Strategic Airlift

Operationalizing Constructive Simulations

Rob Barwell
robbarwell@cmail.carleton.ca

ABSTRACT

The evolution of computers have enabled large organizations to expand their use of modeling and simulation and increase their modeling of complex systems. This evolution has also resulted in a lack of simple tools and processes to define models and experiments which hinders wide scale user adoption of modeling and simulation to solve many real world problems. This paper introduces a process and tools similar to the computer science DEVOPS life cycle concept, to take a model from conception to conclusion that an average operator can use. This will be illustrated by exploring the Arctic logistical support base problem and implemented as a combination of microservices into an application called Sim Manager. An instance of the Arctic logistic support base problem is what location is most ideal to re-supply various Arctic locations with people and goods.

1 INTRODUCTION

The evolution of computers have enabled large organizations to expand their use of modeling and simulation and increase their modeling of complex systems and general modeling and simulation capabilities. In addition to higher fidelity models, this evolution also supports the increase of experimentation on any given model. The operator can run an unlimited number of experiments bounded only by the computing power available to them. This evolution has also resulted in an ever increasing and complex process to develop a model, run a model, refine a model, re-run a model, and visualize a model. A lack of simple tools and processes to define models and experiments has hindered wide scale user adoption of modeling and simulation to solve many real world problems. This paper will introduce a process and tools similar to the computer science DEVOPS life cycle to take a model from conception to conclusion that an average operator can use. This will be illustrated by exploring the Arctic logistical support base problem and implemented as a combination of microservices into an application called Sim Manager.

2 BACKGROUND

2.1 Modelling and Simulation Process

Historically, the focus of research in the modelling and simulation field has been formalisms to define various systems. These formalisms started to converge with the introduction of the Discrete Event Systems Specifications (DEVS) formalism introduced in the 1970s by Bernard P. Zeigler[10]. DEVS models systems as submodels called atomic models which can be used to express a system at any level of granularity. These independent atomic models can then be coupled together to form a system of systems. Other areas of modeling and simulation have been studied in a more limited capacity such as the modeling and simulation life cycles.

Some academics such as Osman Balci have created extensive process flows such as the one in figure 1 [1]. These process flows

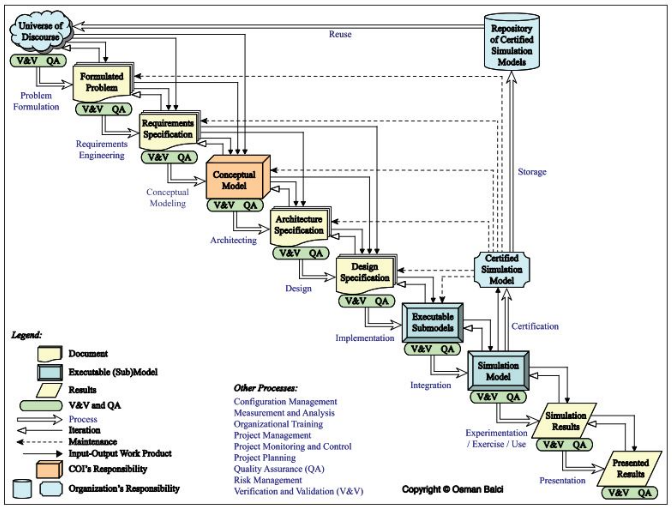


Figure 1: A Life Cycle for Modeling and Simulation

have been expanded and refined by other scholars including Margaret L. Loper in the book The Modeling and Simulation Life Cycle Process[3]. Many of these models try and distil down the steps required to build and execute a model. Loper provides a clear consolidation of these processes by formulating a representative life cycle process, which includes: "establishing purpose and scope, formulating a conceptual model, acquiring and analyzing data, developing the simulation model and program, verifying and validating the model and simulation, designing experiments, executing the simulation and analyzing output, and configuration control" [3]. Which ever process is used to generate experiments usually requires a deep background knowledge of modeling and simulation, specifically in the area of analyzing data, developing the model and program, and verifying the model and simulation. This reduces the ability for operators with limited knowledge of modeling and simulation to conduct experiments.

Computer scientists encountered the same issues in the late 2000's as the development process kept expanding due to the introduction of cloud and virtual computing and switch from waterfall to agile development methodologies. There was an increased requirement to reduce the time between committing a code change and seeing that change reflected in a production environment while ensuring quality code[2]. This resulted in the introduction of the DEVOPS process. This helped developers unfamiliar with the build and deployment process to easily make changes to the code and have it move through the process automatically to a production environment while maintaining checks and balances to ensure a high quality product. The modeling and simulation community could develop a process similar to DEVOPS and allow operators with limited training the ability to define models and run experiments using them. This paper will focus on trying to address the issue of having an average operator define a coupled model with very limited programming experience. It will also look at how to progress the model through a basic modeling and simulation process to produce results where the user can gain knowledge and draw conclusions.

Developing a simplified expression of a couple model requires a structured format which most operators can understand and express, while also maintaining a format that is easy to understand for a computer. Currently, there are two primary data interchange formats which are commonly used in web application. These are XML and JSON. Both are able to transport a wide variety of data, however JSON allows a flexible and agile definition of a data object, where XML must be well formed and follow a defined structure[5]. This paper will present an XML structure for defining coupled models which provides the structure required to guide an operator in re-using existing atomic models with limited training and knowledge.

The field of human computer interaction (HCI) explores how humans interact with computers and has a fundamental base in cognitive psychology. There has been a wide variety of research conducted in the area of how humans translate data points into knowledge and specifically how to increase the transfer rate from raw data into useful knowledge. One of the primary references for this field is Colin Ware's Information Visualization: Perception for Design textbook. One of the key points in the book is short term memory can typically only hold between 3-5 objects at any time[8], however there are a number of enhancements which can be applied to increase the number. This presents challenges in the modeling and simulation field where a single experiment could produce an exponential amount of data. The model used as an example in this paper is primarily focused on geographical information. Therefore a map visualization will be used to demonstrate how mapping simulation output to an easy to understand visualization increases the ability for an average operator to conduct experiments.

Combining together the pieces of converting an XML to a format the simulator understands, running the simulator, and taking the simulation output to a visualization such as a map requires an underlying framework or system. This has been explored recently by Sixuan Wang when they developed CloudRISE as part of their thesis MAMSaaS: Mashup Architecture with Modeling and Simulation as a Service. The thesis provides a detailed overview of technology trends with simulation as a service and highlights the transition of many simulators to web services based on REST web services[7]. This paper will follow the same trend using web services and REST APIs to couple all the components together and introduce the concept of containerization to the process. Containers have existed since the late 1970s and didn't become mainstream until the mid 2010's with the emergence of Docker. A recent article in Network World concisely highlights the use cases for containers versus virtual machines. Containers are used to run single applications such as microservices and virtual machines are used to run multiple applications[6]. This paper will demonstrate how to take a basic modeling and simulation process and turn it into individual components. These components can be translated into individual microservices and then containerized. This results in an improved solution from using virtual machines by allowing individual components within the modeling and simulation process to be changed out easily due to their loose coupling. For example if the user wanted to use another simulator they could simply swap out one simulator microservice and introduce another simulator without having to change or re-deploy other parts of the modeling and simulation process. Successful containerization will require defining loosely coupled services that can easily communicate via REST APIs.

