CAE Inc.

1135 Innovation Drive
Ottawa, Ont., K2K 3G7 Canada
Tel: 613-247-0342
Fax: 613-271-0963

HUMAN BEHAVIOUR REPRESENTATION
TASK 2
FINAL REPORT

FOR

DEFENCE RESEARCH AND DEVELOPMENT CANADA

1133 Sheppard Avenue West, Toronto, Ontario M3K 2C9
Vlad Zotov, Defence Scientist
PWGSC contract number: 135235-04

29 September 2015

Document No. 5746-004 Version 01

DRDC-RDDC-2015-C351

APPROVAL SHEET

Document No. 5746-004 Version 01

Document Name: Human Behaviour Representation
Task 2
Final Report

Primary Author

5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004

Name **Alain Dubreuil** **Date**

Position Senior Modelling and Simulation Consultant

Reviewer

5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004

Name **Scott Guenther** **Date**

Position Senior Modelling and Simulation Consultant

Approval

5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004
5746-004 5746-004 5746-004 5746-004 5746-004 5746-004 5746-004

Name **Richard Smith** **Date**

Position Senior Project Manager

REVISION HISTORY

Revision	Reason for Change	Origin Date
Version 01	Initial document issued.	29 September 2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Defence Research and Development Canada contracted CAE to develop a behaviour model using the Integrated Performance Modelling Environment (IPME) toolkit, to represent the behaviour of the crew members of the Leopard tank. CAE was also contracted to integrate the speech engines with the IPME model and to integrate the resulting system with the Virtual Leopard 2 Unity3D environment developed by Department of National Defence (DND). This work was conducted as part of Task 2 of the Human Behaviour Representation (HBR) contract.

CAE carried out the work required for HBR Task 2 from December 2014 to September 2015. CAE developed the IPME model to represent the behaviour of the gunner and loader crew members of the Leopard tank. A speech recognition engine and a speech synthesis engine were integrated with the IPME model to accept commands from a crew commander trainee and to utter responses from both the loader and the gunner. Due to the delay in the development of the Unity3D virtual environment, CAE developed a test bed to duplicate the interactions that are expected to occur between the IPME model and the virtual environment. Finally, CAE documented the system, producing a Hierarchical Task Analysis of the crew's actions and appropriate documents to explain the system design and to specify the interactions between the parts of the system.

CAE tested the system, using the test bed in lieu of the virtual environment. CAE is confident that the produced system meets the requirements of HBR Task 2. CAE recommends that a subject matter expert be engaged to validate the system and that a subsequent HBR Task be put in place to support the development of the virtual environment.

1 INTRODUCTION

1.1 Background

The Army Learning Support Centre (ALSC) at Combat Training Centre Gagetown requested Defence Research and Development Canada (DRDC) for assistance to determine whether their model of a Virtual Leopard 2 (VLeo2), developed in the Unity3D game engine environment, would be suitable for training certain tank crew roles and to determine what level of immersion is necessary for effective training. To help determine the training suitability of VLeo2, DRDC staff will run experiments using the VLeo2 simulator integrated with a behaviour modelling application, a speech recognition engine, and a speech synthesis engine.

DRDC contracted CAE to develop the behaviour model using the Integrated Performance Modelling Environment (IPME) toolkit. CAE is also contracted to integrate the speech engines with the IPME model and to integrate the resulting system with the VLeo2 Unity3D environment. This work was conducted as part of Task 2 of the Human Behaviour Representation (HBR) contract.

1.2 Objective

The overall objective of this project was to develop a Leopard tank experimental trainer using IPME, Unity3D, and speech recognition and synthesis engines.

1.3 This Document

This document is the final report describing the development of three of the four parts of the VLeo2 trainer: the IPME model, the speech recognition implementation and the speech synthesis implementation.

1.3.1 Scope

This document does not address the development of the VLeo2 virtual environment. That sub-system is the responsibility of the ALSC and will be documented under a different cover.

1.3.2 Organisation

This document is organised in the following sections:

- Section 1: Introduction.
- Section 2: System Design. A description of the design of all of the components that comprise the VLeo2 trainer, excluding the Unity3D based VLeo2 virtual environment.
- Section 3: Supporting Elements. Additional information on topics not directly related to the system design or that aid in understanding the scope of the system.

- Section 4: Concluding Elements. Tests results, conclusions and recommendations.
- Appendix A: Acronyms.
- Appendix B: Unity3D Animations. Additional information for the development of avatar and equipment animations in the virtual environment.

2 SYSTEM DESIGN

2.1 High-Level Components

At the highest level, there are four major components to the VLeo2 trainer system:

1. the IPME application;
2. the VLeo2 virtual environment;
3. the Microsoft Speech Platform 11 system; and
4. the Microsoft SAPI 5 speech synthesis application with Ivona voice files.

The IPME application simulates the behaviour of three of the four Leopard tank crew members, i.e. the gunner, the loader and the driver. The driver model is currently empty as no driver behaviour actually needs to be modelled in IPME, as tank movement is being handled by the Unity3D environment. The behaviours are modelled using a network of tasks, many of which are triggered by state changes. The represented behaviours are based on Leopard 2A4 variant of Leopard tank. Standard Operating Procedures (SOPs) and do not incorporate any Leopard 2A4M or 2A6M variants procedures.

The VLeo2 virtual environment is a Unity3D representation of the interior of the Leopard 2A4 tank, with full animation of the controls and equipment movement. The application will also animate the posture and movements of three simulated crew members. Animation triggers will come from the IPME model through a communication interface. In addition, the VLeo2 application will model the environment in which the Leopard tank operates, including the representation of the terrain and enemy forces.

Microsoft Speech Platform 11 is a speech recognition system. The platform does not need to be trained for specific users and is capable of “understanding” different English accents. Developers can define a grammar that limits what the application reacts to.

Microsoft SAPI 5 is a speech synthesis system. This system has been augmented with three different Ivona voices to help differentiate the verbal communications coming from the different crew members.

More detailed information on these systems is provided in individual sections of this document.

2.2 Interactions

Figure 2-1 depicts the interactions amongst the four major system components. From the figure, it can be seen that a plugin component to the IPME application was created to interface with the three other components. When a trainee speaks into the headset’s microphone, the speech recognition system sends a message to the plugin. The plugin interprets the message and sets a variable in the IPME model. The IPME model reacts to the variable’s state change and executes an appropriate series of tasks which culminate, in most cases, in another variable

state change. In turn, this variable state change is picked up by the plugin who sends a message to the VLeo2 application through a Representational State Transfer (REST) fully compliant (RESTful) interface. The VLeo2 application receives the message and reacts by executing an associated animation. Once the animation is complete, the VLeo2 application fires a message back to the plugin through its own RESTful interface.

