

Modeling and Simulation of System-of-Systems Timing Constraints with UML-RT and OMNeT++

J. Bret Michael, Man-Tak Shing, Michael H. Miklaski and Joel D. Babbitt
Department of Computer Science
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943 USA
{bmichael, shing}@nps.edu, miklaskim@nimitz.navy.mil, joel.babbitt@us.army.mil

Abstract

There is a growing interest in using object-oriented analysis and design techniques in conjunction with UML to develop large complex systems. This paper presents an iterative approach for studying the timing constraints of a system-of-systems using models expressed in UML for Real-time extension (UML-RT), which are then translated into coarse-grained simulation models that are exercised using the OMNeT++ simulation engine. The integration of the UML-RT models with simulation models provides a seamless process for rapidly constructing executable prototypes for the purpose of analyzing timing constraints and deriving system requirements from those constraints. The effectiveness of the approach is demonstrated with a case study of the sensor-netting capability of a missile defense system.

1 Introduction

There is a growing interest in using object oriented analysis and design techniques in conjunction with the Unified Modeling Language (UML) [6] to develop system-of-systems. The object-oriented designs of these systems tend to be very large and complex [1,2]. Feasible requirements for large dynamic systems are difficult to formulate, understand, and meet without extensive prototyping. Modeling and simulation holds the key to the rapid construction and evaluation of prototypes early in the development process. We use an iterative process (Figure 1) that starts with Use Case analysis to identify user needs—defined as high-level system capabilities—and the construction of an object model to capture essential information about the environment in which this system will operate. However, adopting UML, we could not specify the architectural design with a formal architectural language such as MetaH [12]. Instead, in our approach we develop an object-oriented architecture of the system using UML-RT [8]. We refine the internal structures of the component systems using the Hierarchy

plus Input, Process, Output (HIPO) technique [3] until the components are readily mapped to modules of the target simulation written in OMNeT++ [11]. We use the simulation to study the feasibility and correctness of the timing requirements and apply the lessons learned to modify the system architecture and timing constraints accordingly.

In comparison to other attempts to extend UML syntax to support automatic generation of simulation code from design, such as in [7,10], our initial investigation leads us to tentatively conclude that UML-RT is much better suited for use in modeling complex system architectures. We demonstrate later in the paper that there is a straight-forward mapping between a UML-RT architectural model and the corresponding OMNeT++ simulation model, opening up the opportunity for automatic generation of simulation control codes from UML-RT models.

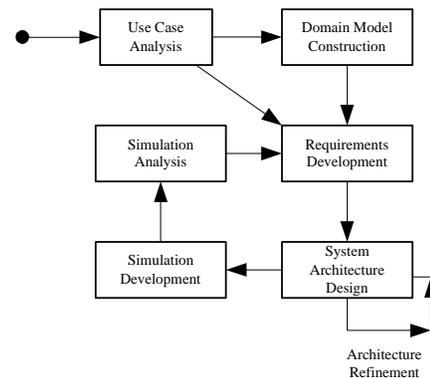


Figure 1. The Iterative Prototyping Process

The rest of the paper is organized as follows. Section 2 provides an introduction to UML-RT and Section 3 gives an overview of the OMNeT++ system. Section 4 presents a case study of the missile defense system to demonstrate the use of UML-RT in system architecture specification and their mappings to the OMNeT++ executable model. Section 5 presents a discussion of the approach.

2 UML-RT

UML-RT is an extension of UML and is based on the concepts underlying the ROOM language [9]—an architectural definition language developed specifically for complex real-time software systems. UML-RT provides three principal constructs (*capsules*, *ports*, and *connectors*) for modeling the structures of a real-time system. *Capsules* are specialized UML active objects for modeling self-contained components of a system with the following two restrictions: (1) capsule operations can only be called within the capsule and (2) capsules can only communicate with other capsules through special mechanisms called ports. *Ports* are objects within a capsule that act as interfaces on the boundary of the capsule. A capsule may have one or more ports through which it is interconnected with other capsules via connectors. *Connectors* represent communication channels through which capsules communicate via the sending and receiving of messages. Each port is associated with a *protocol* that captures the semantics of the interactions between the port and its counterpart on the opposite end of the connector.

Figure 2 shows a simple UML-RT model consisting of a set of sensor capsules, a set of sensor fusion processor capsules and a sensor net capsule. Each sensor capsule has three ports. It uses one of the ports to communicate with its associated sensor fusion processor capsule. Each sensor fusion processor capsule has multiple ports for communication with its associated sensors (as indicated by the multi-object icon) and uses a single port to communicate with the sensor net capsule. The “white-filled” icons on the sensor fusion processor capsule indicate that the sensor fusion processor capsule plays the “slave” role of a binary protocol when communicating with its associated sensor capsules. A capsule may contain collaborating sub-capsules, as shown Figure 3, and may have at most one state machine that specifies the dynamic behavior of the capsule.