2.2 Arctic Logistical Support Base Problem

Recently in 2017, the Government of Canada clearly articulated the tasks it wants the military to conduct and how often it wants the military to conduct those tasks[4]. This resulted in many questions about whether the Canadian military has enough people and resource to accomplish their missions. Using the assumption that there are enough people and resources to conduct concurrent operations the questions then expand into the area of strategic airlift and whether Canada could sustain the supply chains required for these missions. Strategic airlift is defined as the ability to rapidly transport a large number of passengers and/or over-sized heavy cargo over long distances within Canada or between Canada and a theatre of operations[9].

A simple instance of this problem is the Arctic logistical support base problem. The Canadian military conducts operations in the Arctic and requires strategic airlift to bring people and resources to various locations to support missions. Typically, most cargo originates in Trenton, Ontario and then requires transport to an intermediate logistical support base in the north and then forwarded on to its final destination. The question then arises as to where should this base be located and how many aircraft are required to sustain the mission.

This question has two potential solutions. One solution would focus on where to permanently establish a support hub, where the other solution could be a temporary support hub. The temporary support hub would only be used for the duration of a mission. This paper will focus on a temporary support hub as the permanent support hub problem would require historical data that is beyond the scope of this paper.

2.2.1 Alternative Application of Model. The Arctic problem described above is one instance of the strategic airlift problem. Below are other examples that were envisioned for this model.

- Determining the throughput of pallets to deployed operations using various locations and various numbers of aircraft;
- Identifying options for search and rescue locations;
- Identifying bottle necks in the supply chain by determining which locations are the slowest to process pallets;
- Extending the model to do land and sea transportation problems; and
- Applying the model to non-military application such as intermodal logistical transportation.

3 STRATEGIC AIRLIFT MODEL DESCRIPTION

3.1 Conceptual Overview

The strategic airlift model was coarsely defined by design. This ensures the model can be applied to many instances of the strategic airlift problem. The design includes two atomic models, a location model, and an aircraft model. These atomic models can be combined together as shown in figure 2. Any number of locations can be defined and serviced by any number of aircraft.

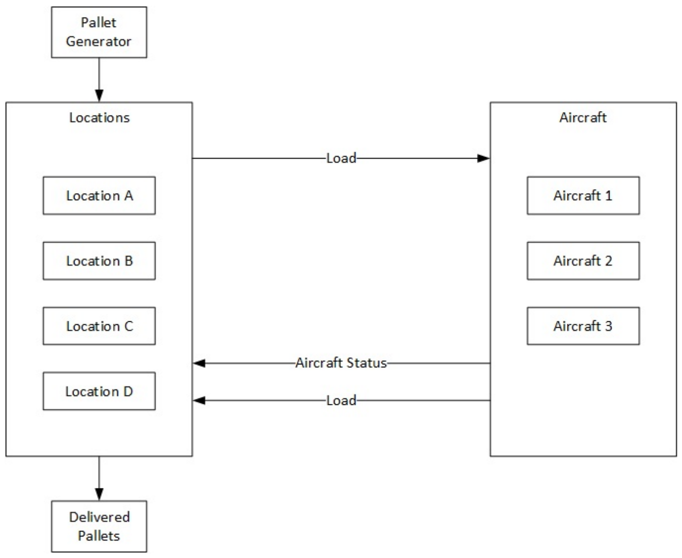


Figure 2: Strategic Airlift Coupled Model

The model begins by receiving a pallet at a given location. A pallet is used to represent a collection of people or goods which require transportation on an aircraft. A pallet is the smallest unit of representation in the model and an aircraft can carry a fixed number of pallets together. Each location keeps track of which aircraft are available at that particular location at that point in time. The aircraft then waits for a load to be ready for transport from the location. The location waits until it receives enough pallets to fill at least half the aircraft. Once a location has enough pallets, it groups them together and produces a load to be transported to another location and dispatches it to a waiting aircraft. The waiting aircraft then flies the load to a destination location and announces it has arrived. After the aircraft arrives, it unloads all pallets to the destination. At the destination location the pallets are either delivered if they are at the final location or added to the waiting pallet queue. Pallets wait in the queue until there is an aircraft available to transport them to the next location.

3.2 Assumptions

Simplicity of the model was a key factor in the design. The following assumptions were made:

- Pallets will each have a size of 1 and can be shipped with any other pallet;
- Each location has an infinite ability to store and process pallets;
- Aircraft are serviceable and available at all times;
- Aircraft fly directly between two locations. The airspeed for each aircraft can be reduced to account for non-straight line routing to avoid diplomatic airspace boundaries and account for other factors such as weather;
- Pallets are already prioritized by pallet id;
- It is assumed there is enough gas to fly directly from any two points provided by the user in the scenario; and
- The smallest granularity of time required by the model is every minute.

3.3 Model Features

This model has a number of beneficial features.

- The number of pallets each aircraft holds and how fast it travels can be configured per aircraft;
- Using the location and aircraft atomic model, the operator can create an experiment with unlimited numbers of each type.

3.4 Location Atomic Model

There is an initialization phase before locations receive and process pallets. This phase is used to provide key information required for the model to run correctly. This includes setting up routes by processing Location Information Messages. Typically, these messages arrive at time 0, however the model is flexible and routes can change dynamically throughout the experiment if required. The user would only need to change the time when Location Information Messages are sent. Further information about Location Information Messages is available in section 3.6.4. During the simulation if the model is unable to find a route between the current location and destination the pallet will be delivered with a next destination of -1 which denotes a failure to deliver the pallet.

After the initialization phase the location begins receiving pallets of people and goods to transport to a destination location. These pallets are stored in the waiting pallets queue as shown in figure 3. The pallets remain in the queue until an aircraft is available. Locations become aware of aircraft through Aircraft Status messages. When a location receives an Aircraft Status message, that aircraft is added to the waiting aircraft queue.

Load generation requires a waiting aircraft and at least enough pallets to fill half the aircraft capacity. Once these criteria are met the location retrieves the highest priority pallets which are defined by sequential pallet id, with the lowest being the most important. It then takes the first pallets by priority for the given destination location from the queue and generates a load. This load produces a Load messages which is sent to the waiting aircraft. The aircraft is then removed from the waiting aircraft queue.

When a location becomes aware an aircraft has arrived via the Aircraft Status message, it is unloaded. This is accomplished by the aircraft sending the location a Load message. The Load message contains all the pallets from the aircraft load. The location then proceeds to sort through the pallets and either delivers them if they are at the destination, or places them in the waiting pallets queue for forwarding onto the next location.