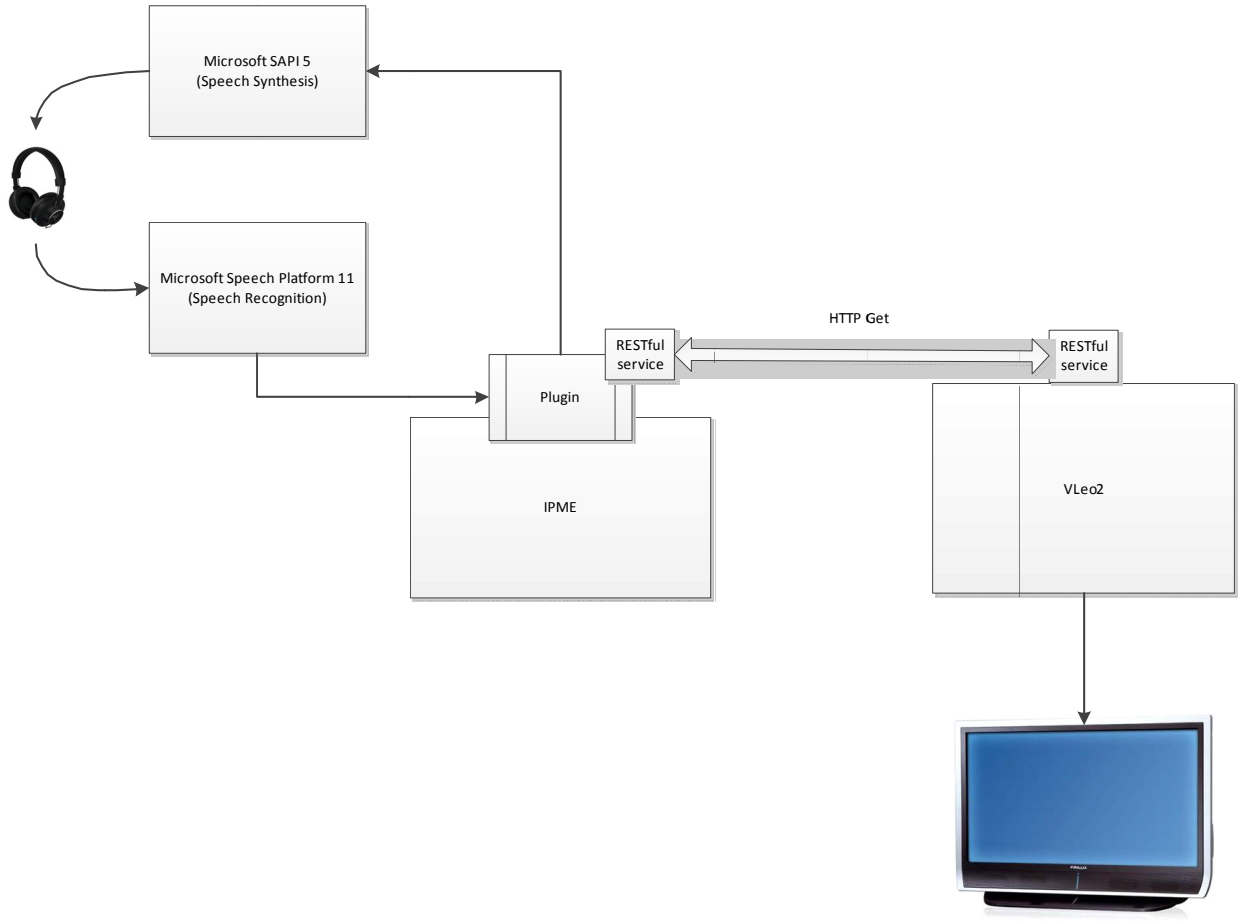

Figure 2-1: System Interactions

Upon reception of the message from the VLeo2 application, the plugin interprets the message and sets a variable in the IPME model. The IPME model continues the series of tasks that it has initially started, which causes it to either request another animation from the VLeo2 application or request that a response be uttered through the speech synthesis system. The plugin reacts to this request appropriately, i.e. it either sends another message to the VLeo2 application through the RESTful interface or sends a phrase to be uttered to the Microsoft SAPI 5 application with a mention of the destination crew member.

2.3 IPME Plugin Details

The plugin to the IPME model is the central point for the interactions between the components. The plugin composition is shown in Figure 2-2.

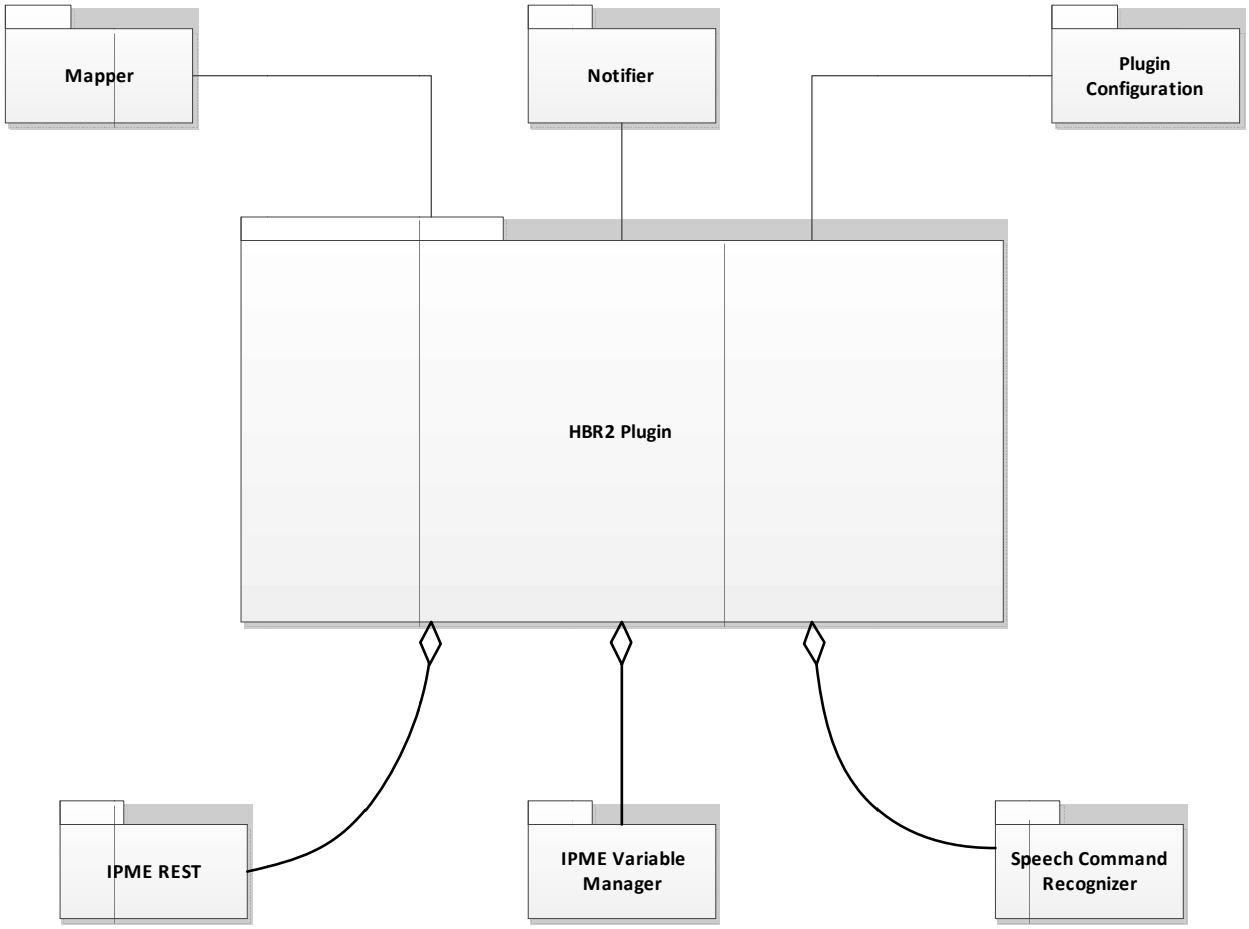

Figure 2-2: Plugin Components

Internal to the plugin are three components: the Mapper, the Notifier and Plugin Configuration. The purpose of each is described below:

- The Mapper's function is to translate the externally submitted variables into IPME variables, and vice-versa. The Mapper also translates speech recognition commands into IPME variable state changes.