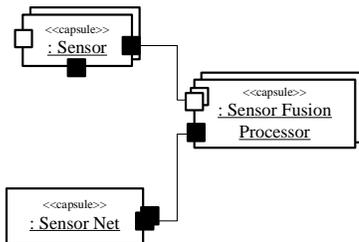


Figure 2. A UML-RT model

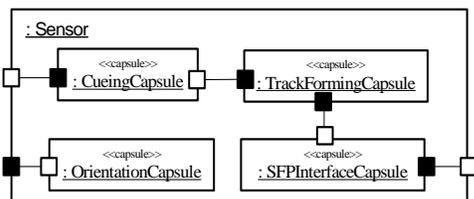


Figure 3. The internal view of the sensor capsule

3 OMNeT++

OMNeT++, which stands for *Objective Modular Network Testbed in C++*, is an object-oriented discrete-event simulator primarily designed for the simulation of communication protocols, communication networks and traffic models, and models of multiprocessor and distributed systems. Similar to UML-RT, OMNeT++ provides three principal constructs (*modules*, *gates*, and *connections*) for modeling the structures of a target system. An OMNeT++ simulation model consists of a set of modules communicating with each other via the sending and receiving of messages. Modules can be nested hierarchically. The atomic modules are called *simple modules*; they are coded in C++ and executed as co-routines on top of the OMNeT++ simulation kernel. *Gates* are the input and output interfaces of the modules. Messages are sent out through output gates of the sending module and arrive through input gates of the receiving module. Input and output gates are linked together via connections. *Connections* represent the communication channels and can be assigned properties such as propagation delay, bit error rate and data rate.

Messages can contain arbitrarily complex data structures and can be sent either directly to their destination via a connection or through a series of connections (called route). Figure 4 shows a simple OMNeT++ model consisting of five modules: a missile (*ICBM*), a ground sensor (*GroundSensor*), a satellite sensor (*SatSensor*), a sensor fusion processor (*SFP*), and the sensor net (*SensorNet*).

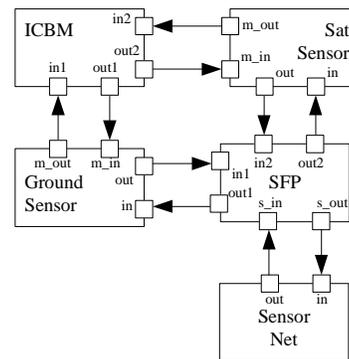


Figure 4. A OMNeT++ model

The OMNeT++ models are expressed in terms of a topology description language NED (NETwork Description). A NED file may contain a set of import statements, a set of channel definitions, a list of simple and compound module declarations, and a network definition (Figure 5). Each module can have parameters to customize module topology, module behavior, and module communication. OMNeT++ provides a Graphical Network Editor (GNED) for viewing and editing of NED files (Figure 6).

```

//-----
// file: missile.ned
//-----

// modules definitions
simple ICBM
gates:
  out: out1;
  out: out2;
  in: in1;
  in: in2;
endsimple

simple GroundSensor
gates:
  ...
endsimple

simple SatSensor
gates:
  ...
endsimple

simple SFP
gates:
  ...
endsimple

simple SensorNet
gates:
  ...
endsimple

// the Missile network model
module Missile
parameters:
  data_rates : numeric,
  radar_size : numeric,
  fused_size : numeric,
  ir_size : numeric;
submodules:
  ICBM: ICBM;
  display: "o=#f10000;p=79,59;b=36,32";
  GroundSensor: GroundSensor;
  display: "p=75,220;i=router;b=32,32";
  SatSensor: SatSensor;
  display: "p=150,150;i=router;b=34,34";
  SFP: SFP;
  display: "p=323,221;i=pc;b=38,32";
  SensorNet: SensorNet;
  display: "o=#0000ff;p=323,53;b=38,32";
connections:
  ICBM.out1 --> delay .5ms --> GroundSensor.m_in;
  ICBM.in1 <-- delay .5ms <-- GroundSensor.m_out;
  GroundSensor.out -->
    delay 250ms datarate data_rates --> SFP.in1;
  GroundSensor.in <--
    delay 250ms datarate data_rates <-- SFP.out1;
  ICBM.out2 --> delay 130ms --> SatSensor.m_in;
  ICBM.in2 <-- delay 0ms <-- SatSensor.m_out;
  SatSensor.out -->
    delay 500ms datarate 93000 --> SFP.in2;
  SatSensor.in <--
    delay 0ms datarate 93000 <-- SFP.out2;
  SensorNet.out --> delay 250ms --> SFP.s_in;
  SensorNet.in <-- delay 250ms <-- SFP.s_out;
  display: "p=10,10;b=405,265";
endmodule

// Instantiates a Missile network.
network UseCase1 : Missile
parameters:
  radar_size = input(100000, "Size of Radar Track File"),
  fused_size = input(200000, "Size of Fused Track File"),
  ir_size = input(1000, "Size of IR Contact Report"),
  data_rates = input(1024000, "Radar to SensorNet Data Rate");
endnetwork

