SOFTWARS

Bridget Free, Andrew Russell, Do Tran and Tate Waters

Functional Specification

Monday March 30, 2020

Table of Contents

Purpose

The purpose of this document is to outline the intended development, capabilities and interactions of the system - LVC integration with Unity and DIS middleware. This document explains the high-level technical and functional requirements of this system, and provides information about the roles and responsibilities needed to support it.

Glossary

LVC: Live, Virtual & Constructive simulation is a broad term used to classify models and simulations that involve real people operating real and simulated systems.

> **Live** refers to real people operating real equipment. **Virtual** refers to real people operating computer-simulated equipment (or real equipment in training mode). **Constructive** refers to simulated, non-real entities that are added to an ongoing virtual simulation (e.g. red-team NPCs).

DIS: Distributed Interactive Simulation is a standard networking protocol for exchanging information among simulation applications. It defines a set of Protocol Data Units for publishing information about the simulation.

HLA: High Level Architecture is a recent standard for interoperability among simulation. It defines an architecture with a set of API.

IEEE 1278 Standard: The standard for Distributed Interactive Simulation used to conduct networked simulation exercises worldwide.

Description and Scope

This Functional Specification Document outlines the project's: description; goal and scope; risks and assumptions; use cases; functional and nonfunctional requirements; and relevant diagrams.

The Unity DIS system will:

- Demonstrate that a specialist system can be linked to a virtual simulation environment to improve the simulation
- Adhere to the IEEE Standard for Distributed Interactive Simulation regarding PDU Entity States and Entity State Updates

The scope of this system includes:

- The initial implementation of multiple simulated behaviour models in Unity
- Configured middleware that can encode DIS traffic and broadcast it across a network
- Using visualisation environments (such as VBS) that can accept broadcasted PDU entity data and map the entities to entities in their own environments, and implement updates to these entities in real time.

This does not include:

- Creating a new middleware platform for the encoding and broadcasting of DIS traffic*
- Configuring individual simulation environments to receive DIS traffic

See Annex A for a diagrammatic depiction of the system and the scope of our product.

*There is a risk that this will become necessary; see risks on page 4

Background and Overview of Distributed Simulation

Simulations are interactive through state-of-the-art communication systems and are environments that can be distributed across multiple computers, potentially at different locations. However, there is a complex reality to all simulations as there are many different standards for interoperability. Issues also arise including security issues, legacy systems and different data and voice communications. There are various established frameworks for the connection of simulations, including HLA and DIS.

DIS Design Principles

- Entity encapsulation in simulations
- Autonomous simulation nodes
- Transmission of both “ground truth” and state change information
- Use of “dead reckoning” algorithms to extrapolate entity states

DIS Individual Limitations

- The use of best effort, i.e. UDP, underlying the bulk of the protocol means that DIS usually has to use other features to ensure delivery of data required to maintain exercise correctness, e.g. heartbeats
- The use of real time PDU transmissions means that the exercise can’t run faster during “slow” periods in the exercise.

DIS Interoperability Limitations

- Since the data model is in the standard, exercise semantics are limited to the standard
- Using experimental PDUs runs the risk of interoperability issues with other DIS-configured simulations
    - Simulations must be using the same version of the standard and / or the same PDU formats to avoid this issue
- Simulations must be using the same enumerations or they will misinterpret PDUs, potentially representing entities incorrectly.

Key Documents and References:

- IT Project 1 - Statement of Work
- IT Project 1 - Resource Log
- IT Project 1 - Client and Team Meeting Notes

Risks and Assumptions

Assumptions

- Operator Knowledge
    - It is assumed that all users will have some basic knowledge of the visualisation environment (simulator) that is receiving the DIS traffic (e.g. VBS)
    - Users will be able to ensure their visualisation environment is configured to receive the DIS PDUs being broadcasted via the middleware
- Resources
    - All required resources (such as Unity and VBS) for the system that are not publically available will be made available through UNSW Canberra