- The Notifier's function is to receive variable change event notifications from the IPME model and pass these notifications through the Mapper so that the appropriate associated message can be sent to the VLeo2 application or to the Microsoft SAPI 5 application.

- The Plugin Configuration's function is to define configuration parameters, such as the location of the grammar files for the speech recognition system, the inbound and outbound rules of the Mapper component, and the parameters for the RESTful interface.

External to the plugin are three other components: IPME REST, IPME Variable Manager and Speech Command Recognizer. The purpose of each is described below:

- The IPME REST component implements a RESTful service to interact with the VLeo2's own RESTful service and to exchange messages between IPME and VLeo2. The RESTful service uses "http-get" functions with the serialization option set to JavaScript Object Notation (JSON). RESTful is a lightweight service; hence, it is not expected to impose significant slowdowns on the communications between IPME and VLeo2.

- The IPME Variable Manager is used by the plugin to set variable values inside the IPME model. Both the Speech Command Recognizer and the IPME REST components may be the source of the change in variable value.

- The Speech Command Recognizer's function is to accept the output from the Microsoft Speech Platform 11 application, interpret the output, and pass the interpretation result to the Mapper component whose output is passed to the IPME Variable Manager.

As stated, these six components are central to the interactions that take place between the four high-level components described in Section 2.1. The configuration for these various components is set in the plugin configuration files, i.e. the intention is to avoid having hard-coded parameters, unless it is judged absolutely necessary.

2.4 IPME Model Design

2.4.1 General

The IPME model contains two main networks of tasks: one for the gunner and one for the loader. The network of tasks for the driver is empty as there are no driver actions that need to be modelled at the moment. This is due to the fact that the driver very seldom speaks in the tank and to the fact that the tank movement will be modelled by the VLeo2 virtual environment. When crew commander (CC) commands destined for the driver are received from the speech recognition system, the plugin sets a specific variable in the IPME model. The change in value of that variable is picked up by the plugin's Notifier function, which in turn sends a message to the VLeo2 environment through the RESTful interface. The VLeo2 environment sends a message back to IPME to acknowledge the command. The reception of this message resets the variable inside the IPME model. As a result, no IPME driver task is involved in the transaction. Eventually, there may be a need to add tasks to the driver network to take into account fault conditions or safety situations.

The gunner and loader task networks adopt the same trigger pattern. A recurring "Listen" task loops continuously to react to change in value of variables brought about by the plugin, in most cases caused by a CC order. The Listen task triggers the appropriate sequence of tasks based

on the variable's change. The sequence of tasks ends when the last action of the sequence is completed. Often, this will be the utterance of a response to the CC.

In both the gunner and loader task networks, a feedback loop is attached to any task that requires an animation in the VLeo2 virtual environment. The feedback loop checks for a change in a variable value that indicates that the animation has been completed in the virtual environment. That change is effected by the plugin upon reception of the appropriate message from the VLeo2 environment.

2.4.2 Gunner Model Characteristics

The gunner network of tasks is the most elaborate network in the IPME model. It reacts to several commands from the CC. There are also many loops and conditional checks throughout the model. The task sequences are generally self-explanatory.

The most complex sequence is related to the target detection and firing process. Detection may originate from the gunner (task 1.33 Detect Contact) or from the CC (task 1.76 Receive LASE order and target description). The sequence of actions in both cases ends with the reporting of the range to the target. In the case of detection by the gunner, the VLeo2 environment must tell IPME about the presence of a target in its scanning area, as IPME does not have any visibility into the virtual environment. This is accomplished via a message that contains the type of target, whether it is moving or not, and the range to the target. This message causes the sequence that starts at task 1.33 to execute.

The reception of a firing order from the CC causes the sequence of tasks starting at task 1.39 to execute. The firing process loops on itself until the target is destroyed or the CC orders "target stop".

2.4.3 Loader Model Characteristics

The loader model contains a lesser number of task sequences than the gunner model. At the moment, the loader model handles only a single type of ammunition, the sabot. The armoured corps Subject Matter Expert (SME) has mentioned that the sabot is fired in 95% of the cases, so the exclusion of the other types of rounds should not be significant. Adding the other types of rounds should not be difficult but would increase the complexity of the model by adding many conditional statements to handle the variety of cases.

2.5 Speech Recognition Implementation

2.5.1 General

The Microsoft Speech Platform 11 application is used to capture CC commands, parse them and send them to the plugin for action. The application uses a grammar configuration file to define the sentences that are significant to the VLeo2 system. Any sentence that is not part of the grammar is not passed along to the plugin.

2.5.2 Configuration

The Speech Recognition implementation is an organic part of the IPME plugin and it must be installed in the same directory as the plugin executables, i.e. in the "Console" directory of the IPME installation. Its functionality can be turned on or off by setting or clearing the *Enable* flag of the <SpeechCommandConfig> tag, which is found in file HBR2Plugin.dll.config. This same tag also specifies which *locale* (i.e. language) will be used. The locale is defaulted to *en-US*. The specified locale must be installed for Speech Platform 11, otherwise the execution will fail.

The commands recognized by the speech recognition engine are defined in XML grammar files where they are associated with IPME variables. The grammar files have to be compliant with the Speech Recognition Grammar specification (http://www.w3.org/TR/speech-grammar/). The current implementation uses two configuration files: GunnerAndLoaderCommands.grxml and DriverCommands.grxml. The following is an example of an association defined in the former:

```
<item>
  <ruleref uri="#CrewAction" />
  <tag> $.Command = 'GunAndLoader.CrewAction'; </tag>
  <tag> $.CC_CrewAction_GNR = 'true'; </tag>
  <tag> $.CC_CrewAction_LDR = 'true'; </tag>
</item>
```

The example reveals that the "Crew Action" command would cause the variables CC_CrewAction_GNR and CC_CrewAction_LDR to get set to True in IPME by the plugin. The sentence to recognize, in this case "Crew Action", is defined in the same grammar file:

```
<rule id="CrewAction" scope= "private" >
  <example>CREW ACTION</example>
  <item> Crew </item>
  <item> action </item>
</rule>
```

Any additional commands can be added to the grammar files to link the speech recognition output with the IPME model. It is also possible to define additional grammar files by specifying their existence using a <Grammar> tag under the <Grammars> section of the HBR2Plugin.dll.config file.

2.5.3 Processing Overview

The IPME plugin and the Speech Recognition module are both loaded when the IPME execution engine is re-started after its initial load. Module initialization occurs when the IPME model is run. During its initialization, the module loads its grammars and enters into a listening mode.

When the user speaks a command into the microphone, the recognized speech is compared against the grammar's vocabulary. If there is a match, then the event is passed to the IPME plugin. The event includes the command identifier and the event variables as they are defined in the grammar file.

