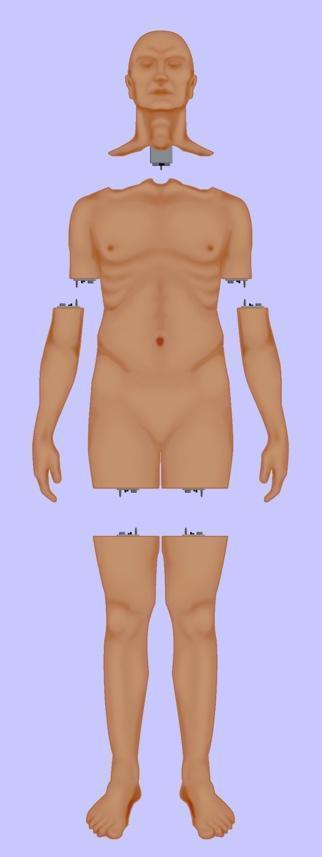
CDRL A001

Software Design Description (SDD) for

Advanced Modular Manikin Project

Phase II Program

Contract # W81XWH-14-C-0101



Date: February 3, 2020

Revision: Rev-2

Unclassified

PRINCIPAL INVESTIGATOR: ROBERT M. SWEET, MD, FACS

REGENTS OF THE UNIVERSITY OF MINNESOTA

OFFICE OF SPONSORED PROJECTS

200 OAK ST SE

MINNEAPOLIS MN 55455-2009

This document is unclassified, and APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

.

Revision History

|  |  |
| --- | --- |
| Date of Revision | Description of Changes |
| 24 Dec 2018 | Original Draft Submitted |
| 26 Aug 2019 | Document Submitted |
| 15 Nov 2019 | Revised Document Submitted |
| 03 Feb 2020 | Distribution Updated |

**Distribution Statement**

This document was created under contract funding W81XWH-14-C-0101 from:

**DEPARTMENT OF THE ARMY**

**US ARMY MEDICAL RESEARCH ACQUISITION ACTIVITY**

**820 CHANDLER STREET**

**FORT DETRICK MD 21701-5014**

It contains no proprietary information, trade secrets, copy righted material or classified information. This document is unclassified, and APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

Table of Contents

[1. Scope 5](#_Toc24720285)

[1.1 Identification 5](#_Toc24720286)

[1.2 System Overview 6](#_Toc24720287)

[1.3 Document Overview 6](#_Toc24720288)

[2 Referenced Documents 7](#_Toc24720289)

[2.1 Industry Documents 7](#_Toc24720290)

[2.2 Government Documents 7](#_Toc24720291)

[2.3 Related Contract Documents 7](#_Toc24720292)

[3 CSCI – Wide Design Decisions 7](#_Toc24720293)

[4 Architectural Design 8](#_Toc24720294)

[4.1 Data Architecture 8](#_Toc24720295)

[4.2 Data Distribution System (DDS) Common Data Bus 10](#_Toc24720296)

[**4.2.1** **DDS Selection** 11](#_Toc24720297)

[4.3 User Interfaces 11](#_Toc24720298)

[4.4 Computer Software Configuration Items (CSCIs) 11](#_Toc24720299)

[4.5 Hardware Configuration Items (HWCIs) 13](#_Toc24720300)

[**4.5.1** **AMM Development Kit (AMMDK)** 13](#_Toc24720301)

[**4.5.2** **AMMDK Application Board** 15](#_Toc24720302)

[**4.5.3** **AMM Network Controller (AMMNC)** 15](#_Toc24720303)

[4.6 Interface Design 16](#_Toc24720304)

[5 Detailed Design 16](#_Toc24720305)

[5.1 AMM Data Model Design 16](#_Toc24720306)

[5.2 Simulation Data 17](#_Toc24720307)

[**5.2.1** **Event Data** 17](#_Toc24720308)

[**5.2.2** **Event Records** 18](#_Toc24720309)

[**5.2.3** **Simulator Data** 21](#_Toc24720310)

[**5.2.4** **Configuration Data Model** 23](#_Toc24720311)

[**5.2.5** **Module Behavioral Requirements** 25](#_Toc24720312)

[5.3 AMM Standard Library Design 26](#_Toc24720313)

[5.4 AMM CORE Module Design – Required Modules 27](#_Toc24720314)

[5.5 AMM CORE Module Design – Extended Core Modules 28](#_Toc24720315)

[5.6 AMM Reference Modules 29](#_Toc24720316)

[5.7 AMMDK Development Environment Design 29](#_Toc24720317)

[5.8 AMM User Interface Design 31](#_Toc24720318)

[6 Requirements Traceability 31](#_Toc24720319)

[7 Notes 31](#_Toc24720320)

[8 Appendix 1: Acronyms 32](#_Toc24720321)

[Figure 1: Functional Overview of AMM Platform 5](#_Toc17698753)

[Figure 2: AMM Processing Architecture Based on DDS Common Data Bus 9](#_Toc17698754)

[Figure 3: AMMDK-CCB Architecture 14](#_Toc17698755)

[Figure 4: AMMNC Enhancements to AMMDK-CCB 16](#_Toc17698756)

[Figure 5: Compound event data flow 21](#_Toc17698757)

[Figure 6: Module Manager Functions 27](#_Toc17698758)

[Figure 7: AMM Web Application Data Flow 28](#_Toc17698759)

[Figure 8: AMMDK Architectural Overview 30](#_Toc17698760)

[Table 1 Requirements Traceability Matrix 31](#_Toc17470235)

# Scope

This document defines the standards for 1.0 release of the Advanced Modular Manikin (AMM) platform and its formal deliverables. The formal deliverables consist of the platform specification, an open source\* Reference Implementation (RI) of the Computer Software Configuration Items (CSCIs), a reference implementation of the Universal Segment Connector (USC) and other hardware defined by the Hardware Configuration Items (HWCIs), the data models that ensure interoperability between the core and modules, and the documents that describe their design, operation, and extensibility through the addition of AMM Modules. Modules are defined as independent building blocks that provide incremental capabilities to the core or provide training opportunities for different medical and trauma related conditions. The focus of this specification is on the platform, a much broader definition than a physical manikin, as illustrated in Figure 1, and on how it can be extended by medical simulation developers by adding:

* Modules that provide incremental capabilities to the core, including authoring tools, after action review tools, different physiology engines.
* Modules that add training opportunities, including IV/IO arms, intubation heads, laparotomy abdomens, virtual stethoscopes. These can be physical, virtual, or hybrid part task trainers.

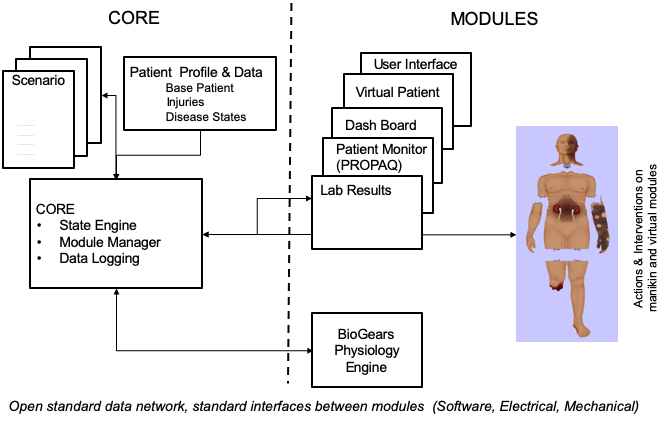


Figure : Functional Overview of AMM Platform

## Identification

This Advanced Modular Manikin (AMM) Software Design Document (SDD) CDRL Item A001 of Contract # W81XWH-14-C-0101, Phase II describes the AMM Core Software.

This CDRL is formatted to the requirements of Data Item Description Number DI-IPSC-81435A as required and tailored as recommended. It describes the design of the AMM Computer Software Configuration Items (CSCI), the architectural design and the software as implemented.

## System Overview

The AMM platform is a modular, distributed, interoperable system that enables physical, virtual, augmented and hybrid modules to work together as an integrated system. The traditional “core”, i.e. computer and state engine, can be in any one of the traditional manikin segments, i.e. torso, leg etc., or external to the human form, as it would be if the system is only running a virtual instance or if the targeted scenario, i.e. patient case, does not allow them to be internal due to the set of interventions that have to be performed on the body. The platform is architected as a system of systems that allow modules to function either as part of an integrated, whole body simulation or as autonomous part task trainers.

The published AMM standards guide the development and integration of AMM compatible modules. The reference designs provided for the final demo including electronics and central supplies were created to demonstrate the operation of the platform and are published as a developer’s tool kit with sources to acquire them from.

The developers of the platform have agreed to publish the AMM platform under the following open source licensing option:

\* *Creative Commons Attribution 4.0 International (CC BY 4.0)* [*https://creativecommons.org/licenses/by/4.0/deed.ast*](https://creativecommons.org/licenses/by/4.0/deed.ast)*.*

*Share — copy and redistribute the material in any medium or format*

*Adapt — remix, transform, and build upon the material for any purpose, even commercially.*

*The licensor cannot revoke these freedoms as long as you follow the license terms.*

This document does not cover modules that were created under separate funding and by other entities to demonstrate the functionality of the AMM Platform under separate funding and are not part of the Open Source agreement.

