APPLICATION OF COMMERCIAL OFF-THE-SHELF GAME ENGINES FOR SAF EXPERIMENTATION

LIM Chee Hong, TAN Chee Ann, LOKE Yu Juan, BOO Tai Yi

ABSTRACT

Commercial Off-The-Shelf (COTS) game engines are gaining traction among developers due to their low learning curve for beginners, wealth of readily available assets and plugins, and their ability to integrate with the fast growing virtual reality and augmented reality devices. DSTA has leveraged COTS game engines in recent years to develop Modelling and Simulation (M&S) environments for military experimentation.

This paper outlines how DSTA has adopted and integrated a COTS game engine, which is based on open standards, as part of its M&S environment to support large-scale, distributed and short-cycle experimentations.

Keywords: COTS, game engine, simulation, experimentation

INTRODUCTION

The DSTA Analytical Lab has been around for several years, supporting the Singapore Armed Forces (SAF) in the experimentations of new warfighting concepts and future technologies since its inception in 2006. A Modelling and Simulation (M&S) environment has been developed in-house in DSTA for the conducting of such experiments. The key architectural considerations of the M&S environment are configurability, scalability and interoperability, to ensure that the environment could be configured and scaled for use across different experimentation scenarios and complexities, and for interoperation with external simulators and Command and Control (C2) systems.

With the increased demand for short cycle and quick system prototyping experimentations, there is motivation to shorten the application development cycle by adopting readily available Commercial-Off-The-Shelf (COTS) products, such as COTS game engines, with additional in-house development to integrate the COTS game engine as part of the M&S environment, fulfilling the needs for configurability, scalability and interoperability.

MODELLING AND SIMULATION ENVIRONMENT

The M&S environment is made up of the following subsystems:

- **Simulation Management** – This is used by the experiment controllers to effect the simulated environment which mimics the real operations environment, and to control the scenario flow during an experiment run. Visualisation and interaction tools are provided for real-time scenario monitoring; scenario controls such as load, start and stop experiment runs; dynamic creation of inject to effect changes to a simulated scenario; and control of simulated entities besides the human subjects of the experiment. In order to minimise manpower requirements, experiment controller features are designed to be scalable for one-to-many controls.

- **Simulation Models** – These represent the entities, called Computer Generated Forces (CGF), that are modelled in the simulation environment and they are largely categorised into two types - physical and behavioural models.

- A physical model is a physics, mathematical or otherwise logical representation of a system, entity, phenomenon, or process (Page & Smith, 1998).

- A behavioural model is a collection of models that represent the doctrine, workflows and behaviours of simulated entities. They define how a simulated entity may interact with other entities in a simulated scenario by controlling the simulation models of each entity.

- **Simulation Visualisation** – This provides the means for experiment subjects to view and interact with the simulated environment during experiments. It is made up of Image Generators and consoles/emulators.

  - Image Generators provide a set of correlated 2D and realistic 3D visualisations of the simulated scenario to experiment subjects. Examples include 2D maps for operator consoles, simulated feeds from sensor payloads and 3D representations simulating Out-The-Window views of platforms.

  - Consoles/Emulators are collections of Man Machine Interface (MMI) consoles and hardware control (i.e. joysticks, touchscreens and headsets) that sufficiently emulate the operating environment required for experimentation. MMI consoles are integrated with Image Generators in order for experiment subjects to interact with the virtual scenario. Examples of MMI consoles include mock-up consoles with envisaged future technologies or existing C2 systems. Virtual reality (VR) and augmented reality (AR) devices are also increasingly being evaluated to enhance the realism of the consoles/emulators.

- **Measurement & Analysis** – This is an important subsystem that supports experiments to capture Measure of Performance. This consists of two components – Data logger and Data visualisation.

  - **Data Logger** – This layer comprises modules for simulation data recording. Examples of captured simulation data include simulation events, video and audio logs.

  - **Data Visualisation** – This layer comprises modules for intuitive data visualisation to facilitate data analysis, synchronised playback to support debriefing during After Action Review, and data verifications.

- **Simulation Engine** – This is the underlying core engine that coordinates and harmonises interactions among various other subsystem components. Specifically, it

  - Keeps track of time and coordinates the execution of processes across all simulation components.

  - Provides a set of Application Program Interfaces (API) for various simulation components to interact with each other. The APIs also provide means for existing simulation components to be extended to include new features.

