT FUNCTIONALITY

29. ☒ Does <componentName>.h contain a declaration of a class derived from `yarp::os::RFModule`?

```
class <ComponentName> : public RFModule {} // first char in uppercase
```

30. ☒ Is a ResourceFinder class, e.g. `ResourceFinder rf`, instantiated in <componentName>Main.cpp?

31. ☒ Does the component set the default configuration filename, named after the component with a .ini extension, in <componentName>Main.cpp?

```
rf.setDefaultConfigFile(<componentName>.ini);
```

32. ☒ Does the component set the default path (context) in <componentName>Main.cpp?

```
rf.setDefaultContext("components/<componentName>/config");
```

33. ☒ Does the component read all its key-value parameters from either a <componentName>.ini configuration file or from the list of command line arguments using the ResourceFinder `check()` method, called from within the `configure()` method in <componentName>Configuration.cpp:

```
<parameterValue> = rf.check(<key>, // parameter key
                          Value(<number>), // default value
                          "key value {int}", asInt()); // key value type
```

34. ☒ Does the component allow the port names to be set and overridden using the port name key-value parameters in the <componentName>.ini configuration file?

35. ☒ Do all port names have a leading /?

36. ☒ Do all input and output port names have a trailing : or :o, respectively?

37. ☒ Does the component allow the default name of the component to be set and overridden with the `--name` parameter?

```
module_name = if .check("name",
    Value("<componentName>"),
    "module_name (default)"), asString();
setName(module_name, n, str());
```

38. ☐ **Optional:** Does the component allow commands to be issued on a special port with the same name as the component by overloading the `respond()` method in the resource finder `RModule` class in `<componentName>Configuration.cpp`?

### COMPONENT COMMUNICATION

39. ☒ Are all *input* ports named according to the port naming convention: `/<component_name>/<port_name>;i`

40. ☐ Are all *output* ports named according to the port naming convention: `/<component_name>/<port_name>;o`

### COMPONENT UNIT TESTING

41. ☒ Is a unit test application named `<componentName>TEST.xml` provided in the app directory?

42. ☒ Are unit test instructions provided in a file named `README.txt` in the app directory?

- 42.1 ☐ Do the instructions identify the system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`)?

- 42.2 ☒ Do the instructions identify the resources required to run the test, including source (input) and sink (output) data files, driver and stub components, and libraries?

- 42.3 ☒ Do the instructions explain how the communication and computation functionality are validated by describing the (sink) output data that will be produced from the (source) input data?

- 42.4 ☒ Do the instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration (`.ini`) file are altered?

- 42.5 ☐ Do the instructions explain how the coordination functionality is validated by describing what changes in behaviour will occur when commands are issued interactively by the user to the component using the port named after the component itself (**optional**)?

43. ☒ Does the test application launch the component being tested on a YARP run servers called `dream1` using the `<node>` `</node>` construct?

44. ☒ Does the test application connect the component through its ports to a data source and a data sink (linked either to files or driver/stub components)?

45. ☒ Are the data source and sink file resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

46. ☒ Are the library resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

47. ☒ Are the driver and stub components provided as separate components in a distinct component directory, just like the one being submitted for integration (where necessary and as indicated in the `README.txt` file)?





### 3.4 assessChildPerformance Report - Complete

#### INTEGRATION CHECKLIST

##### FILES AND DIRECTORIES

1. ☒ Are files for a single component stored in a directory with the same name as the component? For the purposes of this checklist, we will use `<componentName>` to stand for the component name. For example, if we were integrating a component named `myComponent`, then we would use `myComponent` everywhere we find `<componentName>` in the following. Note that the leading letter is in lowercase.

2. ☒ Does this directory have three sub-directories: `src`, `app`, and `config`?

Does the `src` directory contain one header file and three source files, named as follows.

3.1 ☒ `<componentName>.h`

3.2 ☒ `<componentName>Main.cpp`

3.3 ☒ `<componentName>Configuration.cpp`

3.4 ☒ `<componentName>Computation.cpp`

4. ☒ Does the `app` directory contain an XML application file named after the component but with the suffix `TEST`: `<componentName>TEST.xml`?

5. ☒ Does the `app` directory contain a `README.txt`?

6. ☒ Does the `config` directory contain a `<componentName>.ini` configuration file?

7. ☒ Does the configuration file contain the key-value pairs that set the component parameters?

8. ☒ Is each key-value pair written on a separate line?

Note that the instructions in the `README.txt` file should identify all system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`).