Upon arrival in the plugin, a mapping process translates the incoming variables into IPME variables. These mappings are defined in the <MapperConfig> section of the HBR2Plugin.dll.config file. The plugin supports two configuration modes:

1. Single mapping: this mode handles one-to-one mappings, i.e. a single input variable being mapped to a single IPME variable. Each mapping can pass along the value defined in the grammar without any modification:

   ```
   <InMap InVariable="CC_CrewAction_GNR" IPMEVariable="CC_CrewAction_GNR" Type="bool" />
   ```

   Or it can specify which value to set in the IPME variable based on the value received in the incoming variable:

   ```
   <InMap InVariable="turretPowerState" InVariableValue="turretOff"
   IPMEVariable="LDR_ChangeTurretPower" IPMEVariableValue="Off" Type="string"/>

       <InMap InVariable="turretPowerState" InVariableValue="turretOn"
   IPMEVariable="LDR_ChangeTurretPower" IPMEVariableValue="On" Type="string"/>
   ```

2. Multi mapping: this mode allows more complex variable conversions. One or more input variables can be resolved into one or more IPME variables. The number of input variables does not have to match the number of IPME variables. Likewise, the IPME variable value(s) can be copied directly from the incoming variable or can be set to any desired static value.

Inherent to the Speech Recognition module, the variable changes induced by the command recognition event are not correlated with the logical processing order specified in the IPME model. For example, if the Speech Recognition engine recognizes a command "*Eliminate Drift*", the corresponding IPME variable of *CC_EliminateDrift* will be set to *True* regardless of the IPME execution context (e.g. before the "Crew Action" command). In some very specific cases, the IPME model handles the asynchronous nature of the speech recognition events by using flag variables. In most cases, the command will be executed regardless of the context. This is not necessarily different than "real life", as nothing prevents a CC from giving any command to a crew member at any point in time. What differs is the reaction of the crew member – in a real-life situation, the crew member might delay his reaction to the order until his current action is completed. In the IPME model, the crew member will execute both commands simultaneously.

2.6 Speech Synthesis Implementation

2.6.1 General

The Speech Synthesis (SpSyn) module is implemented with Microsoft Speech Application Programming Interface 5.0 (SAPI 5.0) which is a part of the Microsoft Windows operating system. The module can use any synthetic voices compatible with SAPI 5.0.

2.6.2 Configuration

The SpSyn component resides in the same directory as the IPME plugin. A user can turn its functionality *on* or *off* by changing the *Enable* flag of the <*SpeechSynthesisConfig*> node of the plugin configuration file (HBR2Plugin.dll.config).

SpSyn is an "event subscriber" system that reacts to a variable change event within the IPME model. Each event of interest that is expected to result in a speech synthesis action must have a <*Phrase*> entry under the <*SpeechSynthesisConfig*>/<*Phrases*> node in the plugin configuration file. The voice to use to pronounce the phrase is specified as the *Voice* attribute of the <*Phrase*> node.

The configuration consists of the *condition* to react upon and the *action* which is the actual synthetic speech playback. The *condition* specified as a <*Trigger*> node sets one or more IPME variables to monitor for a change. A trigger can react to a single variable value change specified as <*SingleCondition*>; or to multiple variables' values grouped under AND/OR clauses specified as <*AndCondition*> and <*OrCondition*>, correspondingly.

As soon as the condition criteria are met, the SpSyn engine calls the *action* phase. The <*ActionTemplate*> node specifies the actual text to be spoken out. A text can be directly specified, such as in the following example where the spoken text would be "MRS Checked":

```
<Action>
  <ActionTemplate>M.R.S. checked</ActionTemplate>
</Action>
```

Alternatively, the Action node can contain one or more template insertion. The template insertions are specified in curly brackets *{n}* within the text; where *n* is a placeholder number. If the <*TemplateParams*> node is not *nil*, the template insertion placeholders are filled with the specified IPME variables' values. In the following example, the spoken text would be "Contact <type>" where *type* takes on the value that IPME variable *Contact_Type* is currently set to:

```
<Action>
  <ActionTemplate>CONTACT, {0}</ActionTemplate>
  <TemplateParams>
      <Param Position="0" VariableName="Contact_Type"/>
  </TemplateParams>
</Action>
```

Before replacing a template insertion with the IPME value, the value can be modified with a filter specified as the *ValueConverter* attribute of the <*Param*> node. This is usually used for numeric

values pronunciation customization. For example, the *ValueConverter* set to *NumberByDigit* will result in speaking each digit separately rather than a full number; that is, the number *852* will be spoken as '*eight five two*' instead of '*eight hundred fifty two*'. The ValueConverter set to NumberByHundred will result in speaking the numbers ending with 00 as '*hundred*'; that is, the number *2500* will be spoken as '*two seven hundred*' rather than as '*two thousand and seven hundred*'.

2.6.3 Voices

The voices used in the CAE deliverables for HBR Task 2 are Ivona synthetic voices purchased by the Task. There are three voices available:

1. Joey, an American English male voice;

2. Salli, an American English female voice; and

3. Brian, a British English male voice.

In the IPME plugin configuration file, Joey has been assigned as the loader's voice, whereas Salli has been assigned as the gunner's voice. To change the voices, the user needs to access the plugin source code, modify specific files and rebuild the solution. The following changes are required in the IPME.Extent solution:

1. If a new voice needs to be added to the solution, for example after purchasing a new Ivona voice file, access the HBR2Plugin.SpeechSynthesis.xsd file and add the name of the voice to the xs:enumeration fields under XS:restriction. This is only if a new voice needs to be added.

2. In file HBR2Plugin.dll.config, scroll down to the gunner or loader responses (depending on which crew member's voice needs to be changed), and change the "Voice" value to the desired voice name. Example:

```
<Phrase Name="Laser On Confirmation" Voice="Salli"> becomes

<Phrase Name="Laser On Confirmation" Voice="Joey">
```

2.6.4 Processing Overview

The SpSyn module gets loaded along with the IPME plugin. Upon the beginning of the IPME model execution, the configuration specified in the *.config* file is applied to the SpSyn engine. During the initialization, the SpSyn engine collects all the variables specified in the <*Trigger*> nodes and stores them in a cache to monitor their change in value.

Every time the IPME model executes a step (enters a task), the variables of interest are compared against the values in the cache. If the value of a variable has changed, the engine finds the *Trigger(s)* where the variable was specified. If the full *Trigger* condition is met, the

corresponding *Phrase* is picked for the execution, and the cache value is updated with the new value.

The SpSyn engine performs the analysis of the text to be spoken, as it is specified in the <*ActionTemplate*> node. The text may or may not contain template insertions (enclosed in the curly brackets {}). If the phrase to speak contains template insertions, the user must specify the <*TemplateParams*> node. This node contains the list of IPME variables that are to be used in the place of the insertions.

An IPME variable can be transformed to a specific format prior to being used to replace an insertion. The format is stored as an optional *ValueConverter* attribute. Currently, the SpSyn module supports two converters – *NumberByDigit* and *NumberByHundred* that can be applied to numeric values with the purpose of altering the default pronunciation for numbers. A more specific description of the converter can be found in Section 2.6.2.

Once the text to speak out is created, the SpSyn engine switches to the voice reproduction phase. The SpSyn engine initializes the speech context by specifying the voice to be used. This is followed by the text being sent to the SAPI runtime for reproduction.

Note that despite the fact the speech synthesis process is a non-blocking call (which means the IPME Model can continue executing) the actual speech stream is processed by the operating system as a queue. That is, if there are several phrases being sent to the speech synthesis system, they will be spoken sequentially in order of their arrival in the queue rather than simultaneously.

2.6.5 Investigation: Multiple Simultaneous Voices

In the Leopard tank, the CC wears a headset in which is fed two radio streams, i.e. one in each ear. To avoid overpowering the radio chatter, the communications from the crew members are made off the intercom instead of on. This means that both the gunner and the loader yell their responses in the tank's crew compartment, over the ambient noise. The driver does not respond to CC commands, but may communicate an emergency cry of *STILL* over the intercom if he/she notices a safety issue.