  - Handles information exchanges across all components and with external systems, such as emulators, C2 systems or Original Equipment Manufacturer (OEM) simulators, via a common data exchange module.

COTS GAME ENGINE

Game engines are development environments designed for the creation of video games. The core functionalities typically provided by a game engine include rendering engine for 2D/3D graphics, a physics engine, audio, artificial intelligence, networking, streaming and memory management. These are key components in the simulation engine, and hence the motivation to adopt COTS game engines as part of the M&S environment. The ability of COTS game engines to integrate with VR and AR devices provides an additional incentive for their adoption.

There are many game engines[1] available in the gaming industry of which the major player is Unity with approximately 48% of the market share. Unreal Engine is a distant second with approximately 13%. To minimise the development timeline for the support of short-cycle experimentations, key selection criteria of the game engine includes conformance to open standards to ensure no vendor lock-in; market share and popularity to ensure abundance of developmental support and documentations from its large market base; and minimal additional development work required to integrate the game engine into the M&S environment.

Unity is assessed to sufficiently fulfil all the above critieria. It is one of the few engines that caters effective development support for both 3D and 2D games[2]. On top of that, it has an abundance of online training resources, documentation and large community of active users available for assistance.

Unity, together with third party tools available in the marketplace, will give developers a good head start in its adoption. However, there are constraints on scalability such as the ability to scale for the support of a larger number of entities or bigger Area of Operation (AO). Out of the box, it also does not interoperate with external systems that the simulation engine would need to integrate with. There are furthermore limitations in terms of configurability as third party plugins and tools are often closed in nature, with no source codes available for developers to customise according to individual experiment needs.

There is thus a need for additional development work to integrate Unity into the M&S environment, to fulfil the architectural needs for scalability, configurability and interoperability.

INTEGRATING UNITY

The project team adopted a three-step strategy to integrate Unity into the M&S environment.

Step 1: Adopt Game Engine Out-of-the-box with Minimal Modifications

The basic anatomy of Unity is illustrated in Figure 1.

Middleware Software Development Kits (SDK) – This layer includes many third party software libraries specific to various platforms. For the Windows platform, this would include graphics libraries such as Microsoft DirectX and OpenGL.

Core Systems – This layer contains the simulation kernel and general functionalities such as memory allocation, mathematics library and subsystem life cycle management that are used across all subsystems. In addition, this layer also provides a platform-independent abstraction to the middleware SDKs and computing platform, allowing applications built with the engine to be cross-platform, supportable across different operating systems, and work optimally with different hardware.

Resources Manager – This layer handles all the data assets required for a simulation application. This may include loading and unloading of 3D Models for 3D visualisations, 2D icons for user interfaces and physics parameters for collision simulation.

Common Subsystems – This layer contains many modular subsystem components such as audio, graphics, animation, physics and collision, and human input devices that support interfaces with keyboard, mouse and game controllers. Many of these subsystems can be integrated into a simulation environment in short timeframes for quick prototyping.

These software layers make Unity a very capable game engine in out of the box consideration. With these features, it is already possible to implement simple standalone prototypes that can be used for quick concept demonstrations or rapid prototyping. It is advantageous to minimise customisations on these existing features to allow the adopted game engine to stay up-to-date with the latest version from Unity.