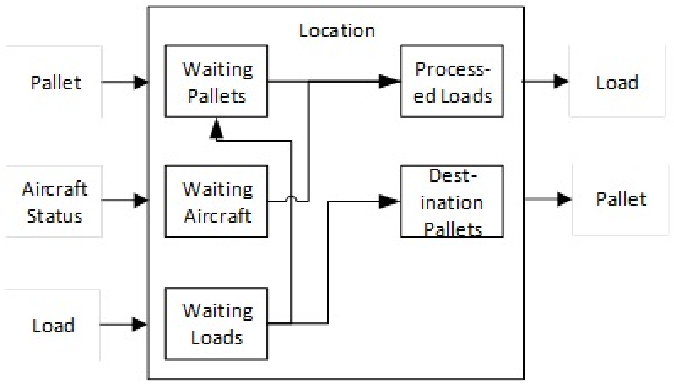


Figure 3: Location Atomic Model

3.4.1 *Formal Definition.* **Location = <X, Y, S, ta, δext, δint, λ>**

State Variables

```
Phase = Waiting
Sigma = ∞
Location Name = Passed to constructor
Location ID = Passed to constructor
Location Latitude = Passed to constructor
Location Longitude = Passed to constructor
Connections = Set via Location Information messages
Waiting Pallets = {}
Waiting Aircraft = {}
Waiting Loads = {}
Destination Pallets = {}
Processed Loads = {}
```

Formal Specifications

```
X = {Pallet ∈ <N, N, N> | Aircraft Status ∈ <N, N, N,
    ↪  N, N, Time> | Load ∈ <String, N, R, R, R, R,
    ↪  N, vector<Pallet>> | Location Information ∈
    ↪ <N, N, N, R, R, N>}
Y = {Pallet ∈ <N, N, N> | Load ∈ <String, N, R, R, R,
    ↪  R, N, vector<Pallet>>}
S = {{{Phase, Sigma, Location Name, Location ID,
    ↪ Location Latitude, Location Longitude,
    ↪ Connections, Waiting Pallets, Waiting
    ↪ Aircraft, Waiting Loads, Destination Pallets,
    ↪  Processed Loads}}
```

Pseudo Code

```
δext (mbs, e, x <Pallet | Aircraft Status | Load |
    ↪ Location Information>)
{
If x is Pallet and destination is this location
       Set next location on pallet. Use -1 for no
           ↪ location found.
       If there is a next location
              Add to waiting pallet queue
       Else if -1
              Add to destination pallet queue
If x is Aircraft Status and destination is this
    ↪ location
       Add to waiting aircraft queue
       Set aircraft wait time to 0
If x is Load and destination is this location
       Add to waiting load queue
If x is Location Information and location id is this
    ↪ location
       Add to Connections

Phase = remain in current phase, or PROCESSING if
    ↪ current phase is WAITING
Sigma = 0
}

δint (e)
{
Clear Processed Loads queue

case phase:
       PROCESSING:
              For each waiting load sort the pallets
                  ↪ into Waiting Pallets and
                  ↪ Destination Pallets queue based
                  ↪  on whether this location is
                  ↪ the final location of the
                  ↪ Pallet
                     If the pallet is added to the
                         ↪ Waiting Pallets queue,
                         ↪ set the next location
                         ↪ for the pallet.
                     If the next location for the
                         ↪ pallet is -1 add to the
                         ↪ Destination Pallet queue
                         ↪  instead.
              Clear Waiting Loads
              Sort Waiting Pallets by priority
              Determine if there are enough pallets
                  ↪ to make a load and if there is
                  ↪ an available aircraft. There
                  ↪ must be enough pallets to fill
                  ↪ at least 50% of the aircraft.
              If there are enough Pallets and a
                  ↪ Destination aircraft, generate
                  ↪ Load and add it to the
                  ↪ Processed Loads queue
              Check to see if any aircraft are
                  ↪ expired (i.e. been waiting 24
                  ↪ hours without a load request).
                  ↪ If so return the aircraft to
                  ↪ its home base.
              If there are Processed Loads
                     Phase = SENDLOAD
                     Sigma = 10
              Else If there are Destination Pallets
                     Phase = DELIVER
```

```
                    Sigma = Delivery Time (i.e. 30
                        ↪ min)
                Else
                    Phase = WAITING
                    Sigma = ∞
        SENDLOAD:
                Phase = PROCESSING
                Sigma = 10
        DELIVER:
                Clear Destination Pallets queue
                Phase = PROCESSING
                Sigma = 10
}

λ(s)
{
If phase = DELIVER
        Output all pallets in the Destination Pallets
            ↪ queue
If phase = SENDLOAD
        Output all loads in the Processed Loads queue
}
```

3.5 Aircraft Atomic Model

Each aircraft starts the simulation at a home base and will transport loads of pallets to other locations as directed by each location. If there is no requirement to transport loads of pallets at a given location the aircraft will return to its home base after a defined period of time (default 1 day). Once the aircraft receives a load message from a location it will add the load to the aircraft and begin to fly. Every period (default 30 minutes) the aircraft will announce its position along with a status message until it reaches its destination. At the destination the aircraft will announce its arrival using an Aircraft Status message and unload the load to the location.4

There is an initialization phase before aircraft are ready to receive loads. This phase is used to send an Aircraft Status message to all locations to announce where the aircraft is currently located. At the start of the simulation an aircraft is located at its home base as specified during initialization of the model. Further information about Aircraft Status messages are available in section 3.6. Once the aircraft is initialized it waits to receive a Load message from a location as shown in figure 4.

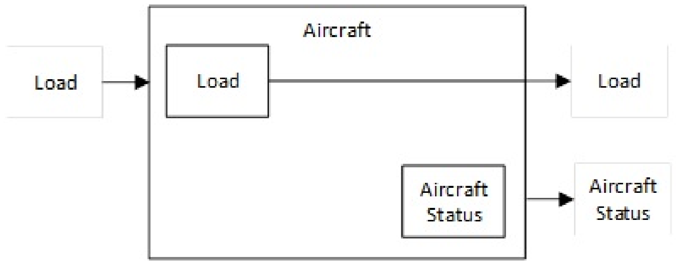


Figure 4: Aircraft Atomic Model

Once a Load message is received the aircraft stores the load and begins to fly. The aircraft will broadcast its position every 30 minutes similar to the ADSB functionality on most modern aircraft. This information is used for visualization. Once the aircraft reaches the destination location it will announce its arrival via an Aircraft Status message and send a Load message to the destination location.

Once at the destination location, the aircraft will wait 24 hours to receive another load. If no Load message is receive in that period, the aircraft will return to its home base. If the aircraft is already at its home base, the aircraft will continue to wait until there is a Load message for it.

3.5.1 *Formal Definition.* **Aircraft = <X, Y, S, ta, δext, δint, λ>**

State Variables

```
Phase = INIT
Sigma = 0
Aircraft ID = Passed to constructor
Type = Passed to constructor
Average Speed = Passed to constructor
Home Location = Passed to constructor
Max Load Size = Passed to constructor
Load = {}
Current Latitude = 0
Current Longitude = 0
Distance Remaining = 0
Total Distance = 0
```

Formal Specifications

```
X = { Load ∈ <String, N, R, R, R, R, N, vector<Pallet
    ↪ >>}
Y = { Aircraft Status ∈ <N, N, N, N, N, Time> | Load
    ↪ ∈ <String, N, R, R, R, R, N, vector<Pallet>>}
S = {{Phase, Sigma, Aircraft ID, Type, Average Speed,
    ↪  Home Location, Max Load Size, Load, Current
    ↪ Latitude, Current Longitude, Distance
    ↪ Remaining, Total Distance}}
```

Pseudo Code

```
$\delta$ext (mbs, e, x <Load>)
{
If x is Load and the load is for this aircraft
        Add x to aircraft Load
        Calculate flying route and set associated
            ↪ state

If phase = WAITING and Distance Remaining > 0
        Phase = FLYING
        Sigma = 0
Else If phase = FLYING
        Sigma = Update time period with e
Else If phase = WAITING
        Sigma = $\infty$
}

$\delta$int (e)
{
```

```
Case
        INIT:
                Phase = WAITING
                Sigma = $\infty$
        FLYING:
                If Distance Remaining < 0
                        Phase = UNLOADING
                        Sigma = Unload Time (i.e. 30
                            ↪ min)
                Else
                        Update current position
                        Sigma = Update Time (i.e. 30
                            ↪ min)
        UNLOADING:
                Phase = WAITING
                Sigma = $\infty$
}

$\lambda$(s)
{
If Phase = UNLOADING
        Send Aircraft Status message to destination
            ↪ location
        Send Load message to the destination location
If Phase = INIT
        Send Aircraft Status message to home location
}
```

3.6 Message Types

The strategic airlift model relies on four different messages to pass information between atomic models and control the simulation flow. This next section provides a brief outline of the message types. A standard strategic airlift model requires two input readers, one for Location Information messages to initialize location connections, and one for pallet messages to control the simulation flow.