In the current speech synthesis engine setup, responses from the crew members are queued; and therefore, cannot be uttered simultaneously. CAE investigated options to overcome that limitation so that the responses would be more realistic. A relatively inexpensive option would be to save the output of the SAPI 5 engine to a .wav file, which in turn could be read. The advantage of .wav files is that they can be played simultaneously on a single sound card. However, reading the .wav files would require a DirectX implementation. Unfortunately, DirectX has a reputation to be error-prone. It would be feasible to work out most of the problems if the right environment could be achieved on the target computer, but the stability of the system could prove problematic.

CAE did not push the investigation further as the desire to output simultaneous responses is not a requirement of the VLeo2 system at this time. Other options may be investigated at a later time under a subsequent Task if required.

2.7 RESTFul Interface

2.7.1 General

The REST module (RM) is responsible for establishing communications between the IPME model and the VLeo2 virtual environment. It could be used for communications with other external systems, if required. The main purpose of the communication is variable exchange, that is, the IPME and VLeo2 applications can notify each other about specific events or requests using a set of variables whose values are meaningful. This set of variables, their values and meaning are defined in an Interface Control Document (ICD). The ICD is delivered under separate cover. In general, the variables sent from IPME to VLeo2 represent requests for avatar or virtual environment animations; whereas variables sent from VLeo2 to IPME are animation completion notifications or virtual environment information.

RM plays a dual role. When used as a means to notify external systems, RM acts as a REST client that calls an external party over the Hypertext Transfer Protocol (HTTP). When used to accept the invocations from external systems, RM acts as a REST service host accepting requests over HTTP. Note that currently RM supports only HTTP GET verbs for both incoming and outgoing communications. Both incoming and outgoing communications use the JSON serialization mode.

2.7.2 Configuration

2.7.2.1 Installation

RM resides in the same directory as the IPME plugin. A user can turn its functionality *on* or *off* by setting the *Enable* attribute of the <*RESTConfig*> node of the plugin configuration file.

The Hosting Service name and the Service Port number are specified as <*ServiceName*> and <*Port*>nodes correspondingly.

2.7.2.2 Methods to Access IPME Variables

To get an IPME variable, the invoked Uniform Resource Locator (URL) has to be in the following format:

http://{ComputerName}:<Port>/<ServiceName>/getvar/{IPMEVariable}

To set an IPME variable, the external system has to invoke the following URL:

http://{ComputerName}:<Port>/<ServiceName>/setvar/{IPMEVariable}={newValue}

To set multiple IPME variables inside a single call, the external system has to invoke the following URL:

http://{ComputeName}:<Port>/<ServiceName>/setvars/{IPMEVar1}={newVal1}&{IPMEVar2}={newVal2} etc…

Note that all parameters passed over HTTP using the HTTP GET verb are strings. Hence, it is the responsibility of the caller to convert the obtained value into the appropriate data type.

2.7.2.3 Notifier Configuration

Outgoing notification of the external systems is stored in the <*NotifierConfig*> node of the plugin configuration. Notification can be turned on/off with the *Enable* attribute of the node. The base URL of the external system to notify is set as the *ServiceUri* attribute.

All the notification entries are configured as a list of <*Notification*> nodes under <*NotifierConfig*>/<*Notifications*>.

The notification structure is organized the very similar way as the Speech Synthesis section. A condition for notification invocation is specified as <*Trigger*> that can include a single IPME variable to react upon, or as a series of variables grouped with <*AndCondition*> or <*OrCondition*> clauses.

Each <*Trigger*> has an associated <*Action*> node that contains the relative URL of the remote service to be called. The relative URI is specified in the <*ActionTemplate*> node. The following is an example of a Notification entry in the configuration file:

```
<Notification Name="Send Check Thermal Switch command to Vleo2">
    <Trigger>
        <SingleCondition>
            <VariableCondition Variable="GNR_CheckThermalSwitch" Condition="Equal" Value="Check"/>
        </SingleCondition>
    </Trigger>
    <Action>
        <ActionTemplate>/animate/thermalSwitchStatus=check</ActionTemplate>
    </Action>
</Notification>
```

Similar to the SpSyn implementation, the <ActionTemplate> may contain template insertions that are replaced with the current IPME values at the time of invocation.

2.7.3 Processing Overview

2.7.3.1 General

RM gets loaded along with the IPME plugin. Upon beginning of the IPME model execution, the configuration specified in the *.config* file is applied to the RM engine. During the initialization, the RM engine collects all the variables specified in the <*Trigger*> nodes of the <*NotifierConfig*> section and stores them in the cache for value change detection.

Incoming REST service is a singleton, that is, only one REST Hosting instance can be initiated per *IPMEConsole* instance.

2.7.3.2 Incoming Communication

An external system may invoke HTTP GET verbs in one of the formats specified in the section 2.7.2. When a request is received, the associated IPME variable value is retrieved from the IPME model and its value is returned with JSON serialization notation.

A request for setting IPME variables is processed over several processing pipes. First the RM extracts the incoming variable name(s) from the request. Note that the names may or may not correspond to existing IPME variables. Using the retrieved name(s), RM searches for a matching variable mapping entry in the plugin configuration file under the <*MapperConfig*> node. How the search is performed is dependent on whether the request is for a single variable value or for multiple variable values.

In the case of a single variable value, the RM searches through the <*Incoming*> section among all <*InMap*> entries by comparing the *InVariable* attribute value with the incoming variable name. If a value is specified for the variable, RM will also search for the specified value. If there is a match for the variable name (and value when specified), the IPMEVariable attribute value is used to identify the IPME Variable name that is to be set. In addition, the <InMap> node specifies the incoming variable type as the *Type* attribute. The incoming value gets converted to the specified variable *Type* and stored in IPME. If no single entry mapping is found under the <Incoming> section, RM carries out a *Multi Mapping* search for the variable. The following are examples of <Incoming> section entries, with and without specified values:

```
<InMap InVariable="CC_EliminateDrift" IPMEVariable="CC_EliminateDrift" Type="bool" />

<InMap InVariable="laserPowerStatus" InVariableValue="laserOn" IPMEVariable="GNR_LaserOn"
IPMEVariableValue="false" Type="bool"/>
```

In the case of a request to update multiple variables, the *Multi Mapping* section of the plugin configuration file (specified as <*IncomingMulti*> node) is checked. *Multi Mapping* allows resolving of one or more incoming variables into one or more IPME variables. The number of incoming variables doesn't have to match the number of IPME variables specified in a node. For example, two incoming variables may be resolved to three IPME variables; or four incoming variables resolved to a single IPME variable, and so on.

RM compares incoming variables (and optionally their values) with a set of <*Variable*> under the <*InMultiMap*>/<*InVars*> node. If there is a matching candidate, the set of IPME variables specified under <*InMultiMap*>/<*OutVars*> is identified. The value to assign to the IPME variable can be static or dynamic. A dynamic value is specified in the configuration file in the format IPMEVariableValue=$.IncomingVariableName. At runtime its value is replaced with the value that the specified incoming variable possesses. The following is an example of a Multi Mapping entry:

```
<InMultiMap> <!-- Voice Recognition Command -->
    <InVars>
        <Variable InVariable="CC_CrewAction_GNR_LDR" InVariableValue="true"/>
    </InVars>
    <OutVars>
        <Variable IPMEVariable="CC_CrewAction_GNR" IPMEVariableValue="$.CC_CrewAction_GNR_LDR" Type="bool"/>
```

```
        <Variable IPMEVariable="CC_CrewAction_LDR" IPMEVariableValue="$.CC_CrewAction_GNR_LDR" Type="bool"/>
    </OutVars>
</InMultiMap>
```

Once the incoming mapping is resolved, the RM sets IPME variables to their new values. Note that if no appropriate mapping can be found in the plugin configuration file, the incoming values are passed as is. That is, the variable name(s) used in the request are passed with their value being left as a string, but no IPME variable will be changed in the IPME model.