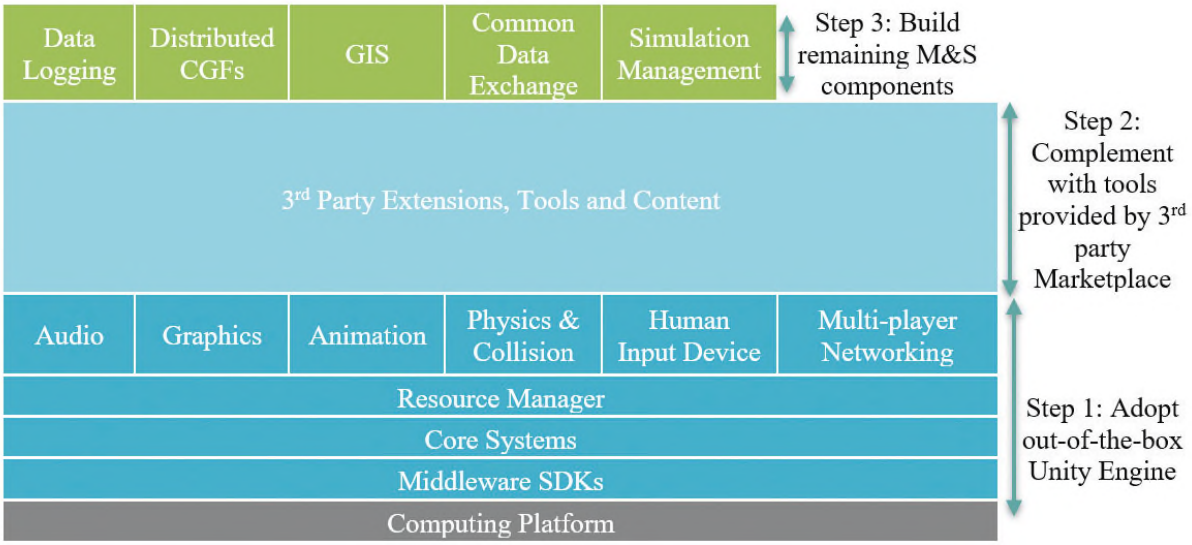

Figure 1. A three-step strategy to integrate Unity

Step 2: Complement with Curated Third Party Extensions

Another key strength of Unity is its large user base and the contributions of a large quantity of user-developed third party assets available on its online marketplace. Third party assets may include fully optimised and ready-to-use 3D models with animations or open source subsystem extensions to introduce new functionalities to Unity. Often, the better developed third party assets stand a good chance to become mainstream functionalities of Unity. One such example is TextMesh Pro, which was originally a third party extension for dynamic font resolution for graphical user interfaces. Its popularity and reliability led it to be acquired by Unity and incorporated directly into its game engine as a core feature. The team's strategy for supporting short-cycle experimentations is also to tap the pool of third party assets for additional functionalities that are not yet available in Unity, and integrate them into the M&S environment. They were carefully curated and selected based mainly on the availability of source codes for modification and customisation.

Step 3: Build Remaining M&S Components for a Full M&S Environment

Although Unity is a very capable game engine, it does not contain all functionalities required for the full M&S environment. For example, it is not able to import and consume common Geospatial Information System (GIS) data formats directly to generate 3D terrains for visualisation. Thus, additional development work needs to be done to create the Unity instances, and for the remaining M&S components to integrate the Unity instances to fulfil the architectural needs for configurability, interoperability and scalability.

Configurability – Configurability is the ability to support a wide range of experimentation scenarios in the M&S environment. Scenario simulation should be easily created, configured, controlled, monitored and maintained. One key characteristic of experimentations is that they often require the environment to be configured based on experiment objectives and the subjects of experiments. Having a configurable engine will enable the deployment of different simulation set-ups to be much quicker and requiring less effort.

This design consideration is addressed in the simulation management subsystem, which manages the configuration and takes care of the scenario-loading of Unity instances. This subsystem performs load distribution based on a configuration file that states the entities and the specific machines that they will be running in. With this, scalability is also achieved. As illustrated in Figure 2, a configuration file is created in the experiment controller module and synchronised by sharing it with all client machines.

Another aspect of configurability is the ability to support various terrain types required for experimentation quickly. There is also the need to facilitate correlation of terrain information across both subsystems and external systems. To achieve this, a GIS module was developed on top of open source libraries. This module consumes and translates GIS data to create the synthetic environment. Other than generating the 3D terrain for visualisation, the ingested data also allows CGFs to perform terrain reasoning during simulation for navigation, collision avoidance as well as other behaviours that require interaction with the surroundings.

Interoperability – Interoperability allows the M&S subsystems to integrate seamlessly with external simulators, C2 systems and OEM emulators, to offer a plug-and-play, mix-and-match M&S environment.