```

Figure 5. The Network Description file

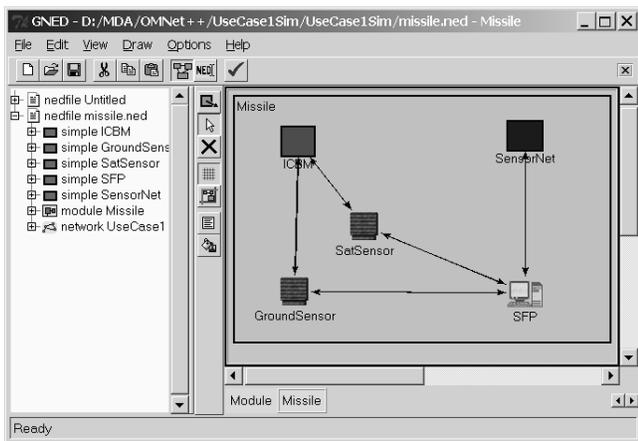


Figure 6. The Graphical Network Editor

In addition to GNED, OMNeT++ also provides a compiler (NEDC) to generate the simulation control code and a user-interface from the NED files. The control code calls the functions provided in the OMNeT++ simulation kernel library and allows the user to control the simulation execution either via a command-line (Cmdenv) or a graphical user (Tkenv) interface. Data from the simulation executions

are saved as specially formatted text files which, after some preprocessing using sed, awk or perl, can be read into math packages such as Matlab or Octave, or imported into spreadsheet programs. OMNeT++ also provides a vector-plotting tool (Plove) for filtering (e.g., averaging, truncation, smoothing) the output data and displaying the results as graphs.

4 Missile Defense System – A Case Study

In this section, we illustrate the iterative approach with a hypothetical ballistic missile defense system (BMDS). The BMDS is an integrated system-of-systems which provides a layered defense that employs complementary sensors and weapons to engage threat targets by land, sea, air, or space in the boost, midcourse, and terminal phases of flight, and incrementally deploying that capability. In parallel, sensor suites and the battle management and command and control (BMC2) software will be developed to form the backbone of the BMDS.

4.1 Use Case Analysis

To understand the requirements and constraints of the proposed system, we developed six UML use cases to identify the external agents and systems that are involved in a typical missile-defense scenario and the necessary interactions between these entities:

1. Detect Potential Threat Ballistic Missile - The goal of this use case is to detect possible threat ballistic missiles, and push the track data onto the sensor net.
2. Generate and Transmit a Local Track - This is a sub-use case of 1. The goal of this use case is to have a sensor generate a local track based on valid detection parameters of the sensor.
3. Cooperatively Track and Classify Threat Ballistic Missiles - The goal of this use case is to identify and type-classify the threat ballistic missiles, develop fire-quality tracks for engagement solutions, and forward the target track list to Weapons Net.
4. Cooperative Weapons Assignment - The goal of this use case is to assign targets to weapons via cooperative target bidding.
5. Engage Targets - The goal of this use case is to engage threat ballistic missile.
6. Assess Kill - The goal of this use case is to determine the kill status of the threat ballistic missile.

We then developed sequence diagrams based on the use cases to identify the flow of events and messaging between the external agents and the BMC2 system. Figure 7 shows the sequence diagram for use cases 1 and 2, where the flow

of events begins with the assumption that a ballistic missile threat exists and that there is a sufficient amount of time to conduct deliberate planning prior to the anticipated first available launch window. In this instance the commanders, via the BMC2 and Sensor Net, issue a warning in the form of cueing messages for sensors to observe a specific region. Once a missile is detected, the sensor commences continuous tracking of the missile and forwards a cueing message to the BMC2 and Sensor Net so that other sensors can detect and track the missile.