2.7.3.3 Outgoing Communication

Outgoing communication is an asynchronous process that reacts to value changes to variables of interest. If the <*NotifierConfig*> is enabled, the plugin manager collects all the IPME variable names under <*Notification*>/<*Trigger*> nodes and stores those in the cache for future value comparison.

Once an IPME variable value change event is detected, the manager checks whether the variable whose value has changed is a variable of interest. In the affirmative, conditions specified in the associated <*Trigger*> node are checked.

When all the conditions of the <*Trigger*> node are met, the manager executes the <*Action*> part of the <*Notification*>. If there is a <*TemplateParams*> node, then the <*ActionTemplate*> contains template insertions that have to be replaced with the current IPME variable values. The entire process is identical to that of SpSyn with the exception that no *ValueConverter* is currently supported for the *Notifier*.

Processing of <*ActionTemplate*> results in the invocation of a relative URI of the remote REST service. The plugin manager passes the base URL (specified as a *ServiceUri* attribute of <*NotifierConfig*>) along with the generated relative URI to the RM. The RM then carries out the invocation by forming a valid HTTP string, initiating a GET verb and invoking the HTTP stack.

2.8 Test Bed Implementation

CAE implemented the test bed to support the testing of the integration of the IPME model and speech engines. Due to delays in the availability of the VLeo2 virtual environment, the development of the test bed became essential to verify the implementation of the other parts of the system.

The test bed's RESTFul interface mimics closely the one implemented in the plugin. Connected to the interface are a small set of functions whose purpose is to implement timers for each command received from the IPME application. The aim is to simulate the time that it would take for the Unity3D application to animate the movements of the loader and gunner avatars. Once an animation is complete, or in the case of the test bed a timer completes, a message is sent back to the IPME application via the RestFul interface. The message informs IPME that the animation is complete, which causes a transition to the next task inside the IPME model.

Central to the test bed is a configuration file that lists the message payloads and the timer duration associated with each message. The file is in Comma Separated Value (CSV) format

and is best read and modified using Microsoft Excel. It is located under the C:\Vleo2\Vleo2Testbed folder and is named CommMatrix.csv. The file contains the following fields:

- NameIn: this is the name of the variable that is part of the message payload received from IPME.
- ValueIn: this is the value of the variable when it arrives from IPME. The test bed will react to the message only if the name and value of the variable matches a line in the configuration file.
- NameOut: the name of the variable that is part of the message payload sent to IPME when the timer has completed.
- ValueOut: the value of the variable that is in the payload of the message sent to IPME upon timer completion.
- Delay: the length of time in seconds that the timer will run when the associated message from IPME is received. The value for each timer is arbitrary, but where possible CAE adjusted the delay so that the associated action would take a total amount of time that is close to the timings mentioned by the SME in discussions.

Adding messages to the application is done by adding entries to the CommMatrix CSV file. All message payloads are listed in the ICD and must be abided to for the system to function correctly.

2.9 System Limitations

The speech engines and IPME model implementation produced under HBR Task 2 provides a system that meets the requirements that were agreed upon during the visit to Gagetown in February of 2015 and the discussions with ALSC and the armoured corps SME the details of the equipment that is available to the crew members (weapons, sights, lasers, etc). As such, the system can handle CC commands given in the correct order as per the standard operating procedures of the Leopard tank. The system will also handle most commands that are not given in the proper order but that are independent from other orders. The implementation covers a large array of situations, but it has some limitations that are worth mentioning. Some of the most important limitations are:

1. Except where it is necessary, the IPME model does not keep track of the context. For example, there is nothing preventing the CC from giving a CREW ACTION order in the middle of a firing sequence. The effect would be unrealistic in that the gunner and loader would carry the firing sequence and the action drill simultaneously, whereas in a real situation the crew members would likely ignore the action drill orders and tell the CC to update his situational awareness level!
2. The IPME model currently handles fully any order containing the sabot ammunition. Orders containing any other type of ammunition have not been fully implemented. The SME has

mentioned that the sabot is fired 99 percent of the time, so to reduce complexity CAE decided to forego the other types of ammunition at this time.

3. The IPME model currently handles the *tank* target type, but no specific effort has been made to include other types of targets at this time. However, CAE is confident that any of the listed target types would be handled correctly. This has not been tested due to time limitations.

4. No user errors have been implemented. No equipment faults have been implemented. See Section 3.4 for more details.

5. No action requiring that a crew member pops up through an open hatch has been implemented.

6. The smoke grenade discharger and anti-aircraft machine gun ("ack ack" located on the turret) will not be modelled in the virtual environment. Hence, no action related to those systems has been implemented.

7. Although the COAX machine gun will not be modelled in the virtual environment, some of the responses that involve the COAX have been incorporated in the IPME model. CAE expects that there would be no avatar animation related to these responses.

8. The IPME model expects a single contact report from the virtual environment. Hence, if multiple targets appear in the gunner's or CC's sights, the virtual environment will have to provide information on only one of these targets, presumably the one designated by the CC.

9. The IPME model does not handle a change in ammunition at this time. According to the information gathered from the SME, the CC may:

    a. order an ammo type that is already loaded, a situation that is covered by the model;

    b. order an ammo type that is different than the one loaded, in which case the one loaded would be fired and then replaced with the ordered type; or

    c. order the loader to unload the currently loaded ammo type and replace it with a different type.

10. Orders that are not understood by the speech recognition engine are ignored. There is no provision at this time to have the speech synthesis engine utter "say again" or similar sentence.

11. Some orders relating to infrequent engagement situations have not been implemented. Examples are RELASE, TOP EDGE, BOTTOM EDGE, CC ELEVATION, CC LINE, STOP, CARRY ON. Similarly, some gunner's responses have not been implemented, such as WRONG LAY or the reporting of corrections being made between firings.

12. The test bed does not generate gunner-initiated contact information or BDA reports on its own. To generate these reports, the user must use a browser (Chrome, Internet Explorer, or other) and insert the following URL in the address bar of the browser at the appropriate time based on the state of the IPME model:

    a. To generate a gunner-initiated contact report, insert http://localhost:8089/Hosting/setvars/contactType=tank&moveStatus=stat&contactRange=1500 in the address bar. The contactType does not have to be a tank, and the moveStatus can be “stat”, “left” or “right”. The contactRange can be any value from 100 to 10,000.

    b. To generate a BDA report, insert http://localhost:8089/Hosting/setvar/bda=target in the address bar. The bda value can be “target”, “targetDestroyed” or “notObserved”.

3 SUPPORTING ELEMENTS

3.1 Hierarchical Task Analysis

CAE developed a preliminary Hierarchical Task Analysis (HTA) as part of HBR Task 1 (C15-1209-1209 contract report). This preliminary HTA was re-used in HBR Task 2 as a basis for discussions with SMEs in Gagetown in the last week of February 2015. The HTA was reworked, validated, and expanded to cover the scenarios requested for HBR Task 2.