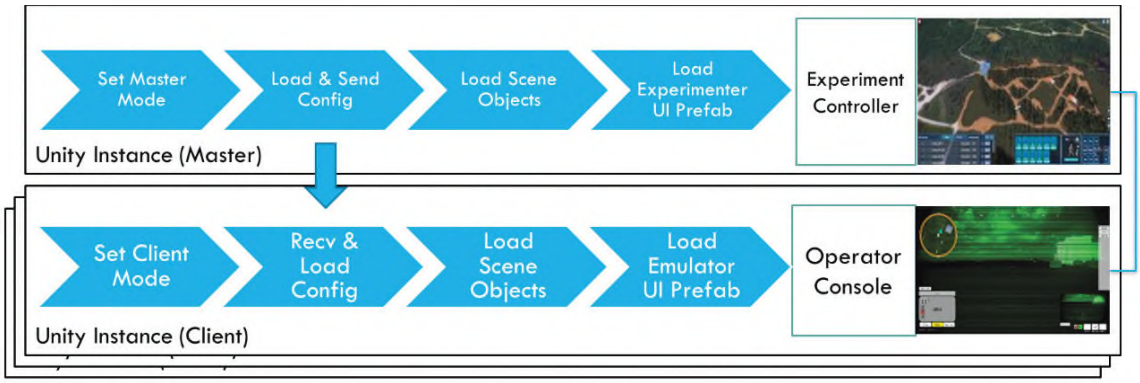

Figure 2. Scenario Management Module

Interoperability is addressed in the common data exchange module, which is responsible for 1) information exchange within and across Unity Instances, and 2) data exchange with external simulators, C2 systems as well as OEM emulators. As illustrated in the Figure 3, every Unity instance contains a common data messenger. This module takes care of two broad types of messages – cyclic messages and events. Cyclic messages are for states updates and synchronisation on every time tick of the simulation, while events handle asynchronous messages that happen occasionally.

Scalability – Scalability is the ability for an M&S environment to support and meet demands of increasing scenario complexity, such as larger number of entities and larger AO without drops in simulation performance. However, it is worth noting that in reality, no software scales indefinitely due to the limits imposed by network infrastructure and computing resources.

To cater for scalability, the CGF entities in the M&S environment have been designed to be distributed across several machines during simulation. In addition, the CGF entities are developed based on a master and slave concept in which a CGF will have an owner responsible for simulating the behaviour of the CGF it owns and synchronising with other Unity instances within the simulation. Figure 4 illustrates the distributed CGF architecture.

Open Standards – As much as possible, open standards were adopted to prevent vendor lock-in. In this manner, modules developed will have better reuse value and more interoperability with other engines.

Implementation on the game engine was based on open standards as far as possible. The model data structure is based on High Level Architecture (HLA)[3] Simulation Interoperability Standards Organization (SISO)[4] Real-time Platform Reference Federation Object Model (RPR FOM)[5] 2.06 to support simulation-to-simulation integration with other simulations that are HLA-compliant. Other than supporting simulation-to-simulation interoperability, the M&S environment was also extended with a gateway to support simulation-to-C2 integrations.