## Document Overview

This document is the AMM Software Design Description CDRL A001 SDD of Contract # W81XWH-14-C-0101, Phase II. The outline and subject matter content are based on DID DI-IPSC-81435A as required by the contract. The DID has been tailored to describe an open platform and open-source reference software that can be run on either the AMM reference computer hardware, or other user-selected computer systems. This document is unclassified and contains no proprietary information, trade secrets, copyrighted material or classified information. Unlimited distribution.

The purpose of the SDD is to describe the AMM reference software that, in conjunction with the Interface Design Description (IDD) CDRL A007, and Software Product Specification (SPS) CDRL A002, can be used by developers to create new AMM modules. The SDD does not include AMM modules that are demonstrated as part of the American College of Surgeons (ACS) study but are not part of the open-source AMM platform.

# Referenced Documents

## Industry Documents

|  |  |
| --- | --- |
| Doc. No. | Title |
| DDS | Data Distribution Service |
| DDSI-RTPS 2.3 | DDS interoperability Wire Protocol |
| IEEE 802.3 | IEEE Standard for Ethernet |
| IEEE 802.11 | IEEE Standard for Wireless Local Area Networks |

Note: Links to Industry Standards can be found in the Interface Control Document (ICD), CDRL A011.

## Government Documents

|  |  |
| --- | --- |
| Document Number | Title |
| W81XWH-14-C-0101 | AMM Phase II Contract, DOD |
| DI-IPSC-81435A | Data Item Description |

## Related Contract Documents

|  |  |
| --- | --- |
| Document Number | Title |
| CDRL A002 | Software Product Spec |
| CDRL A007 | Interface Design Description (IDD) |
| CDRL A008 | System/Subsystem Design Description |

# CSCI – Wide Design Decisions

In order to support communications among the core resources and any number of modules, a publish-subscribe messaging architecture was selected. Specifically, the Data Distribution Services (DDS) open standard was chosen based on the following criteria:

* The DDS specification is maintained by the Object Management Group, which grants a fully paid up, non-exclusive, non-transferable, perpetual, worldwide license to use this specification and to create and distribute software and special purpose specifications that are based upon this specification.
* The DDS specification is platform independent.
* There are multiple royalty-free and open source implementations of the specification:
  + [https://opendds.org](https://opendds.org/)
  + <https://www.rti.com/free-trial/open-source-projects>
  + <https://github.com/ADLINK-IST/opensplice>
  + <https://github.com/eProsima/Fast-RTPS>
    - RTPS is the DDS Interoperability communication standard
    - FastRTPS is used by our team in the AMM Reference Implementation
* DDS is a pure data-centric network architecture that scales to hundreds of publishers and subscribers in a robust manner. It enables the control of Quality of Service (QoS) that affect predictability, overhead, and resource allocation.

All modules, whether manikin, virtual, or auxiliary, publish and subscribe to shared data via the DDS bus. Except for data required to manage, configure, and test hardware modules, modules publish and subscribe to clinically relevant learner actions (including errors), patient events, and physiological variables in terms that are independent of the implementation. For example, a hemorrhage is represented as a loss of blood at a certain rate from a body segment or specific vessel, as opposed to a control voltage for a pump or valve. This makes it possible for the same data to be used to recognize actions and render events in multiple simulator modalities, whether whole-body manikins, part-task trainers, virtual patients, virtual reality, or standardized patient actor. In the case of a live role actor, for example, the actor could receive text or verbal input to represent particular behaviors.

# Architectural Design

## Data Architecture

Figure 2 shows the overall AMM data architecture. This architecture provides for:

* A Common Data Bus, based on the Data Distribution Services (DDS) standard, for communication between manikin modules, virtual patients, virtual and blended reality simulations. simulated medical equipment, physiology models, user interfaces, and performance assessment.
* Core software services, including Simulation Manager, Module Manager, and Physiology Manager.
* Standardized requirements for module configuration and communication.
* Physical mechanical, power, fluid, and data connectors between the torso and the head, arms, and legs.
* Power, network and fluid distribution and management among physical modules.
* Standard male and female patient anatomy and physiology and its digital representation.
* Standard representations of scenarios in support of authoring tools that should be developed.

Figure : AMM Processing Architecture Based on DDS Common Data Bus

In order to present a typical use case of the AMM Architecture, three types of components are shown with color coding:

* AMM Platform Core, Resource, and User Interface modules (in blue) provide required services. Open Source Reference Implementations are included for each of these modules. The Computer Software Configuration Items (CSCIs) for these modules are further described in Section 4.1 of this document. Note that, although a Reference Implementation is provided for each CSCI, it is not necessary to use this implementation to be AMM compatible. AMM adopters are free to develop modules or core components from scratch or to derive them from the Reference Implementations, under the Creative Commons 4.0 License, with only the requirement to provide appropriate attribution to their authorship. Compliance is assured through adherence to the data models.
* Additional Module Types have been demonstrated as part of the AMM project (in green) include:
  + Standard Manikin Modules for each of the six AMM segments. Head, Torso, Right/Left Arm, and Right/Left Leg Modules were provided by the University of Washington; with an abdominal skills plug-in provided by ACDET and an ultrasound simulator provided by CAE Healthcare.
  + Auxiliary Modules demonstrated as part of the project included commercial Virtual Patient and Virtual Patient Monitor apps provided by Vcom3D, Inc.
  + Auxiliary Simulation Modules that support interaction with the patient through means other than the physical modular manikin. As part of the AMM project, the following auxiliary simulation modules from Vcom3D were demonstrated: Virtual Infusion Pump, Virtual Ventilator, Virtual Urine Gauge, and Virtual Labs.

These non-platform modules are commercial products or are technology proprietary to the individual vendors and are not provided as part of the reference implementation.

* Future Module Types also fall into the categories of Core, Resource, User Interface, Auxiliary, and Auxiliary Simulation modules:
  + Future Auxiliary Modules that have been anticipated as part of the AMM Architecture include Scenario Generation and Learning Management System (LMS) or Learning Reference Story (LRS) Interface:
    - Currently, AMM Scenarios are created by configuring modules and loading the desired initial Medical Treatment Environment, Ambient Environment, and Patient Physiological State. A future Scenario Generation capability would author a Scenario File that includes this data.
    - An Assessment data type has been defined to support recording specified events and potentially creating corresponding Experiential Application Programmer’s Interface (xAPI) statements.
  + Future Auxiliary Simulation Modules that have been anticipated include:
    - Virtual Reality (VR) or Augmented Reality (AR).
    - Virtual Medical Devices, such as an instrumented tourniquet, blood pressure cuff, or syringe.
    - Environment Modules that might simulate the changing ambient temperatures, gas pressures, or humidity of a Point of Injury or Medical Treatment Facility.
    - Part Task Trainers (PTTs) that might simulate a specific portion of the human anatomy.
    - A “Manikin as a Module” that uses the AMM data architecture but does not implement segmentation of the manikin into segments.

Although these Future Model Types have not been demonstrated, the AMM Architecture is designed to support them in the same way that it supports standard, segmented manikins.

## Data Distribution System (DDS) Common Data Bus

DDS is networking middleware that simplifies complex network programming. It implements a publish–subscribe pattern for sending and receiving data, events, and commands among the nodes. Nodes that produce information (publishers) create "topics" (e.g., temperature, location, pressure) and publish "samples". DDS delivers the samples to subscribers that declare an interest in that topic.

The key benefit is that applications that use DDS for their communications are decoupled. Little design time needs to be spent on handling their mutual interactions. In particular, the applications never need information about the other participating applications, including their existence or locations. DDS transparently handles message delivery without requiring intervention from the user applications.

DDS allows the user to specify quality of service (QoS) parameters to configure discovery and behavior mechanisms up-front. By exchanging messages anonymously, DDS simplifies distributed applications and encourages modular, well-structured programs. DDS also automatically handles hot-swapping redundant publishers if the primary fails.

### **DDS Selection**

To provide a standard communications protocol between connected AMM modules, the Real-Time Publish Subscribe (RTPS) Data Distribution Service (DDS) Interoperability Wire Protocol, as defined by the Object Management Group (OMG), was selected. The DDS standard defines an application programming interface (API) to provide a publish-subscribe pattern for data distribution between nodes in a network. RTPS defines a method for DDS implementations to interoperate by providing a data encapsulation standard. RTPS uses the User Datagram Protocol (UDP) for data transport, enabling DDS communication to use widely available Internet Protocol suite (IP) networking equipment and technology.