Risks and Mitigation Strategies

- A suitable open source middleware implementation of DIS (in the preferred language of c sharp) may not be found in time to deliver the final product
    - If this transpires, a non-c sharp implementation of DIS will be used and a translation between that language and c sharp will also be developed
- Labs at UNSW Canberra will become unavailable for the duration of the project due to COVID-19 restrictions, making VBS unavailable to test the PDUs being sent
    - As the virtual environment is only for use as a visualisation tool, it is non-crucial. Instead, Wireshark may be used to view all PDU data being broadcast to the network
    - Alternatively, other openly available virtual environments (such as Arma) may be used as substitutes to VBS and paid for by UNSW if need be

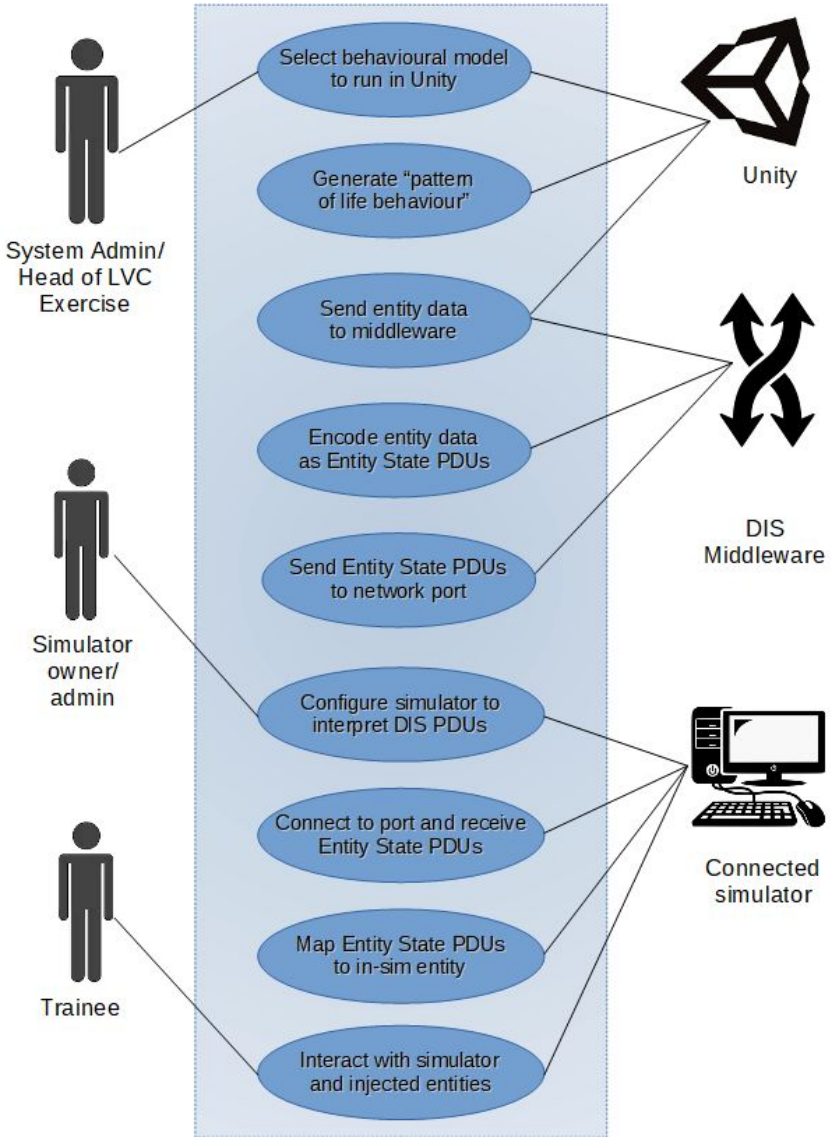
System Definition

This project will inject behavioural models of varying fidelity through a DIS middleware into a visualised environment. The three core pieces of the product are: the behavioural algorithms, a DIS-configured middleware and any arbitrary connected virtual visualisation environment (simulator). The main functionality of the product is to inject PDU entity state data into a broadcast network in order to populate connected simulations with “pattern of life” behaviour.