##### INTERNAL SOURCE CODE DOCUMENTATION

Does the `<componentName>.h` file contain a documentation comment with the following sections and accompanying text:

9. ☒ `/** @file <componentName>.h <one line to identify the nature of the file>`

10. ☒ `* <version information>`

11. ☒ `* <date>`

12. ☒ `\section component_description Component Description`

13. ☒ `* <some meaningful descriptive text>`

14. ☒ `\section lib_sec Libraries`

15. ☒ `* \section parameters_sec Parameters`

16. ☒ `* <b>Command-line Parameters </b>`

17. ☒ `* <b>Configuration File Parameters </b>`

18. ☒ `* \section portsa_sec Ports Accessed`

19. ☒ `* \section portsc_sec Ports Created`

20. ☒ `* <b>Input ports</b>`

21. ☒ `* <b>Output ports</b>`

Component: assessChildPerformance

Integrator: Rob Homewood

Date: 21/12/16

22. ☒ \* <b>Port types </b>
23. ☒ \* \section in\_files\_sec Input Data Files
24. ☒ \* \section out\_data\_sec Output Data Files
25. ☒ \* \section conf\_file\_sec Configuration Files
26. ☒ \* \section example\_sec Example Instantiation of the Component
27. ☒ \* \author

\* <forename> <surname>

Do all source files contain a block comment that gives the copyright notice, as follows.

```
/*
 * Copyright (C) 2014 DREAM Consortium
 * FP7 Project 611391 co-funded by the European Commission
 *
 * Author: <name of author>, <author institute>
 * Email: <preferred email address>
 * Website: www.dream2022.eu
 *
 * This program comes with ABSOLUTELY NO WARRANTY.
 */
```

- 28.1 ☒ <componentName>.h
- 28.2 ☒ <componentName>Main.cpp
- 28.3 ☒ <componentName>Configuration.cpp
- 28.4 ☒ <componentName>Computation.cpp

## COMPONENT FUNCTIONALITY

29. ☒ Does <componentName>.h contain a declaration of a class derived from `yarp::os::RFModule`?

```
class <ComponentName> : public RFModule {} // first char in uppercase
```

30. ☒ Is a ResourceFinder class, e.g. ResourceFinder `rf`, instantiated in <componentName>Main.cpp?

31. ☒ Does the component set the default configuration filename, named after the component with a .ini extension, in <componentName>Main.cpp?

```
rf.setDefaultConfigFile(<componentName>.ini);
```

32. ☒ Does the component set the default path (context) in <componentName>Main.cpp?

```
rf.setDefaultContext("components/<componentName>/config");
```

33. ☒ Does the component read all its key-value parameters from either a <componentName>.ini configuration file or from the list of command line arguments using the ResourceFinder `check()` method, called from within the `configure()` method in <componentName>Configuration.cpp?

```
<parameterValue> = rf.check(<key>, // parameter key
                          Value(<number>), // default value
                          "key value {int}", asInt()); // key value type
```

34. ☒ Does the component allow the port names to be set and overridden using the port name key-value parameters in the <componentName>.ini configuration file?

35. ☒ Do all port names have a leading /?

36. ☒ Do all input and output port names have a trailing : or :o, respectively?

37. ☒ Does the component allow the default name of the component to be set and overridden with the `--name` parameter?

```
module_name = if .check("name",
    Value("<componentName>"),
    "module_name (default)"), asString();
setName(module_name, n, str());
```

38. ☐ **Optional:** Does the component allow commands to be issued on a special port with the same name as the component by overloading the `respond()` method in the resource finder `RModule` class in `<componentName>Configuration.cpp`?

### COMPONENT COMMUNICATION

39. ☒ Are all *input* ports named according to the port naming convention: `/<component_name>/<port_name>;i`

40. ☐ Are all *output* ports named according to the port naming convention: `/<component_name>/<port_name>;o`

### COMPONENT UNIT TESTING

41. ☒ Is a unit test application named `<componentName>TEST.xml` provided in the app directory?

42. ☒ Are unit test instructions provided in a file named `README.txt` in the app directory?

- 42.1 ☐ Do the instructions identify the system architecture ports used by the component (i.e. any port that is defined in one of the three system architecture placeholder components `sensoryInterpretation`, `childBehaviourClassification`, or `cognitiveControl`)?

- 42.2 ☒ Do the instructions identify the resources required to run the test, including source (input) and sink (output) data files, driver and stub components, and libraries?

- 42.3 ☐ Do the instructions explain how the communication and computation functionality are validated by describing the (sink) output data that will be produced from the (source) input data?

- 42.4 ☒ Do the instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration (`.ini`) file are altered?

- 42.5 ☐ Do the instructions explain how the coordination functionality is validated by describing what changes in behaviour will occur when commands are issued interactively by the user to the component using the port named after the component itself (**optional**)?

43. ☒ Does the test application launch the component being tested on a YARP run servers called `dream1` using the `<node>` `</node>` construct?

44. ☒ Does the test application connect the component through its ports to a data source and a data sink (linked either to files or driver/stub components)?

45. ☒ Are the data source and sink file resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

46. ☒ Are the library resources provided in the `config` directory (where necessary and as indicated in the `README.txt` file)?

47. ☒ Are the driver and stub components provided as separate components in a distinct component directory, just like the one being submitted for integration (where necessary and as indicated in the `README.txt` file)?