DDS was selected as the communications middleware after reviewing the communications needs of the AMM and evaluating several alternatives. Because of the variety and volume of data expected to be transmitted in such a sophisticated system as AMM, the need for a publish-subscribe data-exchange pattern was recognized even in Phase 1. Additionally, because module connectivity may be unreliable, the need for Quality of Service (QoS) controls for data delivery was identified. Finally, to facilitate ease of connecting modules external to the physical manikin, modules need to be able to connect using standard IP networking technology.

Based on the communications needs of the project, Entropic Engineering evaluated ZeroMQ, MQTT, and DDS. ZeroMQ is the simplest, light-weight data transport option, with very low overhead, but no QoS controls. MQTT is the most widely adopted of the options we evaluated, has only slightly more overhead than ZeroMQ, and does provide very simple QoS controls. DDS is far more sophisticated and complicated than either ZeroMQ or MQTT. However, it handles many of the complexities of networked data distribution that would otherwise need to be handled by the modules or by a centralized broker. DDS has very robust QoS controls, handles node discovery via RTPS, is completely distributed and requires no central broker, and provides a standardized means of data encapsulation. Furthermore, DDS completely abstracts the actual data exchange, meaning module software doesn’t need to send or receive individual messages, but rather simply publish and subscribe to changes in data.

## User Interfaces

The user interacts with the system via the AMM User Interface or through a Command Line Interface (CLI). Overall control of the system is shared by two modules: Module Manager for startup, shutdown and exceptions; and the Simulation Manager to run the AMM Scenario. This architecture is easily expandable to include other modules.

## Computer Software Configuration Items (CSCIs)

The AMM project is comprised of the following CSCI components:

1. **AMM Data Model definition CSCI**

All AMM-connected modules share a common Data Model definition, expressed in a standard Interface Definition Language (IDL) as defined in <https://www.omg.org/spec/IDL/3.5/> and by the Common Object Request Broker Architecture (CORBA) Specification.

1. **AMM Standard Library software CSCI**

A reference implementation of the AMM Data Model definition developed in C++, exposing an application programming interface (API) for developing modules and other AMM-connected software.

1. **AMM Core module software CSCI**Reference implementations of the software required to run an AMM-compatible manikin, including:

* Module Manager
* Simulation Manager
* Physiology Manager

Extended core modules (not required for simulation operation) included with the Core software package include:

* REST adapter
* TCP bridge

1. **Reference** **modules:**

* Virtual Equipment simulator
* Command executor
* “Kitchen Sink” example
* Fluidics Manager

1. **AMMDK Microcontroller Development Environment CSCI**

Programming and debugging embedded microcontroller systems is usually a complex, daunting and often expensive task requiring the purchase of expensive hardware and software tools. To address this issue, a fully integrated system has been developed that allows users of the AMM Developers Kit to program and debug the system’s microcontrollers without the need for additional software or tools. The AMMDK Microcontroller Development Environment enables its users to build any AMM Module they desire.

1. **AMM Network Controller CSCI**

Provides IP (internet protocol) networking connectivity, including routing, DHCP addressing, local DNS addressing, and time service, for all physically connected modules as well as Wi-Fi access for wirelessly connected modules.

1. **AMM User Interfaces CSCI**

A dashboard interface is provided as a simple means to perform standard AMM tasks such as loading a scenario, loading a patient physiology state and starting/pausing/resetting a simulation.

## Hardware Configuration Items (HWCIs)

While the AMM Reference Implementation software has been developed to be buildable on both Windows and Linux systems, performance and resource requirements will vary greatly depending on the module that is being developed. A full AMM CORE Reference Implementation–including a Module Manager, Simulation Manager, Physiology Engine Manager with BioGears, REST Adapter and TCP Bridge–will require at least 1GB of RAM, and a multi-core processor is highly suggested. A simple module that drives hardware, for example the fluidics manager, can exist on far fewer resources and a single processor.

The CORE reference implementation has been tested on a Qualcomm Dragonboard 410c and was able to run a simulation with real-time performance when no modules were connected. However, once just a few modules were subscribing to messages, performance degraded below real-time. The CORE reference implementation has specifically been tested with the AMM Development Kit (AMMDK), described below, and can run faster than real-time with multiple modules connected.

### **AMM Development Kit (AMMDK)**

The AMM Development Kit consists of hardware, the AMMDK Common Compute Board (AMMDK-CCB) and software, including a modified Linux distribution and libraries designed to assist in development. AMMDK software is discussed Section 5.6.

The AMMDK Common Compute Board (AMMDK-CCB) features an Arm64 System on Module (Snapdragon 820E) running embedded Linux connected via SPI to an Arm Cortex M4 microcontroller (K66F) for external hardware peripherals. Additionally, a second, lower-powered Cortex M4 MCU (K20) is connected to both the K66F as well as the Snapdragon to enable real-time execution control of the K66F, which greatly facilitates in development and debugging. This architecture is shown in Figure 3.

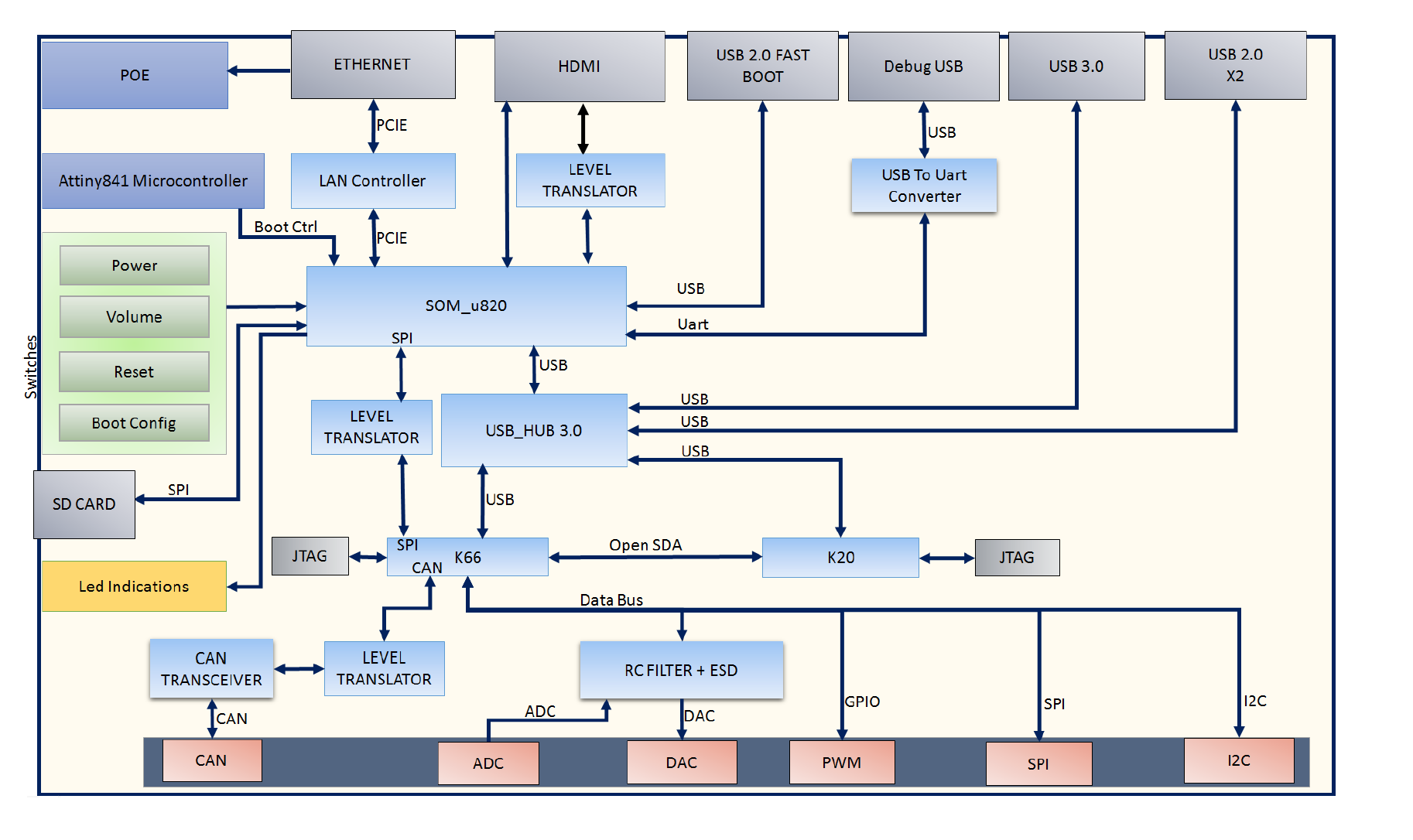


Figure : AMMDK-CCB Architecture

The AMMDK-CCB is built around the “Qualcomm® Snapdragon™ 820E high performance embedded platform”, which is designed to provide powerful, energy-efficient, multi-core processing for the next generation of embedded computing applications, with manufacturer support until at least 2025. The Snapdragon 820E SoM provides sufficient computational power to run the entire AMM core software stack with plenty of performance to spare for developing computationally intensive modules. This facilitates the development process by running everything on a single system. The Snapdragon 820E also provides support for Wi-Fi and Bluetooth Low Energy for wireless communications.