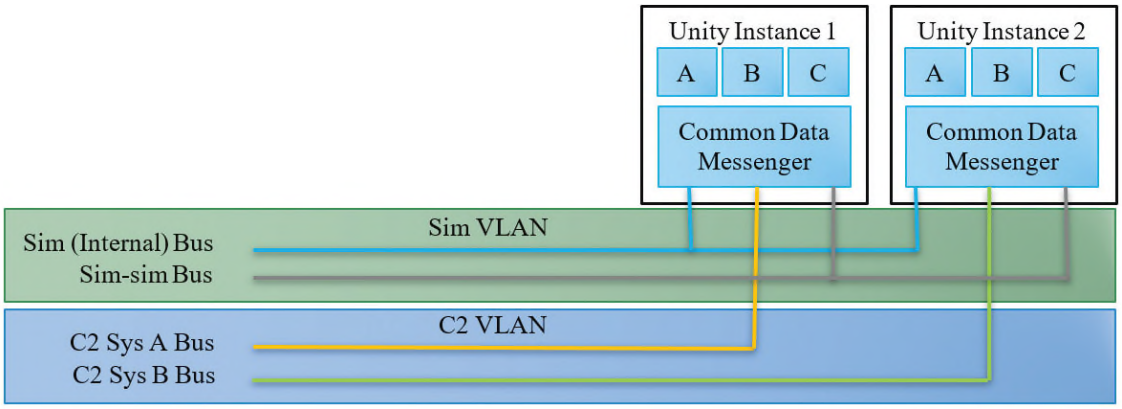

Figure 3. Common data exchange module

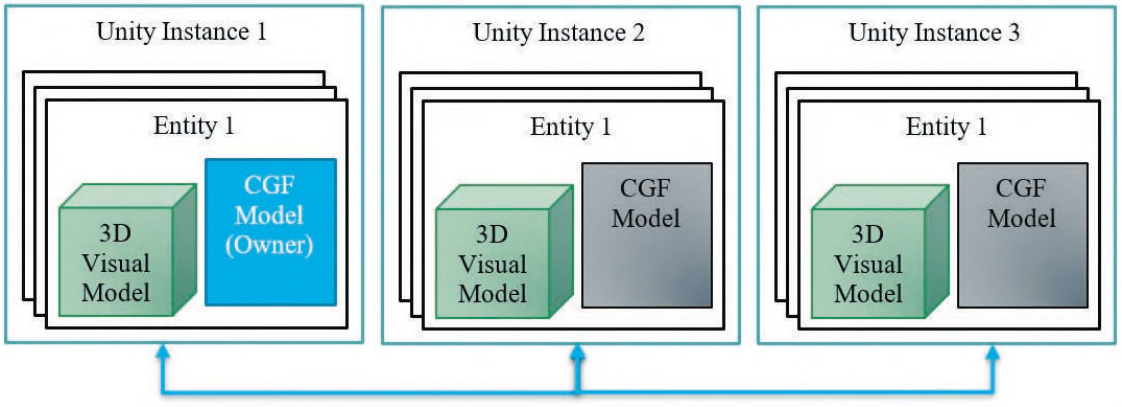

Figure 4. Distributed CGF architecture

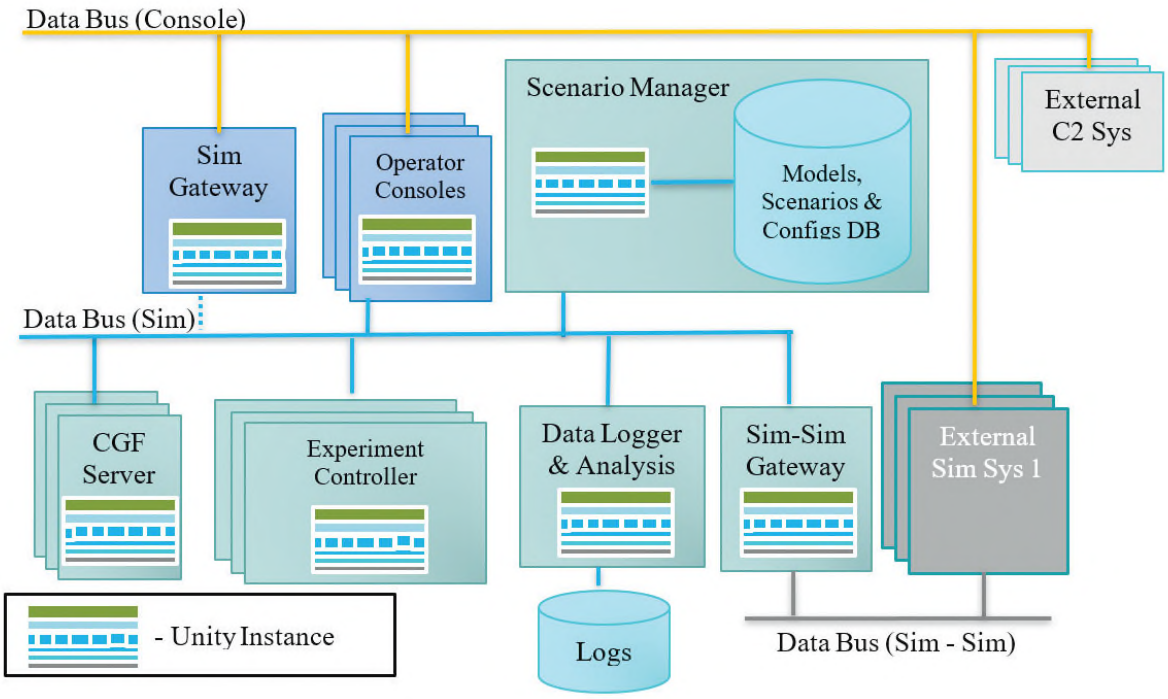

Figure 5. Example of complex environment

The Unity integrated M&S environment has been used to support several SAF experiments. Figure 5 illustrates a full set-up for a complex scenario during experimentation. A Unity instance can be configured to serve different purposes such as emulators for different operators, gateways, CGF server, experiment controller, data logger, etc.