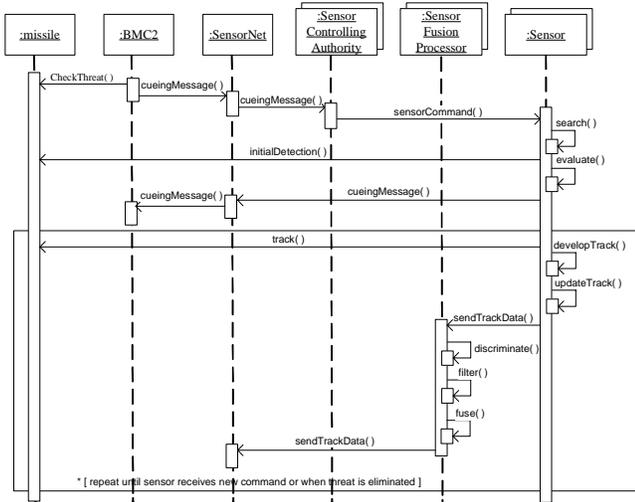


Figure 7. The sequence diagram for use cases 1 and 2

We refer the readers to [4,5] for details of the use cases and the rest of the sequence diagrams.

4.2 A Distributed BMC2 Architecture

The huge complexity, physically dispersed geography, and distributed nature of global ballistic missile defense necessitate a distributed approach to ballistic missile defense battle management. Based on the use cases, we developed the top level of a distributed architecture shown in Figure 8, along with the corresponding UML-RT model shown in Figure 9.

The overarching BMC2 System will consist of a loosely coupled set of regional BMC2 systems; geographically separated networks interconnected much like the Internet. The intent is to allow all participants to pull the information from specific regions as desired, but also to ensure that time-critical information can be pushed to those geographically collocated units that need it to effect destruction of a threat missile or to hand-off the information to non-geographically collocated units as a missile transits from one region to another.

Each regional BMC2 system consists of three major sub-systems: a C2BMC node, a Sensor Net and a Weapons Net,

where the C2BMC node refers to the automation support for the Command/Control, Battle Manager and Communication (C2BMC) functions, the Sensor Net refers to a distributed system that provides the sharing of track data among Sensor Fusion Processors, Weapons Net, Weapon Platforms and the C2BMC node, and the Weapons Net refers to a distributed system for cooperative target assignment.

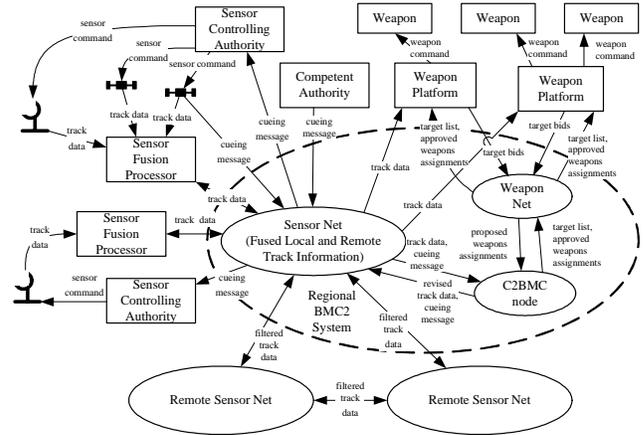


Figure 8. A distributed BMC2 architecture

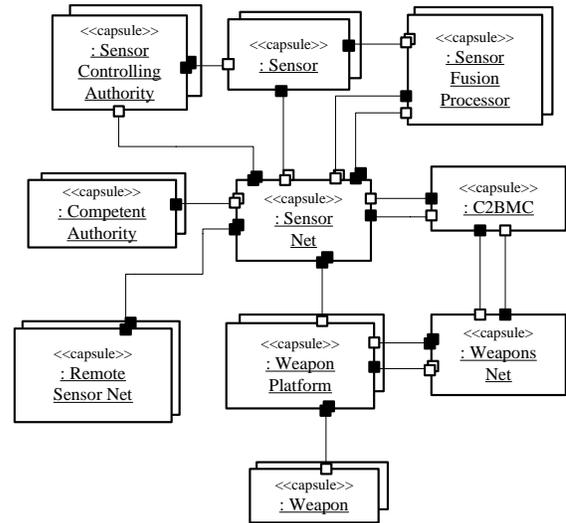


Figure 9. The UML-RT model for the BMC2 architecture

We refined the internal structures of the Sensor capsule, the Sensor Net capsule and the Sensor Fusion Processor capsule using the HIPO technique. Figures 10 shows the internal structure of the Sensor Fusion Processor (SFP) capsule, which consists of five sub-capsules (Sensor Interface, Track Fusing, Collaborative Fusion, Track List and Sensor Net Interface).