3.6.1 *Pallet Message.*

Pallets are the main item which is monitored or counted during the simulation. Below is an example of the pallet message input format:

```
1/1 2 1 3
```

There are 4 fields per pallet message which are separated by a space.

(1) Time - Time using the format specified in the configuration XML;
(2) Pallet Id - The unique Pallet Id. This is also used to denote priority of pallet;
(3) Next Location Id - Id of the next location. This is used by locations to determine if a pallet should be added to their waiting pallet queue; and
(4) Destination Location Id - Id of the destination location.

Operators can define one pallet per line and have the ability to define an unlimited number of pallets.

3.6.2 *Load Message.*

Load messages contain the following information:

(1) Load Id - Identification of the load;
(2) Destination Id - Id of the destination location;
(3) Source Latitude - Latitude of the source location;
(4) Source Longitude - Longitude of the source location;
(5) Destination Latitude - Latitude of the destination location;
(6) Destination Longitude - Longitude of the destination location; and
(7) Aircraft Id - Id of the aircraft the load is destine for.

3.6.3 *Aircraft Status Message.*

Aircraft Status messages contain the following information:

(1) Aircraft Id - Id of the aircraft sending the message;
(2) Location Id - Id of the location where the message is destine for;
(3) Capacity - Number of pallets the aircraft can carry;
(4) Type - Type of aircraft;
(5) Home - Home location id for the aircraft; and
(6) Waiting Time - How long the aircraft has been waiting at a location.

3.6.4 *Location Information Messages.*

Location information messages provide locations information about routes that exists between two locations. Below is an example route input format:

```
0/1 1 2 2 82.501667 -62.348056 2
```

There are 7 fields per route message which are separated by a space.

(1) Time - Time using the format specified in the configuration XML;
(2) Source Location Id - Id of the source location. This is used by locations to determine if they should add this route to their connections;
(3) Destination Location Id - Id of the destination location.
(4) Next Location Id - Id of the next location to send pallets to. This will pass the pallet along until it arrives at the destination or there is no further routing available;
(5) Next Location Latitude - The decimal latitude of the next location;
(6) Next Location Longitude - The decimal longitude of the next location; and
(7) Aircraft Type - The type of aircraft that can fly this route segment.

Operators can define one route per line and if they want a bi-directional flow between locations, two lines will need to be created to represent a segment in each direction.

3.7 Coupled Model

The strategic airlift coupled model was intended to be implemented as two sub-coupled models. This design choice was made to reduce complexity with understanding and implementing the model. As shown in figure 2 the operator has the ability to define an unlimited number of locations and aircraft. The coupling is simplified by connecting the input and output ports of each location or aircraft to the associated coupled model input and output ports. Location and aircraft ids are used to filter messages for destination locations and aircraft. This is similar to the real world where information would be broadcast via a radio or scheduling system and only the locations or aircraft that are effected by the messages respond.

An additional benefit to this approach is re-using a group of locations or aircraft between experiments. For example the operator could envision a scenario where they want to keep the locations the same, but vary the number or type of aircraft to see its effect.

Running the coupled model requires 2 inputs. One input will provide pallets to the model that will need to be moved between locations until they reach their destination. The other input is required to provide routing for each location.

4 FEATURE IMPLEMENTATION

Operationalizing the constructive simulation of the strategic airlift model requires the development of a number of key components. These components include:

- An input mechanism to define the experiment;
- A conversion mechanism to take the input and translate it to a language the simulator understands;
- A simulator to run the experiment; and
- A way to visualize the experiment.

The primary design criteria for these components is enabling constructive simulation of the strategic airlift model by developing a system which is easy for the operator to understand and use. Multiple designs were considered for each component and continually iterated until they met the following conditions:

- Usability - The system must be easy to understand and operate;
- Flexible - Each component needs to accommodate a wide variety of use cases which an operator would use the system for;
- Agile - The system must quickly adapt to changes, for example a new visualization is required and needs to be developed and incorporated into the system;
- Extensible - The system must allow for growth and the ability to add new feature; and
- Technology agnostic - The system must be flexible to not force the operator to use a specific technology or implementation.

Specific components that were developed to demonstrate a simplified operator experience include:

- XML DEVS Definition;
- Generic DEVS JSON logging format; and
- Flexible and agile visualization service.

4.1 XML DEVS Definition

XML was chosen over JSON to generically represent a DEVS coupled model due to it rigid formal specification. DEVS itself is a defined formalism for describing models and implementing JSON would allow the operator too much flexibility that may lead them to making errors. Two XML files were developed to define DEVS models in XML. One defines a coupled model and the other defines a simulator configuration. This provides the operator the flexibility to describe any DEVS model using XML and easily swap out the configuration XML to use it with a different simulator. Having one coupled model per XML also allows the flexibility for different coupled models to be swapped out similar to atomic models. In the Arctic example below this will be demonstrated by having many of the same XMLs between experiments for configuration, defining locations, and defining aircraft.

Each XML below is validated against an XSD in Sim Manager and the XML tags are always listed in order to comply with the XSD. Information between the tags is taken directly from the XML and merged into a text template. If the operator is specifying strings as values they need to ensure & quot; are used at the beginning and end of values.

4.1.1 *Configuration XML.*

The configuration XML is specific for each simulator and the one developed for this paper uses the Cadmium simulator which is used at Carleton University. This XML format can be used as a template for anyone wishing to translate it to another simulation environment.

Below is a partial representative example of a configuration XML for illustrative purposes.

```
<?xml version="1.0" encoding="UTF-8"?>
<configModel name="Cadmium">
        <headers>
                <header>&quot;../../messages/
                    ↪ StratAirLiftSimMessage.hpp&quot
                    ↪ ;</header>
        </headers>
        <objects>
                <object>StratAirLiftSimMessage</object>
        </objects>
        <time>
                <timetype>EIRational</timetype>
                <rununtil>-1</rununtil>
        </time>
        <loggers>
                <loggerfile name="stratairliftstates"
                    ↪ />
                <logger name="state" type="logger::
                    ↪ logger_state" format="dynamic::
                    ↪ logger::formatter&lt;TIME&gt;"
                    ↪ loggerfile="stratairliftstates"
                    ↪  />
        </loggers>
</configModel>
```

All tags are required except for <headers> and <objects>. The XML is defined by the <configModel> tag which takes an attribute called name. This attribute is used to denote which simulator the config XML is for.

Below is a list of tags and their description for the config XML:

- <headers> - Each configuration can include one or more header files.
- <header> - This tag is used to specify any headers required for the model to run. Typically, this includes the message header file and other speciality C++ packages used to make the model function.
- <objects> - Each configuration can include one or more object files that are used to pass pre-compiled objects to the simulator. This option reduces the compile time required for

experiments that re-use common libraries such as messages or common functions.