To interface with hardware peripherals and enable real-time control, the AMMDK-CCB includes the Kinetis® K66 microcontroller built on the Arm® Cortex®-M4F core. It features connectivity to nearly any hardware peripheral with support for common electrical protocols such as I²C, SPI, CAN, UART, USB, and Ethernet, along with standard GPIO. The K66F MCU communicates with Snapdragon SoM via SPI (at 50 MHz).

The K66F was selected because of its many features as well as broader community support. The chip is used in several other open-source hardware projects and, thus, there is a lot of open-source software available for it. Most notably, the Teensy 3.6 development board uses the same chip and features many compatible open source software libraries provided by its creator, Paul Stoffregen.

In order to ease the complex burden of embedded software development, the AMMDK-CCB also features a Kinetis® K20 microcontroller configured solely to provide real-time debugging of the K66F MCU. The K20 MCU is running OpenOCD and requires no direct programming from users. The K20 MCU is connected via USB to the Snapdragon 820E, enabling the widely used GDB debugger to take direct control of the K66F.

The AMMDK-CCB also provides standard HDMI and USB-A ports in order to allow AMM module developers to use the AMMDK-CCB as a stand-alone workstation for AMM development. No external hardware programmers are required to interface with the included MCUs, and all of the required software toolchains needed to program the K66F are pre-installed and configured. This greatly reduces the time required for new developers of AMM.

Further information about the AMMDK, including software, user guides, and purchasing information is available in the AMMDK GitHub repository: <https://github.com/AdvancedModularManikin/development-kit>

### **AMMDK Application Board**

The AMMDK-CCB is designed to provide a standard computational and communications platform for module development. However, connecting to hardware peripherals necessitates additional electronics to ensure electrical compatibility between the K66F MCU and the peripheral. One of the design goals of the AMMDK-CCB is to help simplify the creation of this additional component. Thus, the Application Board needs only to be a simple board with just the necessary electronics for the module (e.g. drivers/signal filters/DC/DC converters to generate any specific voltage rails) that will plug into the AMMDK-CCB via an interconnect.

## 

### **AMM Network Controller (AMMNC)**

The AMMNC is designed to provide network connectivity and power to an AMM system. The AMMNC is a modification of the design of the AMMDK-CCB to enable the AMMNC to function as Power over Ethernet (PoE) Power Sourcing Equipment (PSE). See Figure 4.

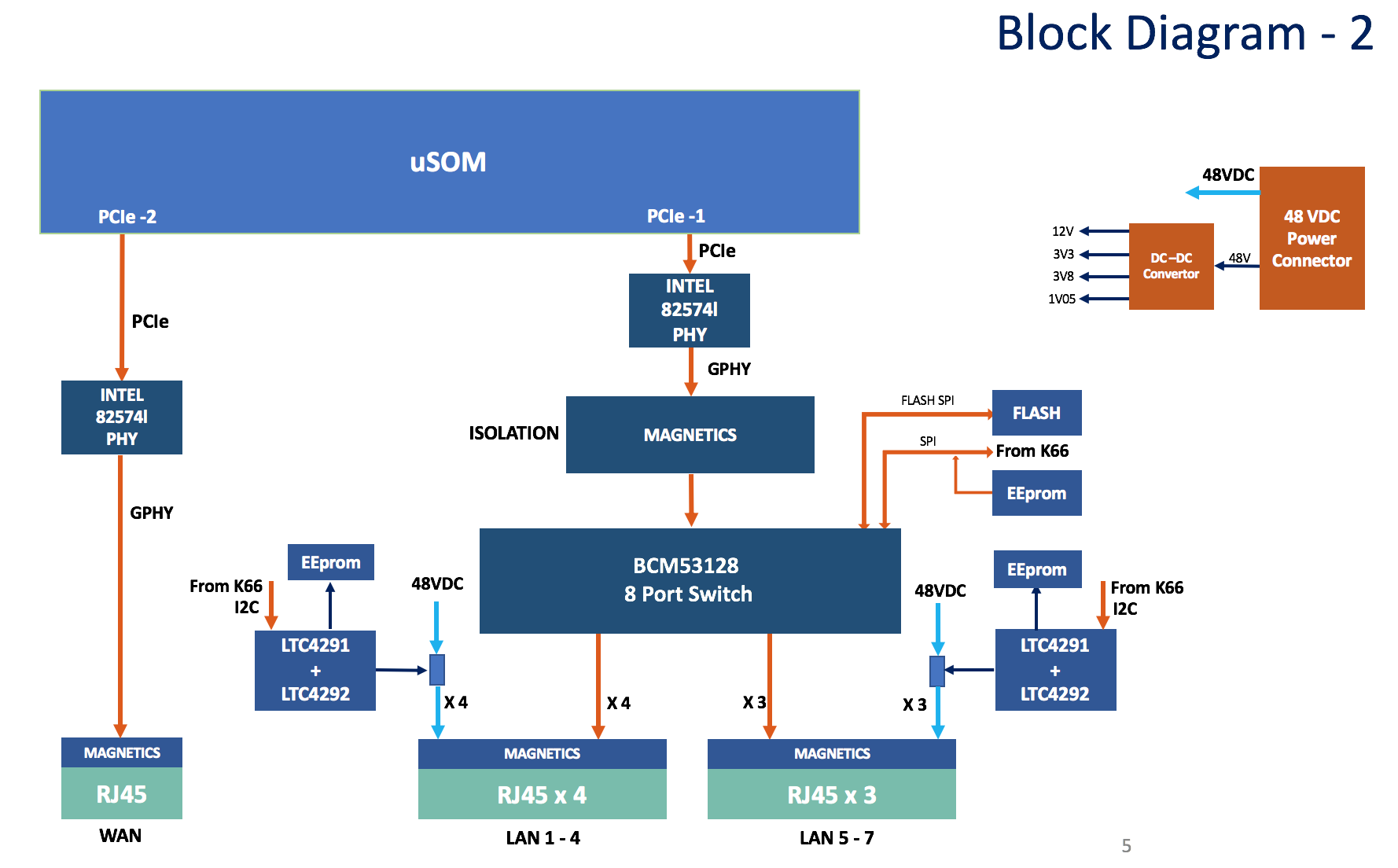


Figure : AMMNC Enhancements to AMMDK-CCB

The AMMNC provides 8 Ethernet ports: 1 WAN/uplink port, and 7 LAN ports with PoE power. The WAN port provides wired connectivity for an AMM manikin to an external network. The LAN ports provide power and data for the 5 ‘extremity’ AMM Segment Modules (head & limbs), with 2 additional ports available for manikin developers. The switching chip for the LAN ports is connected to the K66F MCU for configuration and control. Additionally, the PoE chips (LTC4291/2) are also controllable by the K66F MCU, enabling the AMMNC to control power to all connected AMM Modules.

## Interface Design

Software interfaces are described in the Interface Design Document (IDD), CDRL A007.

# Detailed Design

## AMM Data Model Design

During manikin operation, data transmitted between modules fall into one of three broad categories: The first is the state of the scenario being simulated, which, for AMM, broadly consists of the state of the patient and their environment. The second category is data that are generated as a result of some event, frequently caused by a user intervention, but sometimes triggered by the scenario. The third category of data is information about the state of the simulator, including control of whether the simulation is running.

## Simulation Data

For the first category, the long-term goal is to develop an abstract, extensible, engine-agnostic data model for patient simulations, including patient physiology and anatomy data, as well as environmental data. However, AMM version 1 relies on the BioGears Common Data Model (CDM) for simulation data. Data element names are the same as those in the BioGears CDM.

BioGears data can be accessed by a module using two different modes: low frequency Physiology Values and high frequency Physiology Waveforms. Physiology Values are sent on a best-effort basis, and are not necessarily sent for every BioGears frame. Physiology Waveforms are delivered reliably and are sent for every BioGears frame (20ms). Both Physiology Values and Waveforms have the same format (their only difference is in their respective Quality of Service (QoS) settings for DDS-RTPS):

* Simulation Time (Value of the simulated clock)
* Timestamp (UTC epoch, millisecond resolution)
* Name (Name of data element, taken from BioGears CDM, DDS Topic Key)
* Value (Numerical value of data)
* Unit (The units of the data)

**Environmental Data**

For AMM version 1, all environmental data is defined, accessed, and controlled through BioGears, using the same pathways as physiology data.

**Simulation Time**

Simulation Time is the clock time in the simulated world. Because it is tightly coupled to the simulated physiology, the simulated clock must be managed by the physiology engine. When a scenario is loaded, a starting time shall be part of the physiology engine configuration.

### **Event Data**

Events, broadly speaking, occur when “something happens” during the simulation. The most obvious example of Events are interventions taken by a practitioner in treating the patient.

When a module detects that an event has occurred, it publishes an Event Record. Based on the data included in the Event Record, one or more modules may publish additional data generated by the event, such as a change to physiology or a performance assessment. Each Event Record has a unique ID that is referenced by all other data generated from the event.