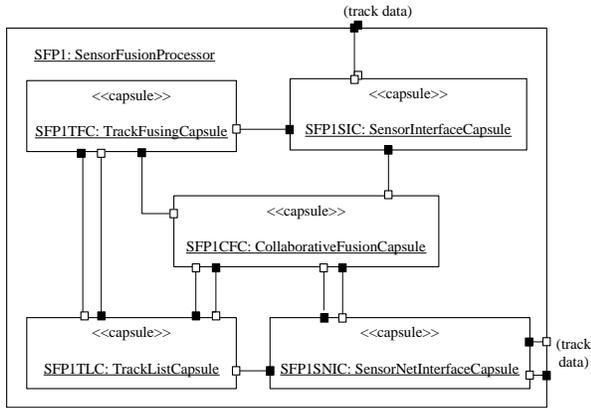


Figure 10. The internal structure of the Sensor Fusion Processor capsule

The Sensor Interface capsule serves as the primary interface to all assigned sensors. It sends all tracks to the Track Fusing capsule if it is receiving data from more than one sensor; otherwise, it passes the data directly to the Collaborative Fusion capsule. The Track Fusing capsule takes multiple tracks per target from the Sensor Interface capsule, correlates or fuses them into one single track per target in real time. It also performs track discrimination as a backup to the sensor's native discrimination capability to prevent overload on the Sensor Net. The Collaborative Fusion capsule takes fused or raw local tracks (one per target) and fuses them with tracks received from other SFPs via the Sensor Net. The Track List capsule is responsible for compiling and providing the internal list of tracks for the SFP and preventing duplicates. It provides this data to both the Track Fusing capsule and the Collaborative Fusion capsule, and provides this information to other SFPs upon request via the sensor net. It also serves as a repository for commands received from the Sensor Net. The Sensor Net Interface capsule is responsible for pushing tracks from the Track List capsule to the Sensor Net and routing the incoming tracks from other SFPs via the Sensor Net to the Collaborative Fusion Capsule. (Readers can refer to [5] for the complete UML-RT models.)

4.3 The OMNeT++ simulation model the Sensor Fusion Processor

In order to perform a systematic analysis of the BMDS timing constraints and requirements, we developed an OMNeT++ discrete event simulation of the Sensor Fusion Processor (SFP) as a model for simulating the entire system, using the UML-RT model as a template for incorporating system requirements based on the documented artifacts. The simulation is used to determine whether the requirements have been achieved, that the system operates within acceptable parameters, and to discover any other possible timing considerations and constraints.

As numerous sensors provide track data at asymmetric rates, the SFP must discriminate, filter, and fuse the current track data into a single track and forward that track out to the Sensor Net. The SFP must also compare that track with a track database to determine if it is to be used to update an existing track or develop a new track. Additionally, developing a rough track classification requires a table look up, data comparison, and association based on known parametric data.

The OMNeT++ simulation shown in Figure 11 was created to identify potential bottlenecks at the Sensor Fusion Processor. It consists of twelve modules simulating the five sub-capsules of the Sensor Fusion Processor interfacing with two satellite Sensor capsules, four ground radar Sensor capsules and the Sensor Net capsule. The internal structures of these capsules are mapped to the C++ code of the corresponding modules. Figure 12 shows the graphical user interface for the simulation model shown in Figure 11.

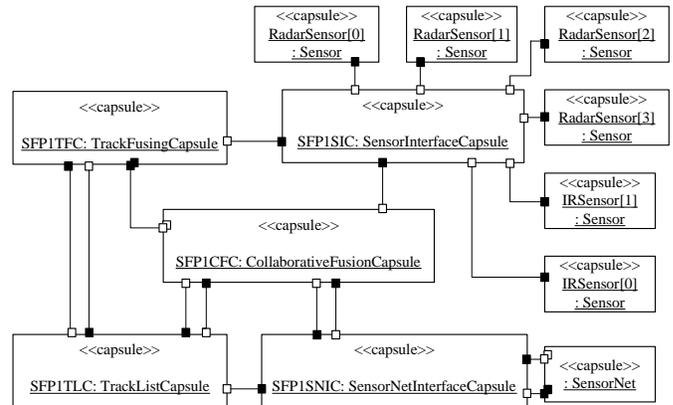


Figure 11. The OMNeT++ Model the Sensor Fusion Processor

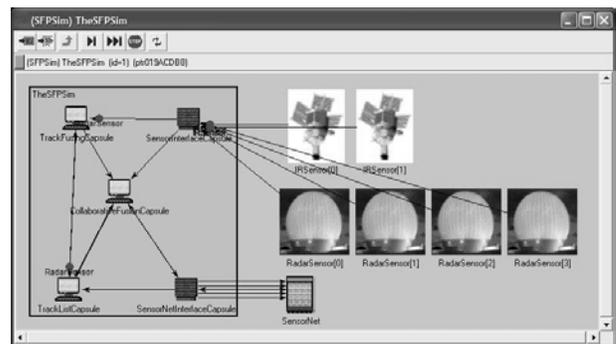


Figure 12. The Graphical User Interface

4.4 Simulation Results

A parametric (i.e., sensitivity) analysis involving multiple simulation runs was conducted to determine what were the most significant timing constraints on the system and at