- <object> - A list of object files for the linker to use when compiling the executable. The .o suffix will automatically be added to the name when merged in the text template.
- <time> - Only one time tag is allowed per configuration to define time parameters.
- <timetype> - The name of the time library to use.
- <rununtil> - Defines how long the simulation should run. -1 is used to instruct the simulator to run the simulation until everything is passive.
- <loggers> - Allows the definition of any number of loggers. All <loggerfile> tags should be defined first followed by all <logger> tags. Each simulation requires at least one logger.
- <loggerfile> - Defines an output log file which can be linked to multiple output loggers.
- <logger> - Defines the loggers for the simulator. Currently, only the state and message logger in Cadmium support JSON output with a minor amendment to Cadmium.

4.1.2 Coupled Model XML. Below is a partial representative example of a coupled model XML for illustrative purposes. The example used is based on an Aircraft coupled model.

```
<?xml version="1.0" encoding="UTF-8"?>
<coupledModel name="Aircraft">
       <ports>
              <port type="in" name="AC_inLoads"
                  ↪ message_type="oLoad"></port>
       </ports>
       <components>
              <submodel type="atomic" name="Aircraft1
                  ↪ " class_name="Aircraft"
                  ↪ xml_implementation="Aircraft.
                  ↪ devs">
                     <param type="int" name="
                         ↪ iAircraftID" value="1" /
                         ↪ >
              </submodel>
       </components>
       <connections>
              <eic in_port_coupled="AC_inLoads"
                  ↪ submodel="Aircraft1"
                  ↪ in_port_submodel="
                  ↪ Aircraft_defs::inLoads" />
              <eoc submodel="Aircraft1"
                  ↪ out_port_submodel="
                  ↪ Aircraft_defs::outLoads"
                  ↪ out_port_coupled="AC_outLoads"
                  ↪ />
       </connections>
</coupledModel>
```

The format presented can be used to represent any DEVS coupled model. The XML is split into 3 different sections, ports, components, and connections. The <coupledModel> tag defines this XML as representing a coupled model. The tag has a single attribute name to specify the name of the coupled model.

- <ports> - A coupled model can contain 1 or more ports.
- <port> - Used to specify input and output ports for the coupled model. Attribute information is used to describe the details of each port. Type can either be ’in’ or ’out’. Name is used in connections to to describe the port. Message_type is used to specify the type of message. Custom message types need to be pre-compiled and included in the <object> tag of the configuration XML.
- <components> - A coupled model can contain 1 or more components.
- <submodel> - Each <submodel> tag includes a number of attributes to define the sub models. The type can either be ’atomic’ or ’coupled’. The name is used by connections and the class_name describes the class of sub model it is. Xml_implementation refers to where the atomic or coupled model is defined. In this iteration of Sim Manager this is only used for coupled models and input stream readers. For atomic models this parameter is ignore, however is still required for compliance. For input readers the xml_implementation is defined as iestream.
- <param> - Each sub-model can include 0 or more parameters which will be passed during atomic model instantiation. Attributes for the parameter includes a type, name, and value. Parameters must be provided in the order they will be sent to the constructor. Don’t forget to include the & quot; if the value is a string.
- <connections> - Connections are used to define the external input connections, external output connections, and internal connections. They must appear in that order to be compliant with the XSD.
- <eic> - External input connections include the attributes: in_port_coupled, submodel, and in_port_submodel which refers to the name of each port or sub model.
- <eoc> - External output connections include the attributes: submodel, out_port_submodel, and out_port_coupled which refers to the name of each port or sub model.
- <ic> - Internal connections include the attributes: from_submodel, out_port_from, to_submodel, and in_port_to which refers to the name of each port or sub model.

4.2 JSON Log

The current log output format from Cadmium was restrictive and required amendments to allow greater flexibility to generate visualizations to communicate results from the simulator. An improved generic JSON log format was developed for Cadmium and any model can implement it. The two main loggers for states and messages were modified to accommodate the new JSON format. However, the message logger format was deeply embedded within Cadmium and this paper was not able to fully convert it to a true JSON format. This is captured as part of future work in section 7.

The JSON logging format consists of the following properties:

(1) time - Denoting the current time for the log entry;
(2) type - Denoting the type of log entry. Currently, this is state or message. Additional types can be added in the future.;
(3) name - Name of the model that produced the log entry; and

(4) data - Any number of properties which the model wants to include in the log.

Below are examples of properties from the strategic airlift model included under the data property in the JSON logging format:

- class - Type of model object. Useful to denote the difference between locations and aircraft;
- location - This property has sub properties of lat and long. This is used to generate maps in visualizations; and
- state - The state of a given model.

The benefits of this JSON format is there are minimal required fields and the format remains flexible to any fields the model wants to include. Various visualizations and other tools can take advantage of this as required. Below is an example JSON log entry from the strategic airlift model.

```
{
  time: "90/1",
  type: "state",
  name: "Trenton",
  data: {
    state: "Processing",
    class: "Location",
    message: "Location Trenton has 2 aircraft waiting
        ↪  and 1 pallets waiting.",
    text: "Trenton",
    pallets: 1,
    location: {
      lat: 44.1,
      long: -77.6
    },
    debug: "2:1:0"
  }
}
```

4.3 Visualization

Two visualization components were implemented to show the agility of the JSON log format. The map component was also produced as an example of visualizing geographic data. Each component is known as a panel and each panel can take up any dimension of space as defined in their associated CSS class. The map panel has been configured to span the whole screen and only be 400px tall. While the text panel has been configured to fill half the screen and the entire height of displayed text.

4.3.1 Map Panel. The map panel displays any log entry with a data property of location. The map uses the class property of data to select the appropriate icon to show the operator. If the operator wanted to extend this to other applications they would simply add the icon for the associated class to the img folder of Sim Visualizer. For example a ship could easily be added by having the model output a log entry with a class of ship, and an associated location property. If the operator includes an image of a ship in the img folder it will be displayed automatically. A map panel example is shown in figure 7.

4.3.2 Text Panel. The text panel is used to showcase how the operator can re-use the same component and use filtering to select which set of log data they want to display. When the text component is created the operator can define the filter type to only include the state, message, or both logs. They can also apply a display parameter that will select one of the data properties to display. In the text panel example the left hand side of the screen shows the state information from location and the right shows the message output for each object. This is shown in figure 7.

Below is an example of how to instantiate a Text Panel:

```
myLogMessageOutput=TextPanel().id("MessageOutput").
    ↪ canvas("#PANELS").filterType("message").
    ↪ filterProperty("Debug").newPanel();
```

Calling the function filterType allows the operator to choose either "message" or "state" logs. The filterProperty function allows the operator to choose which data property they want to display.

5 TECHNOLOGY

The technology used to implement the model and life cycle was deliberately varied to demonstrate how anyone could add a component using any language or framework. The specific technology stack included both frameworks and different languages. The following frameworks were used in the development of the modeling and simulation solution: Spring Boot, D3, and jQuery. These were implemented through the use of Java, PHP, JavaScript, and C++. The system was specifically designed to be flexible to allow for growth using any language or technology stack. These technologies are tied together through containerized microservices using REST APIs. Below are the core components which enable the modularization of the strategic airlift model.