For an Advanced Modular Manikin to respond to an event, two things must happen: a module must publish an Event Record, and one or more modules must publish data generated from the event. This generated data comes in three categories: a change to Physiology (e.g. a hemorrhage), an Assessment of how the event was performed, and a change to what information is Rendered by modules. Data from any of these categories may be generated from an Event Record. These generated data may be published by modules other than the publisher of the Event Record, if appropriate.

In some cases, an Event Record may contain information provided by multiple modules, such as when an instrument can sense when it’s used, but cannot sense where. Compound Data Events are defined below.

### **Event Records**

These data are published primarily in order to review “what happened” over the course of a simulation. They do not directly influence the behavior of an AMM during operation, but serve as a reference for the data that does cause direct influence. Every field is required, though Location and Agent values may be set to Unknown or Not Applicable.

* ID (UUID)
* Timestamp (UTC epoch, millisecond resolution) (‘when’)
* Location (FMAID [Foundational Model of Anatomy ID number]) (‘where’)
* Agent ID (Who caused the event, together with Agent Type) (‘who’)
* Agent Type (One of / Enum)
* Learner
* Instructor (Triggered by Instructor, perhaps on behalf of Learner)
* Scenario (Triggered by scenario definition, including initial patient state)
* Physiology (Triggered by patient physiology reaching a given state)
* Type (Use existing vocabulary if possible) (‘what’)
* Data: XML (Format tied to Type) (‘how’)

The format of the Data field of an Event Record is dependent on the Type field. This mapping is maintained in the Event Record Types Glossary.

**Physiology Modifications**

This data primarily corresponds to changes to patient state caused by external actions rather than just the ongoing development of the patient over time. Common examples include drug administration, ventilation, and causing or stopping a hemorrhage. Because development of an engine-agnostic data model is outside the scope of AMM Version 1, the BioGears Actions are used directly for this data category.

When an Event Record has a Location associated with a segment of the body, and there is a module simulating that Location, only said module is allowed to publish Physiology Modifications in response to that Event Record. This is because there may be local state in the module that is unknown to the rest of the manikin that may impact the physiological reaction to a given event.

The fields for the Physiology Modification data are:

* ID (UUID)
* Event ID (Identifier of the Event Record that caused this Physiology Modification)
* Type (Use full names & capitalization [including spaces] from BioGears CDM, listed under ‘Action’ headings here)
* Data: XML (Provide exact XML required for BioGears to process Action)

**Render Modifications**

This category encompasses changes to any of the information being actively rendered by modules in a simulation. This includes all physical findings, placements of medical devices, internal injuries, and even the presence or absence of data on a patient monitor due to sensor placement. Because this data category is so broad and encompasses many disparate types of information, it may be broken down into more granularity in future versions of the specification.

The fields for the Render Modification data are:

* ID (UUID)
* Event ID (Identifier of the Event Record that caused this Render Modification)
* Type (Per Render Modification Glossary)
* Data: XML (Per Render Modification Glossary)

**Assessments**

This data is generated by modules in order to evaluate learner performance of specific activities. The fields for this data type are:

* ID (UUID)
* Event ID (Identifier of the Event Record that generated this assessment)
* Assessment Value (One of / Enum)
  + Omission Error (Skipped step)
  + Commission Error (Extra/out-of-order step)
  + Execution Error (Done poorly)
  + Success
* Comment

**Omitted Events**

Sometimes, in the course of a procedure, actions that were supposed to have been taken are missed. For proper assessment, these omissions must be captured. Because Performance Assessment records are tied to a specific Event Record, and because Omitted Events are things that did not happen and, therefore, should not cause changes in physiology, Omitted Events are published on a distinct DDS Topic from Event Records, but otherwise share the same format as Event Records.

The Timestamp indicates when the omission was detected, not when the omitted event should have occurred. However, the Location, Agent ID/Type, Type, and Data fields should indicate the who, what, where, or how for the omitted event. Only the ID, Timestamp, and Type fields are required, but any additional data should be provided if known.

**Compound Events**

In some cases, a module may not have all of the information required to publish an Event Record. For example, a smart syringe should publish an Injection Event, but has no way of knowing where the injection was performed. See Figure 5.

To account for these cases, modules may publish Event Fragments. Event Fragments are published on a separate topic from Event Records but contain the same data fields as an Event Record. The difference is that certain fields may be published with a null value.

Other modules may then subscribe to these Event Fragments and publish a Fragment Amendment Request when they have data that is applicable to a particular Event Fragment. These Fragment Amendment Requests are published on their own topic, so modules can subscribe & unsubscribe as necessary. Each Fragment Amendment Request includes a status value. Modules that ‘make’ a Request publish it with a status of Requesting. The module that published the Event Fragment will then respond to the Fragment Amendment Requests by publishing a new version of the Request with the Status field updated to either Accepted or Rejected.

Once a module has ‘accepted’ a Fragment Amendment Request (by publishing an updated version of the Request with the Accepted status) for the missing data element(s) of the initial Event Fragment, the module will then publish a complete Event Record. Finally, the module will publish updates to any outstanding Fragment Amendment Request with a Status of Rejected.

In the case of multiple missing data fields in an Event Fragment, the initiating module may post additional Fragments with incremental updates based on Accepted FAR data.

The data types involved in this interaction are:

**Event Fragment**

Data types are the same as in the Event Record, but some fields may be null.

* ID (UUID) – Required (Unrelated to future Event Record ID)
* Timestamp – Required
* Location – Optional
* Agent ID – Optional
* Agent Type – Optional
* Type – Required
* Data – Required

**Fragment Amendment Request**

The module ‘making’ the request first publishes the Fragment Amendment Request with a Status of Requesting. The module must also subscribe to changes in this Request as the Request will be republished with an updated Status value by the other module.

* ID (UUID)
* Fragment ID (Fragment being amended by this Request)
* Status (Exactly one of / Enum)
  + REQUESTING (Initial status)
  + ACCEPTED
  + REJECTED
* Timestamp – Optional
* Location – Optional
* Agent ID – Optional
* Agent Type – Optional
* Type – Optional
* Data – Optional

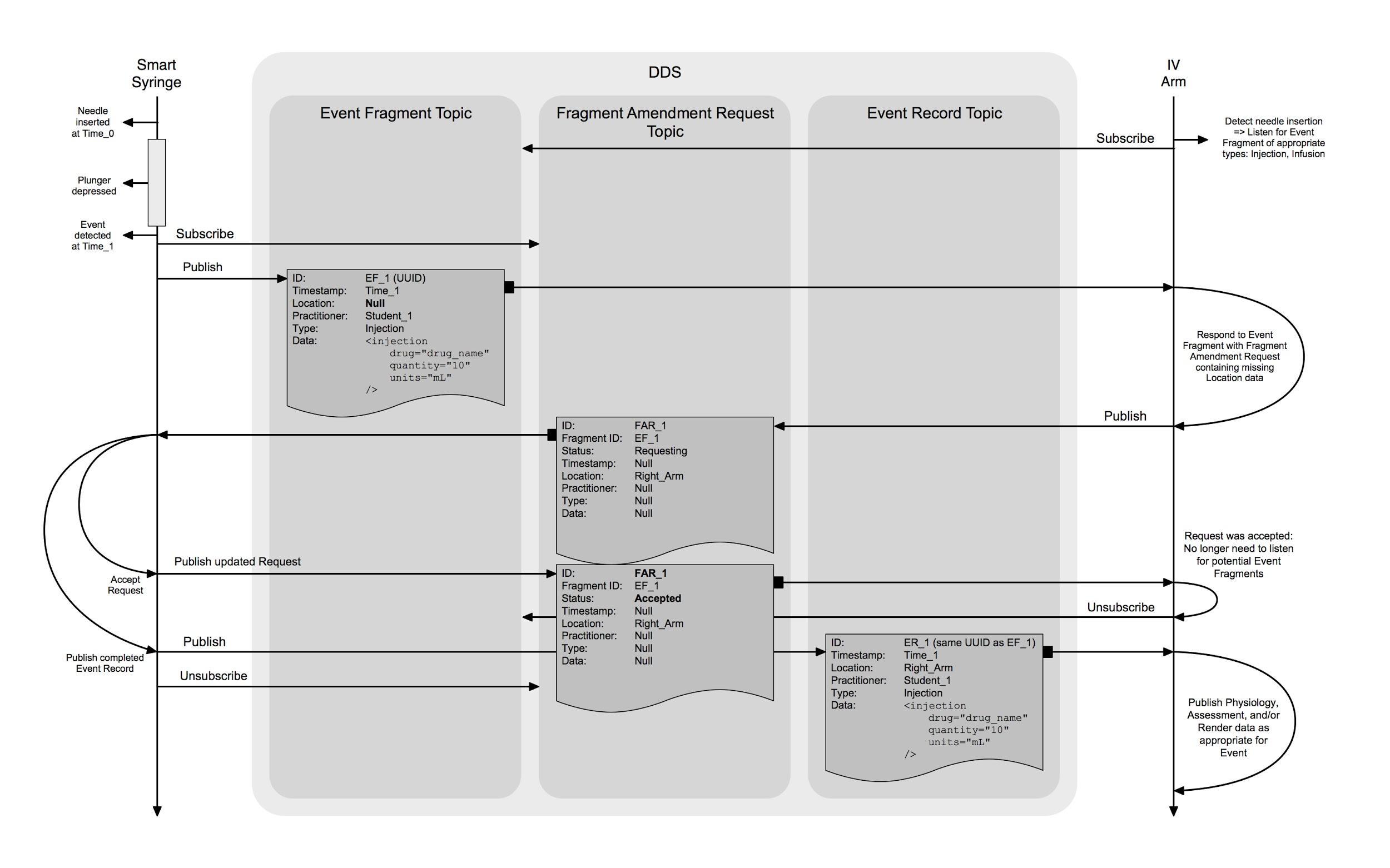


Figure : Compound event data flow

### **Simulator Data**

This third category of data encapsulates the state of the hardware and software that are used by the practitioner in order to interact with the state of the simulation. This includes control of when the simulation is running, the Statuses published by every Capability of every Module, and all Logging.