what point they became critical. A methodical approach was utilized in the process of obtaining data where one input value was varied and the others remained constant to see how that one variable impacted the system. For data input values (data rates, track message sizes, sensor update delay, number of sensors, collaborative fusion requests, module processing time, track list access time, master track list broadcast times and track fusion time), we utilized commercial data-transmission rates and approximate system clock-speed values for internal timing. In doing so we abstracted the data points and precluded any implication of existing or developmental systems, while still obtaining valid research data. Tables 1 through 12 summarize the results of the simulation runs.

	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
5 Tracks								
56.6 Kbps	0.13	0.06	0.0002	0.0005	0.027	0.00008	0.0001	0.00009
1.44 Mbps	0.13	0.055	0.0002	0.0004	0.027	0.00008	0.0001	0.00009
120 Mbps	0.13	0.055	0.00022	0.00045	0.027	0.00008	0.0001	0.00009
50 Tracks								
56.6 Kbps	0.27	0.21	0.0022	0.0045	0.27	0.0008	0.0006	0.0005
1.44 Mbps	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
120 Mbps	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
100 Tracks								
56.6 Kbps	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
1.44 Mbps	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
120 Mbps	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
500 Tracks								
56.6 Kbps	16	11	0.022	0.032	0.92	0.002	0.001	0.0008
1.44 Mbps	16	11	0.022	0.032	0.92	0.002	0.001	0.0008
120 Mbps	16	11	0.022	0.032	0.92	0.002	0.001	0.0008

Table 1. Varying Data Rate

	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
5 Tracks								
1024 bits	0.13	0.055	0.0002	0.0004	0.027	0.00008	0.0001	0.00009
512 bits	0.13	0.055	0.0002	0.0004	0.027	0.00008	0.0001	0.00009
50 Tracks								
1024 bits	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
512 bits	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.00058	0.0005
100 Tracks								
1024 bits	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
512 bits	0.46	0.4	0.005	0.009	0.53	0.0016	0.0011	0.00095
500 Tracks								
1024 bits	16	11	0.022	0.032	0.92	0.002	0.001	0.0008
512 bits	16	11	0.023	0.031	0.94	0.0023	0.001	0.00085

Table 2. Varying Track Message Sizes

Radar Delay (sec)	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
2	0.34	0.24	0.0007	0.0015	0.09	0.0003	0.00025	0.0002
1	0.29	0.22	0.0013	0.0025	0.15	0.00045	0.00035	0.0003
0.5	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.1	6	6	0.01	0.018	0.9	0.0025	0.0015	0.0013
0.01	15	13	0.1	0.011	0.98	0.0018	0.00025	0.0003

Table 3. Varying Ground-based Radar Update Delay

IR Delay	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
5	0.225	0.205	0.0021	0.0042	0.25	0.0009	0.0006	0.0005
2	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1	0.34	0.23	0.0025	0.005	0.3	0.0009	0.00062	0.00055
0.5	0.42	0.25	0.003	0.006	0.35	0.001	0.0007	0.00061
0.1	0.55	0.26	0.007	0.014	0.81	0.0022	0.0014	0.0013
0.01	15	2.5	0.05	0.06	0.95	0.0017	0.00035	0.00033

Table 4. Varying Space-based IR Update Delay

Radar #	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
4	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
10	0.35	0.35	0.0055	0.01	0.58	0.0008	0.00053	0.00047
20	3.2	2.1	0.01	0.019	0.96	0.0007	0.0004	0.00038

Table 5. Varying Number of Ground-based Radar Sensors

IR #	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
1	0.25	0.21	0.0022	0.00453	0.25	0.001	0.00068	0.00057
2	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
4	0.34	0.22	0.0025	0.005	0.28	0.00065	0.0005	0.00042
10	0.52	0.26	0.0034	0.0065	0.35	0.00045	0.00032	0.00029

Table 6. Varying Number of Space-based IR Sensors

# of Collaborative Fusion Requests from CFC	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
1	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
5	0.71	0.36	0.0021	0.005	0.27	0.0011	0.0012	0.0008
10	0	0.41	0.0021	0.0045	0.27	0.0011	0.0012	0.0008
100	0	0.41	0.0021	0.0045	0.27	0.0011	0.0012	0.0008
1000	0	0.41	0.0021	0.0045	0.27	0.0011	0.0012	0.0008