The intended users of this system include military personnel involved in LVC training exercises and any organisation or individual that wants a virtual environment to be populated with “real time”, pattern of life behaviour to improve its simulation.

Competition for our product includes Calytrix Technologies and the LVC Game integration library they built that demonstrated bi-directional communication between Game Engines (Unity) and Military Simulations (VBS2) using the DIS standard.

Use Case Diagram



Functional Requirements

Every entity generated in Unity must be simulated in all connected simulation environments

Entity state changes - mostly movements- in Unity must be updated in all connected simulation environments in real time

Updates to the state of all simulated objects should be smooth

Objects should not be controlled but should freely move, following behavioural model(s)

Non-Functional Requirements

User Requirements

Users should not have overall gameplay impacted by injected behaviour - just enhanced. Possible addition (dependent on time): user behaviour should have an effect on the injected objects' behaviour

Interface Requirements

Standardised format of network messages with no formal programming API - implementation dependent on requirements (C# because of Unity).

Network sends messages defined by DIS utilising a TCP/IP system.

Hardware/Software Requirements

This product is being built for use on a Windows* system capable of running Unity (Personal/free) and the OpenDIS middleware (most likely implemented in c#).

Any simulators intending to receive DIS PDU traffic must be able to connect to the chosen port via TCP/IP and must be configured to receive PDUs and map them to entities within their own simulation environment. Configuring these simulators is beyond the scope of this project.

*However, there is no reason why this product should not work on a Linux system.

Operational Requirements

- System Availability

The system will be available at all times when the DIS Gateway is activated and both Unity and the visualised environment are connected to the same network. However, the simulator must ensure it is connected to the correct port to receive PDU data.

The system must not drop and regular updates must be sent through the DIS middleware.

- Capacity

There are no capacity limitations other than those limitations from the programmer and system itself. For example, the capacity for behaviour models is limited to the capability of the programmer, and the amount of data that can be received by the simulator is limited to network potential.

- Error Handling

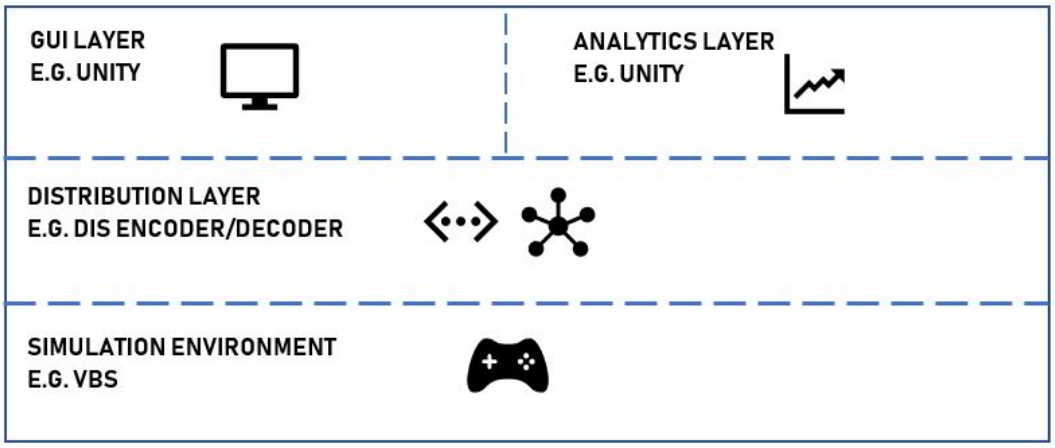
Error detection is handled by the DIS Gateway with the correct formatting of PDU data. This data is then simply interpreted by the visualisation environment (VBS).

- Conventions/Standards

As mentioned earlier, the Distributed Interactive Simulation (DIS) standard from IEEE must be adhered to.

Annex A: Simple Model and Scope Diagram

Concept of "Pattern of Life" Environment



Scope Diagram