**Simulation Control**

These messages are used to control the simulation, itself. All modules must subscribe to this Topic and behave appropriately in order for the simulation to function correctly.

* Timestamp (UTC epoch, millisecond resolution)
* Type (Exactly one of these / Enum)
  + RUN (run simulation)
  + HALT (stop advancing simulation, preserve module state)
  + RESET (end simulation, reset module to default state, implies Halt)
  + SAVE (publish module configuration and saved state)

Note: For the SAVE Simulation Control, modules must publish data to their configuration topic which will allow the module to recover the current state sometime later. The configuration that the module publishes will be saved and may be reloaded as configuration in the future. When feasible, modules should serialize state rather than saving to local storage.

**Capability Status**

Modules report their ability to participate in a simulation by updating the Status for each of the Capabilities provided by the module (as selected by the Scenario). Module readiness is broken-up this way because some module functionality may require resources that other functionality doesn’t, and the Scenario may only specify that only certain functionality is required for a simulation.

* Module ID (UUID, DDS Topic Key)
* Module Name (for UI display purposes)
* Educational Encounter (UUID, assigned by Module Manager when config is loaded)
* Capability Type (XML tag, including attributes, per Glossary; DDS Topic Key)
* Timestamp (UTC epoch, millisecond resolution)
* Status Value (Enum)
  + OPERATIONAL (module is able to run/is running a simulation)
  + INOPERATIVE (module is currently unable to participate in a simulation)
  + EXIGENT (module can currently function, but requires operator intervention to avoid becoming Inoperative during simulation)
* Status Message (for UI display purposes)

Note: The Exigent status value should be used primarily to raise alerts to the AMM operator that action is required in order to continue simulation. The most obvious use-case is to alert users when resources (namely power and/or fluids) are in danger of being exhausted and causing unexpected termination of the simulation.

**Diagnostic Logging**

Modules may publish logging messages during operation which may be collected and presented to end users, or for review. If a module wishes to produce logs, they should conform to the following format and behavioral descriptions:

* Timestamp
* Affected Module ID (UUID, usually but not necessarily the same as module publishing the message)
* Level (Enum)
  + FATAL, ERROR, WARN, INFO, DEBUG, TRACE
* Message

For AMM, these log levels have specific meanings and behavior expectations attached:

**FATAL**

Failure requiring imminent shutdown

Shall be published alongside Inoperative Status(es)

**ERROR**

State where module cannot function, but still potentially recoverable

Shall be published alongside Inoperative Status(es)

**WARN**

Undesired state, but module can still function

If time critical, published alongside Exigent Status(es)

**INFO**

Not tied to Status(es)

Limit publishing frequency to approximately 1Hz

**DEBUG**

Shall be disabled during ‘normal’ operation

Do not publish in commercial modules unless explicitly enabled

**TRACE**

Do not publish in commercial modules

Included only for developer convenience

Note: Log messages should not be used to generate alerts or notifications for the main manikin UI. Capability Status(es) are the correct source for these UI alerts, which should be generated by the Core Software (Module Manager in the reference implementation). However, additional interface (e.g. technician’s view) may wish to leverage logging data directly.

### **Configuration Data Model**

In order to simulate a scenario, modules capable of performing the desired simulation must be present and properly configured. In order for a given scenario definition to be simulated on a variety of hardware, common definitions of module functionality, requirements and configuration must be established. This section outlines the format for these definitions.

**Capabilities**

Capabilities are the logical Unit of Functionality for a given module. Each distinct function of a module should be broken into a separate capability where possible. This not only clearly defines what all a module is capable of, but also enables functionality to be configured and monitor independently, providing better resilience for modules designed and built this way.

Module technical definition and configuration is broken down by Capability, and modules report their status on a per-Capability basis.

When possible, manufacturers shall use existing Capability descriptions to denote functionality and configuration within their modules. This greatly enhances interoperability and enables re-use of scenario definitions.

Definitions for capabilities are in the Capability Types Glossary.

**AMM Standard Version**

Based on Semantic Versioning: Major.Minor.Patch

Major Version: Changes to IDL / XML Schemas, breaking changes to Glossary definitions

Minor Version: New entries added to Glossaries

Patch Version: Non-breaking changes to Glossary definitions, bug fixes

**Operational Description**

Published by the Module

* Name (human friendly)
* Description (1 paragraph)
* Manufacturer
* Model
* Serial Number
* Module ID (UUID, DDS Topic Key)
* Module Version
* AMM Standard Version
* Capabilities Schema ([XML Schema Definition](https://github.com/AdvancedModularManikin/specification/blob/master/schema/CapabilitiesSchema.xsd), more details below)

**Capabilities Schema:**

Always <?xml version="1.0" encoding="UTF-8"?> with a root element of <CapabilitiesSchema>.

If powered via PoE <BaselinePoEPower> must be the first child element. BaselinePoEPower tag always has only two attributes, nominal and units. Together, their value defines how much power the module requires, before any additional capability power requirements. This is usually the power draw of the embedded CPU.

Every module must define at least one Capability. Capability tag types and additional attributes shall be defined per the [Capabilities Glossary](https://docs.google.com/document/d/1jFEMcXX2VFgVjZsFSklCBgwxhAGO5AcqI13vpRp32Ew/edit). Capability tags have exactly four (ordered) child elements:

<Subscriptions></Subscriptions>

<Publications></Publications>

<Assessments></Assessments>

<Resources></Resources>

The Subscriptions and Publications tags contain only child elements defining what topics a module subscribes and publishes to. Child element tag names are identical to the topic names defined by [AMM.idl](https://github.com/AdvancedModularManikin/specification/blob/master/schema/AMM.idl). Element attributes are defined in the [XML Schema Definition](https://github.com/AdvancedModularManikin/specification/blob/master/schema/CapabilitiesSchema.xsd#L139). All child elements of the Subscriptions and Publications tags take an optional match attribute, which indicates whether a matched subscriber or publisher is optional (default) or required for proper module operation. A required match indicates that the module capability will not achieve OPERATIONAL state without the appropriately matching publisher or subscriber.

The Assessments tag contains only child elements defining Event Records (including Omitted Events) that the module may assess. While modules are not required to assess any Events, assessment is a core component of high-quality education, and, thus, a core feature of the AMM standard. As such, module manufacturers should strive to include assessments as a core component of their modules.

The Resources tag contains only two types of elements Requirement and Supply, though each may be repeated to indicate which resources are being supplied or consumed. The element definitions are:

<Requirement

type=”Power, Blood Simulant, Clear Liquid, or Compressed Air”

nominal=”Rate of fluid draw or power usage during normal operation”

peak=”Peak fluid draw or power usage, optional attribute”

unit=”Volume per time unit”

/>

<Supply

type=”Power, Blood Simulant, Clear Liquid, or Compressed Air”

capacity=”Fluid supply capacity”

unit=”Volume per time unit”

/>

Finally, any & all configuration for a module is contained in a final (optional) <Configuration> tag. The Configuration tag should either contain binary data (wrapped in <hex> or <base64> tag), or contain an XML Schema definition for the available configuration values, if feasible. This is to enable future software to prompt end users for configuration values. An example is [available here](https://github.com/AdvancedModularManikin/specification/blob/master/ExampleCapabilitiesSchema.xml).

**Module Configuration**

First published by Module at boot, then updated by Module Manager with configuration when Scenario is loaded. May also be updated later by Module in response to a SAVE control.

* Name
* Module ID (Module-generated UUID, DDS Topic Key)
* Educational Encounter (UUID, assigned by Module Manager when config is loaded)
* Timestamp (UTC Epoch, millisecond resolution) – Time of last update
* IP Address
* Capabilities Configuration (XML)
  + Always <?xml version="1.0" encoding="UTF-8"?>
  + Root element: <Configuration>, derived from Capabilities Schema

**Saving & Loading State**

To load a saved state, the Module Manager publishes the exact same Module Configuration data that was recorded in response to the SAVE control, but with a new Educational Encounter value. Thus, Modules must serialize their internal state in order to publish that state in response to a SAVE control. This also means the Capabilities Schema for the module should reflect possible state serializations.