Table 7. Varying Collaborative Fusion Requests

Time each Module Handles a Track	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
0.000005	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.0005	0.34	0.21	0.0023	0.0045	0.53	0.08	0.059	0.051
0.05	22	6	0.99	0.99	0.99	0.3	0.6	0.5

Table 8. Varying Module Processing Time

Time to Check Track Against List	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
0.0005	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.001	0.4	0.33	0.0023	0.0045	0.52	0.00082	0.0006	0.0005
0.005	13	8	0.0023	0.003	0.94	0.00022	0.00014	0.00012

Table 9. Varying Track List Access Time

Time to Perform Fusion	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
0.01	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.05	0.75	0.85	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.1	6.5	8	0.0023	0.0045	0.27	0.0005	0.00035	0.0003

Table 10. Varying Time to Perform Track Fusion

Delay Between Master Track List Broadcasts	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SMIC %	SN.SFP %
10	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.00055	0.00045
1	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.00055	0.00045
0.1	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.01	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0011	0.00097
0.001	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0056	0.0054

Table 11. Varying Master Track List Broadcast Times

Data Rate Between Capsules	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SMIC %	SN.SFP %
1E+11	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1000000000	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1000000	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1000	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005

Table 12. Varying Data Rate between Capsules

The most significant timing issue that was obtained through multiple iterations of the simulation was that as the track load increased, whether it was from large numbers of tracks being reported, moderate numbers of tracks being reported by large numbers of sensors, or when the sensors increased their update rates, the Track List capsule (LTC) which is responsible for maintaining and broadcasting the update track list would become saturated, thus increasing the time to transmit track data. This is evidenced by corresponding increases of both the TLC’s utilization and average time to broadcast tracks on the Sensor Net with an increase in the number of tracks reported or sensors updating.

We observed that track message size and data throughput rates had little impact on the time to transmit track data. Additionally, as the number of track-collaboration requests increased, the number of normal tracks dropped to zero and collaboratively-fused tracks increased, but the impact to the overall average track-process time is insignificant. As to be expected, increases in module-processing time, track-list-comparison times, and track-fusion times all had corresponding increases to the average track processing and throughput values. (Reader can refer to [5] for the detail data tables and line graphs.)

5 Discussions and Conclusions

In a field of study that is not well defined such as ballistic missile defense and which consists of systems of systems, one must discover and develop methodologies for refining requirements and ensuring a project’s purpose is successfully accomplished. The Use Case-Model-Simulation feedback cycle that we used is a systematic engineering methodology for developing such highly complex systems of systems. Through the use of this methodology, we found it necessary to redesign the Sensor Fusion Processor’s Track List capsule (LTC) in order to handle heavier work loads, in this case more traffic. Figure 13 shows the UML-RT model

of the original design where it uses a single TFC-CFC Communications capsule to handle all local data streams received from the Track Fusing capsule and the Collaborative Fusion Capsule, and uses a single Track Registry capsule to maintain the SFP’s master list of all perceived valid tracks as well as any additional tracks received from the Sensor Net.