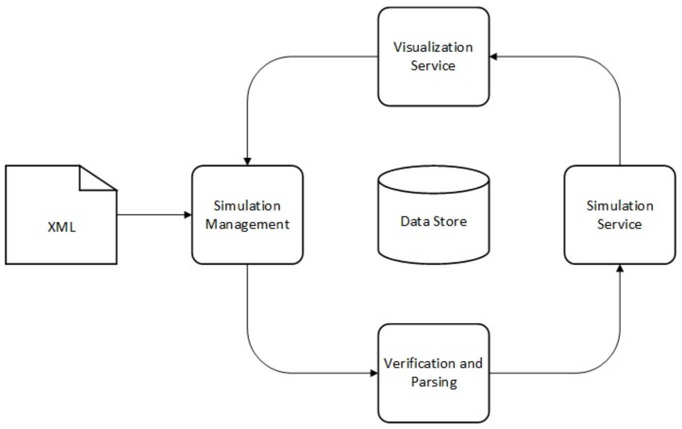


Figure 5: Simulation Components

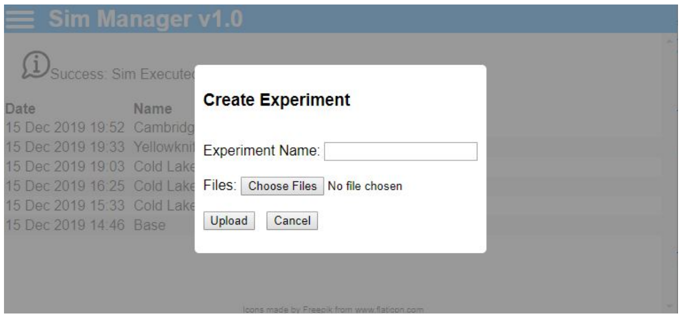
(1) REST API - REST APIs are a great fit for DEVS since they are stateless and complement the DEVS formalism. Everything that is required to complete the transaction is included in the web service call. Additionally, any data type can be sent and returned by REST APIs. JSON was specifically chosen as the data representation format since it is well understood by JavaScript and most modern web browsers. This makes implementation using frameworks such as D3 and jQuery seamless.

(2) Containers - Containers gained wide scale adoption when Docker introduced their implementation in the mid-2010's. This introduction was the catalyst to moving towards microservices and edge computing. These concepts break down historically large monolithic software programs into smaller atomic pieces and delivered as close to the user as possible. This paradigm is enabled by containers which are light weight virtualized constructs that are well suited for single purpose applications. Containers typically only contain the bare essentials for a program to run.
(3) Microservices - Microservices are used to break down each individual component of a software program into atomic pieces. These de-coupled pieces can operate independently allowing faster compilation and easier maintenance.
(4) Visualization - Visualizations are a key feature required to communicate large amounts of data easily to a human observer, to extract key pieces of knowledge.

5.0.1 Overview. As shown in figure 5 the solution has 5 key components and the process starts by uploading a set of XML and data files. This process is iterative and the user can execute and visualize the simulation an unlimited number of times.

- Simulation Management - Management is required to upload, store, and manage the various models and experiments for the operator.
- Verification and Parsing - Verification of the input data and parsing is required to convert the input into a neutral format.
- Simulation Service - The neutral model format is transformed into simulator code and the experiment is executed.
- Visualization Service - The visualization service provides the results in an easy to digest format to the operator.
- Data Store - The entire process is supported by a data store for both files and information in a database.

The specific implementation in this paper is referred to as Sim Manager and has an easy to use web interface shown in figure 6. Using this format allows the operator to extend or replace any component easily. For example if the operator prefers to use CD++ instead of Cadmium as their simulator, they can simply switch the simulation microservice. Additionally, the operator has the flexibility to switch the input format to something like JSON or add additional visualizations.


Sim Manager v1.0

Success: Sim Execute

Date	Name
15 Dec 2019 19:52	Cambridg
15 Dec 2019 19:33	Yellowkni
15 Dec 2019 19:03	Cold Lake
15 Dec 2019 16:25	Cold Lake
15 Dec 2019 15:33	Cold Lake
15 Dec 2019 14:46	Base

Create Experiment

Experiment Name:

Files: Choose Files No file chosen

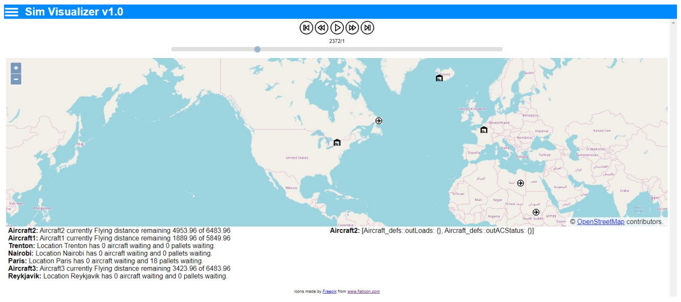
Figure 6: Sim Manager

5.0.2 Microservices. Each component was envisioned as a microservice which could be implemented as containers. Sim Manger is implemented in 6 docker contains which are explained below:

- MySQL Database - The database stores information about each experiment and the status;
- Sim Manager - Is a full stack application that co-ordinates all the different services and provides the operator an easy to use graphical user interface to run their experiments;
- Sim Visualizer - Is a full stack application supported by Open Layers and D3 to visualize results of experiments;
- Manage Experiment - Is a J2EE Spring Boot service which is responsible to store experiment files, delete experiments when the operator is done with them, and list experiments.
- Run Experiment - Is a J2EE Spring Boot service which is responsible to parse a given set of experiment configuration files and run them using the simulator specified in the config file. Currently, this service supports parsing XML and running the Cadmium simulator; and
- View Experiment - Is a J2EE Spring Boot service which is responsible for collating log files for direct download to the operator and parsing log files to transform them into JSON extract which are used by Sim Visualizer.

5.1 Visualization

The visualization component was developed with flexibility in mind. The full stack application starts by making a single Ajax call to retrieve log data from the visualization microservice. The log data comes pre-formatted from the visualization microservice in JSON split by time period. The operator controls playback using control buttons and a slider at the top of the page to control playback of the simulation results as shown in 7. Every time the user selects a new time period, the data for that specific time period is broadcast to all subscribing panels on the page. Each visualization panel subscribes to data updates when it is instantiated and therefore each panel can be developed and implemented independently.



Aircraft2: Aircraft2 currently Flying distance remaining 4953.96 of 6483.96
Aircraft1: Aircraft1 currently Flying distance remaining 1889.96 of 5849.96
Trenton: Location Trenton has 0 aircraft waiting and 0 pallets waiting.
Nairobi: Location Nairobi has 0 aircraft waiting and 0 pallets waiting.
Paris: Location Paris has 0 aircraft waiting and 18 pallets waiting.
Aircraft3: Aircraft3 currently Flying distance remaining 3423.96 of 6483.96
Reykjavik: Location Reykjavik has 0 aircraft waiting and 0 pallets waiting.

Aircraft2: [Aircraft_defs::outLoads: {}, Aircraft_defs::outACStatus: {}]

Figure 7: Sim Visualizer

5.1.1 Map Panel. There are many mapping technologies available, for example Google Maps, MapBox, and Open Layers. Open Layers was chosen due to its flexible license agreement and developed API.

5.1.2 Text Panel. In addition to the map visualization, a text visualization was developed to demonstrate the flexibility of the publish / subscribe data paradigm. This allows an unlimited number of unique visualizations to be included as options for viewing output from a simulator. The two text panels include the ability to filter on state versus message logs and to display any value from the JSON log data structure.