### **Module Behavioral Requirements**

To ensure module interoperability, Modules must adhere to a set of Behavioral Requirements along with the Configuration and Operational Data Models. The Behavioral Requirements ensure a collection of connected AMM Modules will perform as expected, even with individual Modules produced by many different manufacturers. This section defines those Behavior Requirements, breaking them down into three overall domains: Startup, Configuration, Operation

**Startup**

AMM uses the DDS-RTPS (version 2.2) protocol for all data transport between modules.

**Networking**

As an AMM Module boots, it needs to establish a network connection in order to communicate with any other Modules. Because DDS-RTPS is used as the underlying transport for data within AMM, Modules need to connect via UDP over IP(v4) and accept UDP broadcasts.

**Time Synchronization**

Because a simulation is run over a distributed network, syncing time between modules is important. AMM relies on the widely used NTP standard for this.

**DDS**

Once an AMM Module has finished booting, has established a network connection, and synchronized its time, it needs to establish a DDS-RTPS connection to other Modules. AMM uses the default DDS domainId of 0. All DDS QoS values are default (per DDS version 1.4), except as defined in comments for each Topic in the AMM.idl specification file.

**Configuration**

Once an AMM Module has established a DDS-RTPS connection to the other Modules on the network, the Module shall publish to the OperationalDescription and ModuleConfiguration Topics.

**Operational Description**

A Module’s Operational Description is static for the Module. It provides the information required to uniquely identify the Module, as well as providing a description of Module functionality.

**Module Configuration**

The ModuleConfiguration topic provides the current configuration for a Module, broken down by Capability. When a scenario is loaded by the core software stack, the Module Manager (or equivalent) publishes messages to the ModuleConfiguration topic with the data required for the scenario for each Module. When a save Simulation Control message is published, Modules must also publish an updated ModuleConfiguration with the appropriate data. Thus, every Module must publish and subscribe to this Topic.

**Status**

Modules shall publish a Status for each Capability that is enabled. The timestamp field is updated each time a new Status is published. The module\_id field shall match the value of the Module Configuration module\_id field. The educational\_encounter field shall match that assigned when configuration was assigned for the current scenario.

**Simulation Control**

All Modules shall subscribe to the SimulationControl topic and respond appropriately. Modules shall only publish to the SimulationControl topic as a result of direct user input.

## AMM Standard Library Design

The AMM Standard Library has been developed as a reference implementation of the AMM Data Model definition, exposing APIs to allow developers to quickly develop AMM-compliant modules without having to work directly with the data bus itself. The Standard Library has been developed utilizing eProsima FastRTPS for DDS middleware.

The standard library offers API interfaces to:

* Connect to an AMM network utilizing the ‘standards-compliant’ handshake, including publishing capabilities, receiving configuration and publishing operational status.
* Subscribe to AMM Data Model topics
* Publish to AMM Data Model topics
* Publish diagnostic logging information
* Resolve Event Fragments and Event Fragment Amendment Requests

## AMM CORE Module Design – Required Modules

1. **Module Manager**

The Module Manager is a core software component that coordinates the participation, initialization, configuration, and termination of AMM modules during the educational encounter. All AMM compliant modules must perform an appropriate handshake procedure which includes information about the module (operational description), the capabilities it provides, and the configuration data needed to provide those capabilities. The Module Manager is also responsible for loading scenarios and publishing the configuration data specific to each module to enable the capabilities required by the scenario. The Modules must validate their configuration and report the operational status of each of their enabled capabilities. The Module Manager will aggregate the operational statuses of all the modules to determine if all the required capabilities of a scenario are available and operational. Figure 6 details this process.

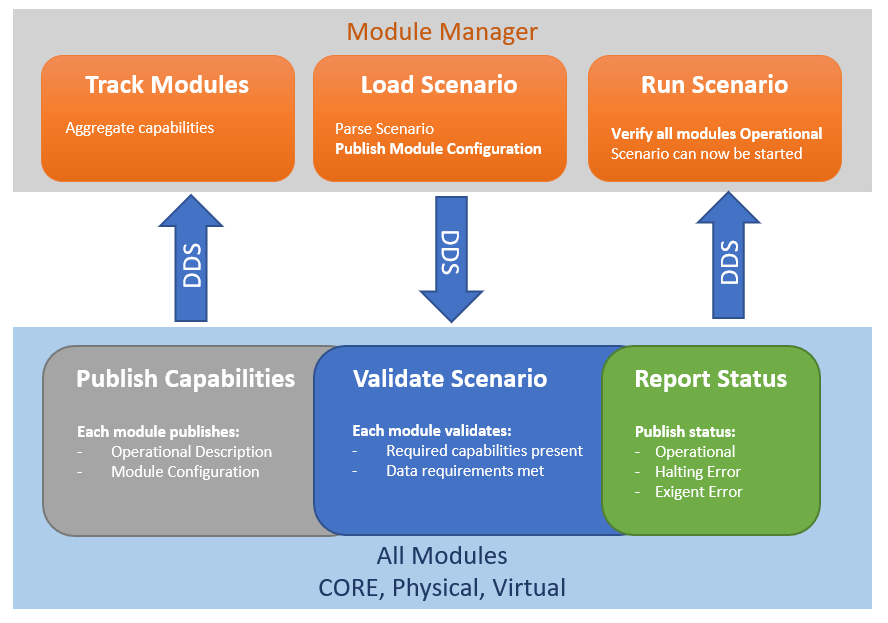


Figure : Module Manager Functions

1. **Simulation Manager**  
   The Simulation Manager is a core software module that drives the simulation by publishing simulation ticks at a frequency of 50 Hz. It operates as a simulation state engine, allowing for control of time-related functions and loading scenarios for execution.
2. **Physiology Engine Manager**  
   The Physiology Engine Manager is a core software module that connects to a physiology engine. The Physiology Engine used for the Reference Implementation is the Open Source BioGears 7.2.0, developed on Government Contract W81XWH-13-2-0068, entitled Developer Tools for Medical Education Public Physiology Platform, and available from <https://biogearsengine.com/>. A BioGears wrapper module is provided as an exemplar and guide for how other physiology engines might be connected to AMM.

## AMM CORE Module Design – Extended Core Modules

1. **REST Adapter**  
   The REST Adapter is a web service that converts HTTP Representational State Transfer (REST) requests to DDS messages in support of web browser-based modules like the Dashboard. A web application can be built on top of AMM utilizing the REST services, as shown in Figure 7.

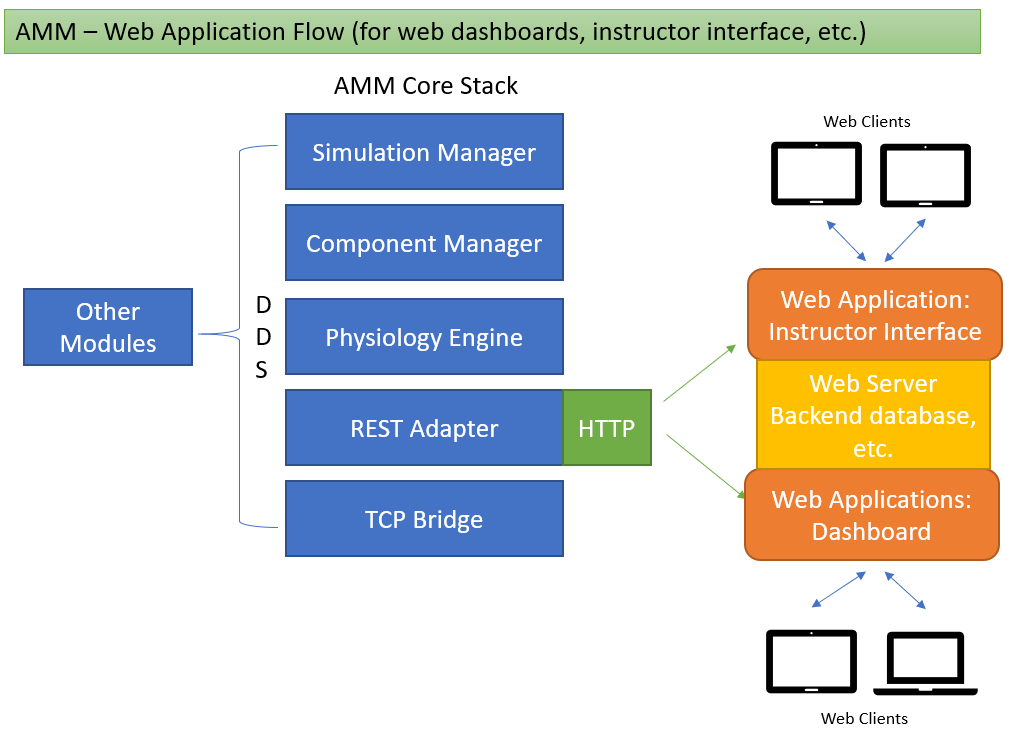


Figure : AMM Web Application Data Flow

1. **TCP Bridge**  
   The TCP Bridge is a socket server that handles TCP communications with socket clients and converts them to DDS messages in support of modules which cannot natively implement DDS but can open network sockets such as the virtual patient and virtual equipment. The TCP bridge has been used to connect Unity games as well as hardware modules that could not easily support a full DDS stack.