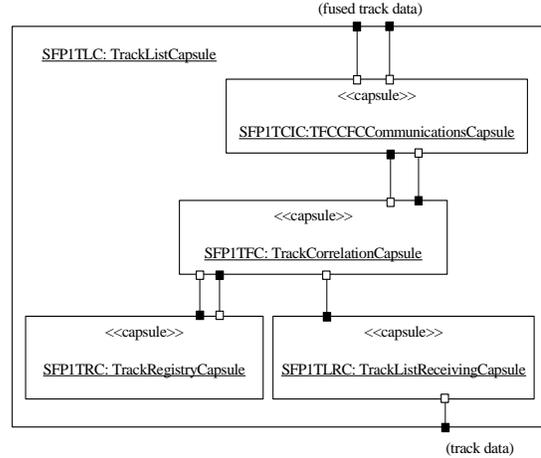


Figure 13. The Original Track List Sub-capsule

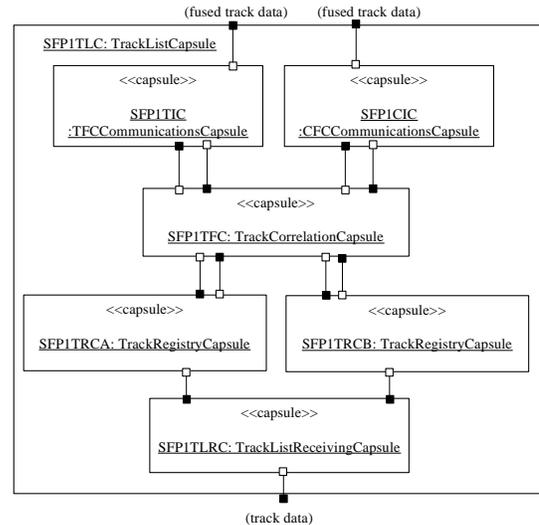


Figure 14. The Modified Track List Sub-capsule

Figure 14 shows the revised design where it uses two capsules (the TFC Communications capsule and CFC Communications capsule) to handle the local track data. The new design uses two Track Registry capsules (A and B) working in tandem to maintain the SFP’s master list of all perceived valid tracks as well as any additional tracks received from the Sensor Net, and allows them to communicate with Track List Receiving capsule directly. Only one Track Registry capsule is active at a time. The other is in a semi-active state, in which it is receiving all updates from the Track

Correlation capsule but its data is not being used by the Track Correlation capsule until it receives a copy of the master track list from the Track List Receiving capsule that is newer than the one held by its active counterpart. When that happens, it goes active and directs the other active capsule to go into semi-active mode.

Based on the results of the simulation with the redesigned TLC, we conclude that the redesign is not sufficient, and that a redesign of the Sensor Fusion Processor as a whole, which would probably place a slave TLC off of the master TLC local to each of the fusing capsules, would be required to reduce the load on the Track List capsule and prevent the system from bottlenecking there. This type of feedback-refinement loop is key to ensuring success in the development of any complex system.

We found that the use cases feed directly into the UML-RT models, which in turn flow directly into the OMNeT++ simulation. OMNeT++, being an open-source project, is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings. In addition to its rich set of tools for the construction, execution, and analysis of discrete-event simulations, there is a rapidly growing library of reusable models and code for rapid construction of simulations. The mappings between the capsules, ports and connectors of the UML-RT model and the modules, gates, and connections of the OMNeT++ models are straight forward and can be easily automated with a simple translation tool.

Acknowledgements and Disclaimer

The research reported in this article was funded by a grant from the U.S. Missile Defense Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotations thereon.

References

- [1] Caffall, D. S. and Michael, J. B. A New Paradigm for Requirements Specification and Analysis of System-of-Systems. In Wirsing, M., Balsamo, S. and Knapp, A., eds. *Lecture Notes in Computer Science*, no. 2941 (*Proc. Monterey Workshop 2002: Radical Innovations of Software and System Engineering in the Future*), Berlin: Springer-Verlag, 2004.
- [2] Caffall, D. S. Conceptual Framework Approach for System-of-Systems Software Developments. Master's thesis, Naval Postgraduate School, Monterey, Calif., Mar. 2003.
- [3] HIPO – A Design Aid and Documentation Technique. Report no. GC20-1851-0, IBM Corp., White Plains, N.Y., 1974.
- [4] Michael, J. B., Pace, P., Shing, M. T., Tummala, M., Babbitt, J., Miklaski, M., and Weller, D. Test and Evaluation of the Ballistic Missile Defense System: FY 03 Progress Report. Tech. Report NPS-CS-03-007, Naval Postgraduate School, Monterey, Calif., Sept. 2003.
- [5] Miklaski, M. H. and Babbitt, J. D. A Methodology for Developing Timing Constraints for the Ballistic Missile Defense System. Master's thesis, Naval Postgraduate School, Monterey, Calif., Dec. 2003.
- [6] Object Management Group. *OMG Unified Modeling Language (UML) Specification 1.5*, March 2003. <http://www.omg.org/technology/documents/formal/uml.htm>
- [7] Savino-Vazquez, N.-N. and Ruigjaner, R. A UML-based method to specify the structural component of simulation-based queuing network performance models. In *Proc. Thirty-second Annual Simulation Symposium*, IEEE (San Diego, Calif., Apr. 1999), pp. 71-78.
- [8] Selic, B. and Rumbaugh, J. Using UML for Modeling Complex Real-Time Systems. Unpublished white paper, Apr. 4, 1998, <http://www.rational.com/media/whitepapers/umlrt.pdf>.
- [9] Selic, B., Gullekson, G., and Ward, P. *Real-Time Object Oriented modeling*. New York: John Wiley & Sons, 1994.
- [10] Speirs, N. A. and Arief, L. B. Simulation of a telecommunication system using SimML. In *Proc. Thirty-third Annual Simulation Symposium*, IEEE (Washington, D.C., Apr. 2000), pp. 131-138.
- [11] Varga, A. OMNeT++ Discrete Simulation System (Version 2.3) User Manual, Technical University of Budapest, Dept. of Telecommunications (BME-HIT), Hungary, Mar. 2002.
- [12] Vestal, S. MetaH User's Guide. Honeywell Technology Center, Minneapolis, Minn., www.htc.honeywell.com/metah.