6 EXPERIMENTS

6.1 Arctic Problem Description

The Arctic logistical support base problem was described in detail in section 2.2. This problem is explored using the strategic airlift model as described in section 3. The location and aircraft atomic model were implemented using the DEVS formalism in Cadmium based on the pseudo code in the model description section. This next section will focus on the set of experiments which were run to explore the Arctic logistcal support base problem.

All experiments were defined in a set of XMLs as described in section 4.1. These XML files were executed using the Sim Manager life cycle and results were explored using Sim Visualizer.

The brief description of the problem that is being explored through these experiments is: "What is the optimal location for a single intermediate logistics base in the Canadian Arctic?". This question is important since the Globemaster aircraft isn't able to operate from all the same airfields that a Hercules aircraft can operate from.

6.2 Scenario Overview

Originally, four experiments were planned to determine the differences between no logistical support base and one logistic support base at Cold Lake, Alberta; Yellowknife, North West Territories; or Iqaluit, Nunavut. However, after review of the Cold Lake experiment results a different path was explored and resulted in six experiments being conducted.

All scenarios have the follow set of parameters in common:

- All pallets will originate from Trenton, Ontario;
- Pallets will be flown by Globemaster aircraft to an intermediate logistical support base location and then to the final destination by a Hercules aircraft.
- The set of pallets will be the same for all scenarios. The pallets will be evenly distributed across each of the four final destinations, with each location having 48 pallets. The pallets are arranged in the same random order for each scenario.
- The four final destinations are: Alert, Nunavut; Hall Beach, Nunavut; Cambridge Bay, Nunavut; and Inuvik. This represents the geographic diversity of Canada's north and reduces bias for any single intermediate logistical support base location.
- Each Globemaster is configured to hold 16 pallets and fly at 490 knots. Each Hercules is configured to hold 4 pallets and fly at 300 knots.

An overview will be provided for each experiment in the next sections. After the overview is a sub section to discuss the results of each experiment.

6.3 Base Experiment

6.3.1 Overview. The goal of this experiment is setting a base line value for how long it would take three Hercules aircraft to resupply all four final destinations listed in the scenario overview. The scenario concludes when all aircraft return to their home base. The baseline parameters in the scenario overview are used for this experiment.

6.3.2 Results. The base line scenario requires 25 days, 14 hours, and 24 minutes (36864 minutes) to transport all the pallets from Trenton to the four final locations using three Hercules aircraft. Table 1 contains the distances between Trenton and the four final destinations. Pallets could be delivered to locations faster using either an intermediate logistical support base, or adding Globemaster aircraft instead of Hercules aircraft due to their larger carrying capacity. Figure 8 shows the pallets are evenly distributed to all locations and the model is appropriately prioritizing pallet transport.

Community	Distance (km)
Alert	4299
Hall Beach	2747
Cambridge Bay	3187
Inuvik	4156

Table 1: Distance from Trenton, Ontario

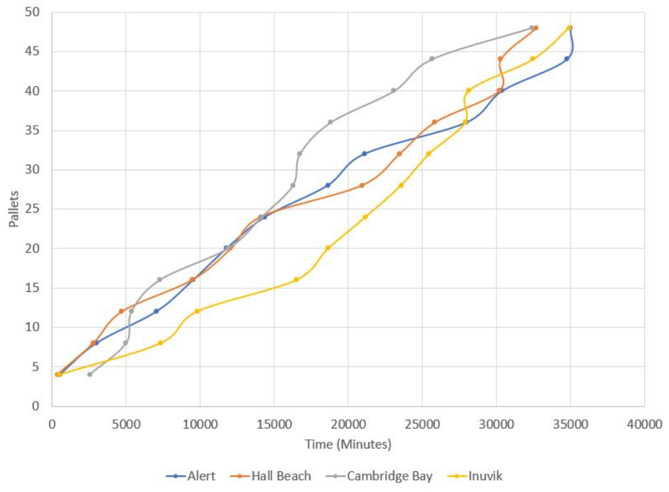


Figure 8: Delivery Time vs Pallets

6.4 Cold Lake Experiment - Version 1

6.4.1 Overview. Using the base experiment as a comparison, this experiment explores the effects of having an intermediate logistical support base in Cold Lake, Alberta. The hypothesis is this would reduce the time required to transport all pallets to their final destinations as specified in the scenario overview section.

One Globemaster will be used to transport pallets from Trenton to Cold Lake and three Hercules will be used to transport the pallets from Cold Lake to the four final destinations specified in the scenario overview. The baseline parameters in the scenario overview are used for this experiment.

6.4.2 Results. The Cold Lake version 1 scenario requires 23 days, 14 hours, and 11 minutes (33971 minutes) to transport all the pallets from Trenton to the four final locations using one Globemaster aircraft from Trenton to Cold Lake and three Hercules aircraft to the four final destinations. This could lead the operator to conclude the addition of an intermediate logistical base in Cold Lake does not have a large effect on the scenario time. With the effort required to set up an intermediate logistical base, it may not be worth the return on investment.

The Globemaster aircraft finished transporting pallets from Trenton at 14 days, 20 hours, and 38 minutes. This not only represents a savings in time, but also a savings on distance flown by aircraft which translates to a reduction in fuel and maintenance. Table 2 shows the distances between locations. This information is used to calculate the distance flown by each aircraft during this experiment. Table 3 shows the distance flown by each type of aircraft, compared between the base experiment and this experiment. The difference is 46 206 km.

Location	Trenton (km)	Cold Lake (km)
Alert	4299	3438
Hall Beach	2747	2166
Cambridge Bay	3187	1649
Inuvik	4156	1962
Trenton	0	2599

Table 2: Distance Between Locations

Aircraft	Base (km)	Cold Lake v1 (km)
Globemaster	0	62376
Hercules	345336	236754
Total	345336	299130

Table 3: Number of Flights

Further experimentation could be conducted to optimize the location that would result in the fewest miles flown, however other options exist to reduce the scenario completion time. The next experiment will look at the effects of adding more aircraft to the scenario. This hypothesis was developed after reviewing the logs to determine how much time aircraft spent waiting for pallets as shown in figure 9. This figure shows that almost half the time there was at least 1 aircraft waiting for pallets in Cold Lake. The waiting time issue could increase if aircraft immediately flew back after delivering their load to the final destination. Currently, all the Hercules aircraft in the experiment wait for 24 hours at the final destination prior to returning to their home base.

6.5 Cold Lake Experiment - Version 2

6.5.1 Overview. The previous experiment demonstrated the scenario time could be reduced by adding an intermediate logistical

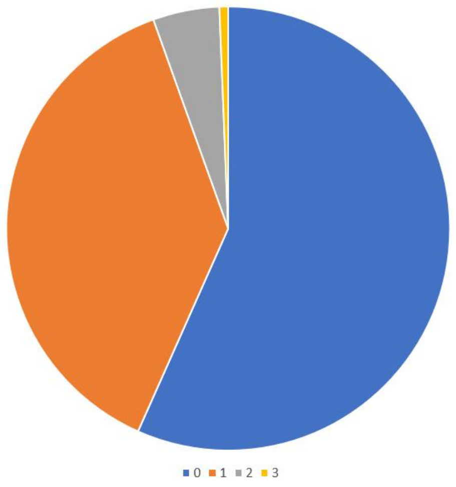


Figure 9: Number of Waiting Aircraft by Aggregate Time

support base, however there was still a bottle neck with transporting pallets from Trenton to Cold Lake. This resulted in a number of Hercules aircraft waiting for pallets. This experiment will look at the effects of adding a second Globemaster to transport pallets from Trenton to Cold Lake. All other experiment parameters will remain the same as version 1 of the experiment.