## AMM Reference Modules

The following reference modules are included as a guide for developers to develop their own AMM modules. Examples are given of how to subscribe and publish data, transmit capabilities to the module manager and set operational status.

1. **Virtual Equipment**  
   The Virtual Equipment simulator subscribes to physiology messages and prints them to standard output.
2. **Command Executor**  
   The Command executor takes arbitrary commands from standard input and publishes them to the DDS bus. This can be used for debugging and testing, or simply as an example for how to publish data that other modules can consume. A simple menu system is also included to show how to initiate common AMM actions, making this a simple command-line-interface to an AMM instance.
3. **“Kitchen Sink” example**

The “Kitchen Sink” example showcases examples of subscribing to multiple AMM data types, publishing multiple AMM data types and utilizing advanced functions of AMM such as Event Fragments.

1. **Fluidics Manager**

The fluidics manager is a reference implementation of the common fluid system, written to operate on an AMMDK board.

## AMMDK Development Environment Design

An overview of the AMMDK Architecture is shown in Figure 8. Refer to Section 4.5.1 AMMDK hardware overview.

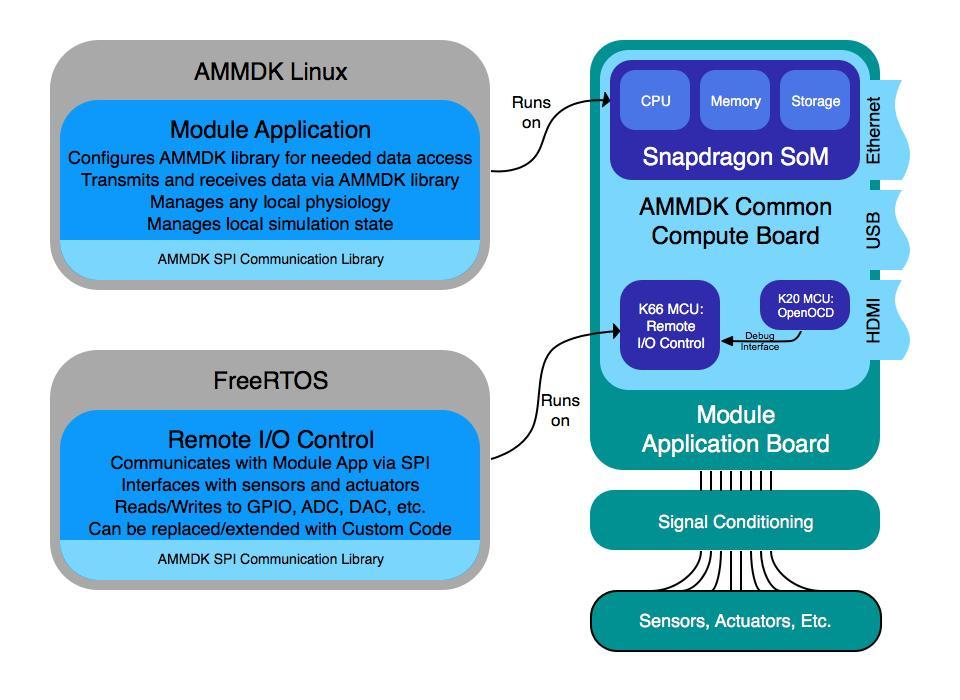


Figure : AMMDK Architectural Overview

For modules built using the AMMDK hardware, code must run in two different places in order to deliver full AMM Module functionality. First, code has to handle AMM-based communication. On the AMMDK, this code is built using the AMM Standard Library (Section 5.2) running on AMMDK Linux, which is based on Debian Linux, modified for the AMMDK-CCB.

There is an additional layer of code needed to communicate with the hardware peripherals attached to the boards. As illustrated in the AMMDK Architectural Overview, for a given AMMDK-based module, there are two different places that run AMM-specific code, the Module Application code, which runs in a Linux environment on the SoM, and the I/O Control code, which runs on the K66F and interfaces with actual hardware peripherals. The AMMDK SPI Communication Library allows these two codebases to communicate seamlessly with one another.

The AMMDK SPI Communication Library utilizes an SPI link between the SoM and the K66F MCU to synchronously pass data with relatively high throughput and low latency, certainly fast enough to be unnoticed by human faculties. The Library transmits via datagram packets while providing error correction and packet delivery guarantees. It also establishes the SPI link using the appropriate resource on the OS hosting the code.

On top of the datagram API, the AMMDK SPI Library also provides a 'Remote Control' API for the AMM Application code (running on Linux on the SoM) that looks as if the Application code is controlling the I/O Controls directly. For the remote API to function, a specific firmware needs to be flashed onto the K66F MCU, which is also provided.

## AMM User Interface Design

Reference implementation user interfaces are described fully in CDRL 0005.

# Requirements Traceability

Table 1 traces the requirements for each configuration item to the overall technical requirements identified in the original solicitation.

|  |  |
| --- | --- |
| CSCI | Reference Paragraphs in Specific Desired Capabilities |
| Module Manager | 2.b.ii. Extensible, 3.c.i. OS utilizing plug-play modular components |
| Simulation Manager | 3.c.iv. State Machine / Interactive Scenario Capability |
| Physiology Engine Manager | 2.b.v. DTME-PRP, 3.c.ii, Basic Physiology, 3.d.xii, Physiology System |
| REST Adapter | 1.d.i. Automated Data Collection |
| TCP Bridge | 2.b.ii. Extensible |
| Virtual Equipment | 3.d.xiii. Electronic Monitoring Capability |
| Command Executor | 2.b.iii, Command Line Interface |
| Fluidics Manager | 2.a.iii. Common Fluid System |

Table Requirements Traceability Matrix

# Notes

None.

# Appendix 1: Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| **A** |  |
| ACDET | Company that provides modules to AMM |
| ACS | American College of Surgeons |
| AMM | Advanced Modular Manikin |
| AMMDK | Advanced Modular Manikin Development Kit |
| API | Application Programmer's Interface |
| **C** |  |
| CAE | CAE Health Care Company |
| CDM | Common Data Model |
| CDRL | Contract Documentation Requirements List |
| CORBA | Common Object Request Broker Architecture |
| CSCI | Computer Software Configuration Item |
| **D** |  |
| DID | Data Item Definition |
| DDS | Data Distribution Service |
| DICOM | Digital Imaging and Communications in Medicine |
| DTME-PRP | Developer Tools for Medical Education Public Physiology Platform |
| **F** |  |
| FMA | Foundational Model of Anatomy |
| **G** |  |
| GUI | Graphical User Interface |
| **H** |  |
| HID | Human Interface Device |
| HLA | High Level Architecture |
| HWCI | Hardware Configuration Item |
| **I** |  |
| I2C | Inter-Integrated Circuit |
| ICD | Interface Control Document |
| ICD-10 | International Classification of Diseases 10 |
| IDD | Interface Design Description |
| IDL | Interface Definition Language |
| IGES | Initial Graphics Exchange Specification |
| IP | Internet Protocol |
| IVC | Inferior Vena Cava |
| **J** |  |
| JSON | JavaScript Object Notation |
| **L** |  |
| LMS | Learning Management System |
| LRS | Learning Record Store |
| **N** |  |
| NTP | Network Time Protocol |
| **O** |  |
| OMG | Object Management Group |
| OPB | Ontology of Physics for Biology |
| OS | Operating System |
| **P** |  |
| PhysDat | Physiology Data |
| PhysMod | Physiology Modification |
| PNG | Portable Network Graphics |
| PoE | Power over Ethernet |
| PSE | Power Sourcing Equipment |
| **Q** |  |
| QoS | Quality of Service |
| **R** |  |
| RenderMod | Render Modification |
| REST | Representational State Transfer |
| RI | Reference Implementation |
| RTPS | Real-time Publish-Subscribe |
| **S** |  |
| SDD | Software Design Description |
| SNOMED | International Health Terminology Standards Development Organization |
| SPI | Serial Peripheral Interface |
| SPS | Software Product Specification |
| SSS | System/Subsystem Specification |
| STL | Stereolithography |
| SUM | Software User's Manual |
| SWCI | Software Configuration Item (a subset of CSCI) |
| **U** |  |
| USB | Universal Serial Bus |
| UI | User Interface |
| UUID | Universally Unique Identifier |
| **X** |  |
| xAPI | Experiential API |
| XML | Extensible Markup Language |