As the IPME model was being developed, several issues were identified and clarifications were obtained from the SME and from the documentation. These issues led to more modifications to the HTA. In its delivered state, the HTA covers all of the tasks present in the IPME model. A few additional processes (e.g. emergency cry of STILL) are also present in the HTA but have not been modeled in IPME.

When navigating the HTA, the user needs to pay attention to the plan for each entry, and to the *triggers task #* and the *triggered by* columns. Together, these three items help the reader understand the flow of actions.

NOTE: As much as possible, CAE avoided repeating identical tasks in the HTA. Nevertheless, some tasks may be found twice in the HTA if the developers assessed that it would help the comprehension of the processes described in the HTA.

3.2 Support to ALSC

Part of the work required for this project was to be carried out by ALSC, namely the development of the virtual environment using Unity3D. CAE was to provide support to ALSC in its development effort, especially regarding the interface requirements between the virtual environment and the IPME model. As the end of HBR Task 2 end neared, ALSC had not contacted CAE to request assistance. It is not clear if any work had been carried out by ALSC one month prior to Task end. As a result, it was agreed in a progress meeting between DRDC and CAE on 3 September 2015 that the funds allocated to the support effort in the Technical Information Package would not be expended and that any required support from CAE to ALSC would be part of a subsequent task.

Nonetheless, to support ALSC's development effort, CAE supplied an ICD to feed the interface development. Additionally, Appendix B – Unity3D Animation describes the animations required for the avatars in the virtual environment and should be used by ALSC as a reference. Finally, the source code for the RESTFul interface implementation could serve as a basis for ALSC's implementation of its own RESTFul interface.

3.3 Design Considerations for the VLeo2 Virtual Environment

Design considerations for the VLeo2 virtual environment are contained in Section 3 of the ICD. CAE recommends that the ALSC developer(s) pay attention to the information contained in that section of the ICD, as it may ease the development and help avoid interface misinterpretations.

3.4 Error Handling

In its delivered state, the IPME model does not incorporate any fault injections and does not handle trainee errors. In preliminary discussions during the visit in Gagetown in February 2015, CAE and DRDC decided that the first instance of the system would assume that all systems were available. However, consideration was given to how fault injection could be incorporated in the model.

The IPME application incorporates a *failure* function inside each networked task. Inside this function, the developer can indicate a probability of failure from 0.0 to 1.0. The probability can be set using several methods: by hard-coding a number, by using a function, or by using a variable containing the number. CAE tested the latter method, i.e. incorporating a variable that takes on a value from 0.0 to 1.0. It was found that this method works well and also offers the possibility of changing the value dynamically. Hence, it would be possible to use that method to allow an instructor to change a probability value dynamically at any time during a training session. The instructor's entry could be sent to the IPME plugin, which would then set the IPME variable to the desired value. The use of a probability function would also be a viable option, and could possibly be used in conjunction with the variable option in which case the variable could be used to influence the function's output.

A preliminary discussion was held with the SME to identify a list of faults and user errors that could be monitored and noticed by the CC. The SME has ascertained that system faults are relatively rare and that when they happen the crew reports them to a technician for repair, especially if the faults occur during the Action Drill. If a fault happens during a mission, the crew follows degraded mode procedures, such as manual firing methods.

User errors are more common and it is part of the CC's responsibilities to monitor the work of his crew members to ensure their safety and that of the tank. The CC will monitor the following errors:

Using the CC's display:

- incorrect range read-out;
- wrong ammunition selection;
- wrong weapon selection;
- incorrect lay on the target;

- wrong target;
- incorrect manual range input; and
- incorrect use of lead locks.

Using visual monitoring:

- loader's unsafe body position;
- wrong ammunition being loaded into the main gun's breech;
- ammunition door left open; and
- some of the switches not being used correctly by the gunner.

All of these errors would require animation in the VLeo2 virtual environment. The CC's display-related errors would all require the addition of error paths in the IPME model, with the exception of *Wrong target* and possibly *incorrect lay on the target* which would be handled solely by the virtual environment. For the visual monitoring-related errors, the IPME model could be modified to generate these errors, although it would also be possible to manage any of these errors directly in the virtual environment. Modification of the IPME model is seen as the preferred option to ensure that the state of the model in relation to the state of the tank's equipment and positioning of the crew members is maintained correctly.

4 CONCLUDING ELEMENTS

4.1 Testing Results

Using the test bed in lieu of the VLeo2 virtual environment, CAE tested the IPME model's integration with the speech engines. To do so, the testers used the document entitled *5746-003 Version 01 Speech Commands and Responses Requirements* to follow the flow of commands and responses that is normally used in the Leopard tank.

When giving orders in the correct order, it was found that the IPME model and speech engines implementation behave properly and provide the expected flow of messages amongst the IPME model and the other components. The messages exchanged with the test bed were found to be in line with the requirements laid out in the document *5746-002 Version 01 Interface Control Document*. Likewise, the IPME variables being set or read by the IPME plugin were in line with the requirements of the same document. The model execution was stable. On rare occasions, it was noted that a response from either the gunner or the loader would be repeated twice. It is likely that these occurrences were due to timing intricacies inside the IPME model, a problem over which the developers have very little control. These occurrences remained rare and are not seen as a road block to successful experimentation with the VLeo2 trainer.

When orders or messages were given in an incorrect sequence, the results varied and depended on the dependencies between the specific order or message and the rest of the model. For example, generating a BDA report before giving a firing order caused the model to flow through the BDA sequence immediately after the firing report from the gunner instead of waiting for a realistic amount of time to give the gunner enough time to carry out the BDA. However, this particular example is not expected to occur with the VLeo2 virtual environment in lieu of the test bed.

More testing will be required once the VLeo2 virtual environment is implemented. In the meantime, CAE is confident that the current implementation meets the requirements of HBR Task 2.

4.2 Conclusion

CAE carried out the work required for HBR Task 2 from December 2014 to September 2015. CAE developed the IPME model to represent the behaviour of the gunner and loader crew members of the Leopard tank. A speech recognition engine and a speech synthesis engine were integrated with the IPME model to accept commands from a CC trainee and to utter responses from both the loader and the gunner. Due to the delay in the development of a virtual environment by ALSC, CAE developed a test bed to duplicate the interactions that are expected to occur between the IPME model and the virtual environment. Finally, CAE documented the system, to include:

- an updated HTA to describe the actions of Leopard crew members when performing the four scenarios specified in the Statement of Work (SOW): action drill, firing from a stationary position, moving between two locations, and firing one the move;

- an ICD to specify the messages exchanged amongst IPME and the other components of the system;

- a commands and responses requirements document to describe the communications amongst the crew members of the Leopard tank when executing the four scenarios specified in the SOW;

- a short system installation and execution document to describe the steps required to install the IPME model, the speech engines and the test bed, and how to run the system; and

- the final report.

CAE tested the system, using the test bed in lieu of the virtual environment. CAE is confident that the produced system meets the requirements of HBR Task 2.

4.3 Recommendations

CAE recommends:

1. That an armoured corps SME be engaged to validate the timings of each task sequence, once the virtual environment is in place. The timings incorporated in the test bed and in the IPME model are based on the discussions held with the SME in February 2015 and on “best guesses”. As such, some of the timings may be unrealistic.

2. That CAE be contracted under a subsequent HBR Task to support the ALSC development and to test the integrated system. This would allow CAE to implement any necessary changes to the IPME model and speech engines, and to answer any queries from the ALSC developer(s).