CONCLUSION

The game engine has been successfully integrated to achieve configurability, scalability and interoperability for the support of large-scale[6], distributed and short cycle experimentations. In the course of using Unity to support a few experimentations, several simulation models were created. The team will continue to adopt open standards while creating simulation models, which will contribute to the repository of models ready to be reused in subsequent experiments. Moving ahead, exploration will continue in the area of large terrain database covering, not only with a larger AO but also different types of AO such as open sea, under water and even build-up areas consisting of indoor operations. Unity being a popular game engine and active with a huge community, the software is bound to have very frequent upgrades. The pipeline for upgrades has to be worked out so as to ensure smoother porting of models built to maximise the value of reuse. Although Unity is the current choice of engine, the team shall continue to keep a lookout for other potential game engines.

REFERENCES

High-level architecture. (n.d.). In *Wikipedia*. Retrieved February 25, 2019, from https://en.m.wikepedia.org/wiki/High-level_architecture

Instabug Blog. (2019). *Top game engines in 2019.* Retrieved from https://instabug.com/blog/game-engines/

Page, E. H. & Smith, R. (1998). *Introduction to military training simulation: A guide for discrete event simulationists.* Paper presented at 1998 Winter Simulation Conference. Washington, DC, USA. doi: 10.1109/WSC.1998.744899

Penev, V. (2003). Advances in modeling and simulation, information and security. *Information and Security, 12*(1), 5-18. Retrieved from https://www.researchgate.net/publication/275956250_Advances_in_Modeling_and_Simulation_Information_and_Security_Volume_12_Number_1

Simulation Interoperability Standards Organization (SISO). (2019). Retrieved February 25, 2019 from https://www.sisostds.org

Slant. (2018). *Unity vs unreal engine 4.* Retrieved from https://www.slant.co/versus/1047/5128/~unity_vs_unreal-engine-4

VT MAK. (2017). *RPR FOM – Real-time platform-level reference federation object model.* Retrieved February 25, 2019 from https://www.mak.com/learn/industry-standards/378-rpr-fom-real-time-platform-level-reference-federation-object-model.html

Winter, B. (2017). *TextMesh pro joins unity.* Retrieved from https://blogs.unity3d.com/2017/03/20/textmesh-pro-joins-unity/

ENDNOTES

[1] Some more popular game engines include Unreal Engine, Unity, Godot Engine, CryEngine, Marmalade SDK, AppGameKit, Cocos2d-x, MonoGame, Amazon Lumberyard, ShiVa Engine and HeroEngine.

[2] This is an important consideration as it significantly reduces the time and effort required for the additional development work on 2D-3D correlation, integration and testing.

[3] The High Level Architecture (HLA) is a standard for distributed simulation, used when building a simulation for a larger purpose by combining (federating) several simulations.

[4] The Simulation Interoperability Standards Organization (SISO) is an international organization dedicated to the promotion of modelling and simulation interoperability and reuse for the benefit of a broad range of M&S communities including developers, procurers, and users world-wide.

[5] The Real-time Platform Reference Federation Object Model (RPR FOM) is a reference FOM that defines HLA classes, attributes and parameters that are appropriate for real-time, platform-level simulations.

[6] The engine with the designed architecture was deployed and had supported an experiment with up to 20 participants and more than 300 CGFs.

BIOGRAPHY

LIM Chee Hong is a Senior Development Programme Manager (C3 Development) overseeing the development of modelling and simulation systems in support of SAF experimentation. Chee Hong graduated with a Bachelor of Applied Science (Computer Engineering) Degree from Nanyang Technological University (NTU) and a Master of Technology (Knowledge Engineering) Degree from the National University of Singapore (NUS), in 1998 and 2003 respectively.

TAN Chee Ann is a Development Programme Manager (C3 Development) managing a development team in delivering modelling and simulation systems in support of SAF experimentation. Chee Ann graduated with a Bachelor of Computing in Communications and Media (B. Comp Hons) Degree and a Master of Technology (Software Engineering) Degree from NUS, in 2008 and 2014 respectively.

LOKE Yu Juan is a Senior Engineer (C3 Development) designing and developing modelling and simulation systems in support of SAF experimentation. Yu Juan graduated with a Bachelor of Engineering (Information Engineering & Media) Degree from NTU in 2012.

BOO Tai Yi is a Senior Engineer (C3 Development) developing modelling and simulation systems in support of SAF experimentation. Tai Yi graduated with a Bachelor of Engineering (Computer Engineering) Degree from NUS in 2017.