APPENDIX A ADDITIONAL INFORMATION

A.1 Definitions and Acronyms

The following list identifies the acronyms and abbreviations used throughout this document:

ALSC	Army Learning Support Centre
CC	Crew Commander
CSV	Comma Separated Value
DND	Department of National Defence
DRDC	Defence Research and Development Canada
HBR	Human Behaviour Representation
HTA	Hierarchical Task Analysis
HTTP	Hypertext Transfer Protocol
ICD	Interface Control Document
IPME	Integrated Performance Modelling Environment
JSON	JavaScript Object Notation
LCB	Loader's Control Box
MRS	Muzzle Reference System
REST	Representational State Transfer
RM	REST Module
SAPI	Speech Application Programming Interface
SME	Subject Matter Expert
SOP	Standard Operating Procedure
SOW	Statement of Work
SpSyn	Speech Synthesis
URL	Uniform Resource Locator
VLeo2	Virtual Leopard 2

APPENDIX B UNITY3D ANIMATIONS

B.1 General

For the CC trainee to gain useful training from this system, it will be necessary to implement animations in the Unity3D environment. The animations will support the scenarios identified in the Statement of Work for this Task. This section identifies some of the animations that will be required. Other animations may be identified as work progresses.

B.2 Scenario: Action Drill

B.2.1 Gunner Animations

Table B-1 lists the animations that will be required for the gunner avatar, along with the trigger message from IPME and the value that needs to be sent back to IPME. The exact format of the trigger and completion messages is defined in the ICD. Generic terms are used in the table. Timings for each action should be discussed with SME; however, the HTA may be used for initial settings. Timers may be put in place initially to simulate the time that it would take the gunner to carry out the action and report back to the CC. The implementation priority should be discussed with SMEs and DRDC staff based on the effect on training effectiveness and the experimentation needs. CAE also suggests that the necessity to animate an action or part of an action be discussed with SMEs, as there may be animations that would not be seen by the CC due to the obstruction caused by the gunner's body position.

Table B-1: Required Gunner Animations

Trigger Message	Animation Required	Message to IPME
laserPowerStatus = "turnLaserOn"	Gunner flicks Laser switch on. Indication at CC station changes if applicable.	laserPowerStatus = "laserOn"
E00ButtonPress = "pressE00Button"	Gunner presses E00 button. Indication at CC station changes, if applicable.	E00ButtonPress = "E00ButtonPressed"
thermalSwitchStatus = "check"	Gunner checks the thermal switch.	thermalSwitchStatus = "TIReady"
ammoSelStatus = "selectMain"	Gunner switches the Ammo Selector	ammoSelStatus = "MainSelected"
gunLevelStatus = "levelMainGun"	Gunner levels the gun	gunLevelStatus = "mainGunLeveled"
bowdenCableStatus = "checkBowdenCable"	Gunner checks the Bowden cable	bowdenCableStatus = "bowdenCableOff"
driftStatus = "eliminateDrift"	Gunner eliminates the drift	driftStatus = "driftEliminated"
MRSStatus =	Gunner verifies the Muzzle Reference	MRSStatus =

Trigger Message	Animation Required	Message to IPME
"checkMRS"	System (MRS)	"MRSChecked"
markArc = "leftHandMarker"	Gunner looks through his sights. Note: VLeo2 must record the left hand marker position.	markArc = "leftMarked"
markArc = "rightHandMarker"	Gunner looks through his sights. Note: VLeo2 must record the right hand marker position.	markArc = "rightMarked"
markArc = "centreOfArc"	Gunner looks through his sights. Note: VLeo2 must record the centre marker position.	markArc = "centreMarked"
scanDirection = "scanLeft"	Gunner selects day sight, looks through his sights continuously and moves the controls so that the main gun scans from the centre arc to the left arc and back, continuously. Turret moves from centre arc to left arc and back, continuously.	(No message is sent back to IPME)
scanDirection = "scanRight"	Gunner selects day sight, looks through his sights continuously and moves the controls so that the main gun scans from the centre arc to the right arc and back, continuously. Turret moves from centre arc to right arc and back, continuously.	(No message is sent back to IPME)

B.2.2 Loader Animations

Table B-2 lists the animations that will be required for the loader avatar, along with the triggers. The same comments as for the gunner animations apply to the loader.

Table B-2: Required Loader Animations

Trigger Message	Animation Required	Message to IPME
turretPowerState = "turnTurretOff"	Loader turns off the turret power switch	turretPowerState = "turretOff"
mainGunSafeStatus = "makeSafeMainGun"	Loader makes safe the main gun on the LCB	mainGunSafeStatus = "mainGunSafe"
mainGunClearance = "clearMainGun"	Loader clears the main gun	mainGunClearance = "mainGunCleared"
coaxClearance =	Loader proves the COAX.	coaxClearance =

Trigger Message	Animation Required	Message to IPME
"clearCoax"	Note: as the COAX is not modelled in the initial version of the VLeo2, replace the animation with a timer.	"coaxCleared"
freeToTraverseState = "checkFreeToTraverse"	Loader checks that the turret is free to traverse	freeToTraverseState = "freeToTraverse"
turretPowerState = "turnTurretOn"	Loader switches on the turret power switch	turretPowerState = "turretOn"
loadSabotMakeSafe = "loadSabot"	Loader loads a sabot in the main gun, then loads the COAX	loadSabotMakeSafe = "sabotLoaded"
loadSabotMakeSafe = "makeSafeMainGun"	Loader presses the safe button on the LCB	loadSabotMakeSafe = "mainGunSafe"

B.3 Scenario: Target Engagement

B.3.1 Gunner Animations

Table B-3 lists the gunner avatar animations required to carry out the gunner engagement scenarios:

Table B-3: Required Gunner Animations for Target Engagement

Trigger Message	Animation Required	Message to IPME
selectLeadLocks = "selectOff"	Gunner deselects the lead locks.	selectLeadLocks = "leadLocksOff"
selectLeadLocks = "selectOn"	Gunner selects the lead locks.	selectLeadLocks = "leadLocksOn"
fire = "gnrFire"	The gunner fires the main gun. Gunner then looks through his sights to perform battle damage assessment.	fire = "gnrFiringNow" followed by bda = "targetDestroyed" / "target" / "notObserved"
E00ButtonPress = "pressE00Button"	Gunner presses the E00 button.	E00ButtonPress = "E00ButtonPressed"
ammoSelStatus = "selectMain"	Gunner switches the ammo selector to the Main position.	ammoSelStatus = "MainSelected"
fireLaser = "fireLaser"	Gunner fires the laser.	fireLaser = "laserFired" range = <range>

B.3.2 Loader Animations

Table B-4 lists the loader avatar animations required to carry out the gunner engagement scenarios:

Table B-4: Required Loader Animations for Target Engagement

Trigger Message	Animation Required	Message to IPME
loadSabotMakeReady = “loadSabot”	Loader loads a Sabot in the main gun and selects SABOT on the LCB. Correct indication appears on the LCB repeater at the CC station.	loadSabotMakeReady = “sabotReady”
loadSabotMakeSafe = “loadSabot”	Loader loads a sabot in the main gun, then loads the COAX	loadSabotMakeSafe = “sabotLoaded”
loadSabotMakeSafe = “makeSafeMainGun”	Loader presses the safe button on the LCB	loadSabotMakeSafe = “mainGunSafe”