6.5.2 Results. The Cold Lake version 2 scenario requires 22 days, 14 hours, and 3 minutes (32 523 minutes) to transport all the pallets from Trenton to the four final locations using two Globemaster aircraft from Trenton to Cold Lake and three Hercules aircraft to the four final destinations. Compared to version 1 of the scenario this resulted in a limited reduction in the scenario time. However, as shown in figure 10 a decrease in waiting time was observed which leads to a decrease in scenario time since aircraft aren't waiting for pallets.

This leads to the conclusion that supply of pallets has a large effect on overall scenario completion time. Therefore, if the goal is to minimize scenario completion time, more aircraft are required to transport pallets to an intermediate logistical support base.

Additional observations include the total time required to transport pallets from Trenton to Cold Lake in this experiment was 6 days, 20 hours, and 5 minutes. Compared with 14 days, 20 hours, and 38 minutes from version 1 of the experiment, this represents over a 50 percent reduction in time. One solution to reducing scenario time would be the aggressive dedication of resources early on to transport goods to the intermediate logistical support location.

6.6 Cold Lake Experiment - Version 3

6.6.1 Overview. The previous experiments demonstrated a bottle neck with not providing a sufficient set of pallets to Cold Lake for distribution to the final destinations. This experiment will continue by exploring what will happen if a Globemaster was position with two Hercules aircraft in Cold Lake instead of the three Hercules. This should hopefully result in quicker pallet distribution to the final destinations.

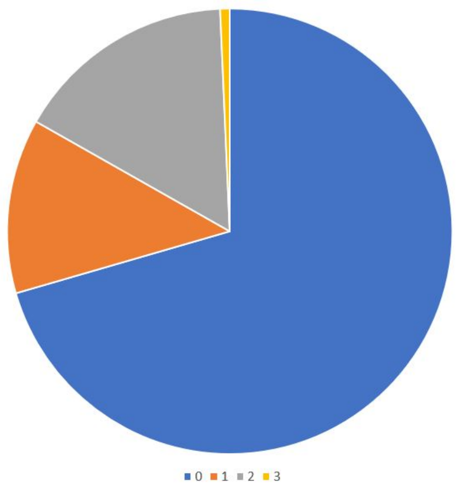


Figure 10: Number of Waiting Aircraft by Aggregate Time

The only change for this experiment from version 2 will be the swap of one Hercules aircraft for one Globemaster in Cold Lake.

6.6.2 Results. The Cold Lake version 3 scenario requires 14 days, 2 hours, and 16 minutes (20 296 minutes) to transport all the pallets from Trenton to the four final locations using two Globemaster aircraft from Trenton to Cold Lake and one Globemaster and two Hercules aircraft to the four final destinations. This is a significant reduction in the over all scenario time compared to the base experiment.

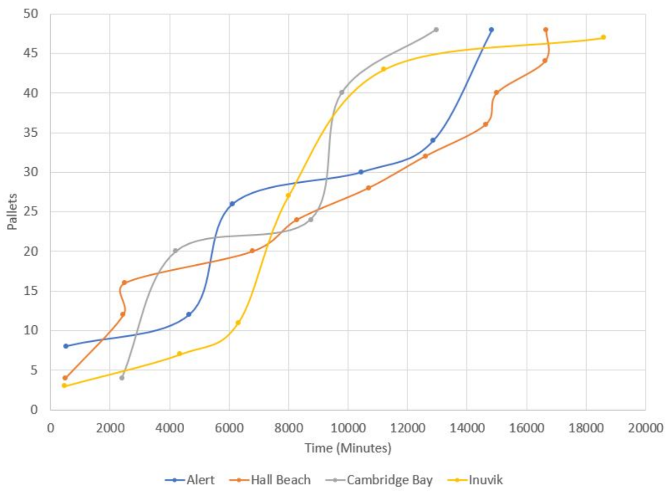


Figure 11: Delivery Time vs Pallets

Figure 11 shows a graph of pallets delivered by time for each of the final destinations. As shown in the graph the use of the Globemaster was relatively equal between all the locations and allowed each location to reduce the number of flights required from twelve to five in some instances. Figure 12 shows the number of aircraft required to transport all goods to each location. One Globemaster significantly reduces the requirement for multiple Hercules aircraft.

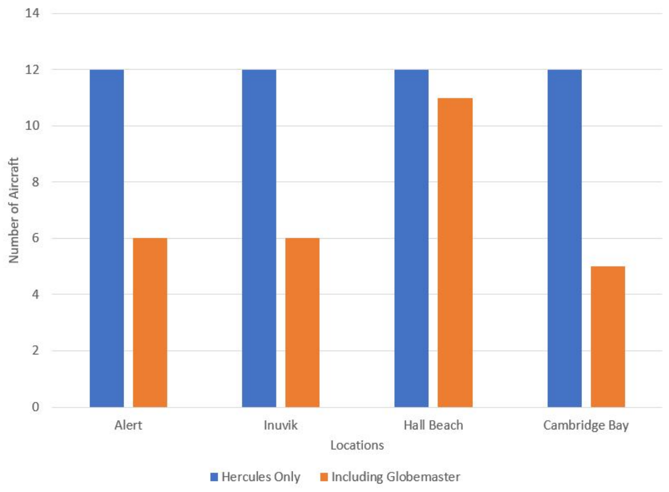


Figure 12: Location vs Number of Aircraft

6.7 Yellowknife

6.7.1 Overview. Version 1 of the Cold Lake experiment demonstrated reduction in flying time by using an intermediate logistical support base. This next experiment is a continuation where Yellowknife is used instead of Cold Lake. This location is further from Trenton, however it moves closer to the center of the four final destinations. This experiment will use all the same parameters as Cold Lake version 3, except switching the location from Cold Lake to Yellowknife.

6.7.2 Results. The Yellowknife scenario requires 12 days, 16 hours, and 56 minutes (18 296 minutes) to transport all the pallets from Trenton to the four final locations using two Globemaster aircraft from Trenton to Yellowknife and one Globemaster and two Hercules aircraft to the four final destinations. This represents a further reduction in scenario time from the other scenarios.

6.8 Cambridge Bay

6.8.1 Overview. The final experiment this paper will conduct is a follow on to the previous experiment where Cambridge Bay is used as the intermediate logistical support base and Yellowknife becomes one of the final destinations.

6.8.2 Results. The Cambridge Bay scenario requires 12 days, 14 hours, and 50 minutes (18 170 minutes) to transport all the pallets from Trenton to the four final locations using two Globemaster aircraft from Trenton to Cambridge Bay and one Globemaster and two Hercules aircraft to the four final destinations. This represents the lowest scenario time for all the experiments.

As shown in table 4 the total distance flown and number of flights for this location is significantly less than previous experiments. However, with the distance from Trenton and addition of the Globemaster to deliver to the final destination it has increased

Location	Number of Trips	Total Distance (km)
Alert	6	21540
Hall Beach	8	15136
Yellowknife	7	11928
Inuvik	7	16058
Trenton	15	95610
Total	43	160272

Table 4: Cambridge Bay Statistics

the percent of time aircraft spend waiting as shown in figure 13. A review of the logs indicate all pallets were in Cambridge Bay at 7 days, 3 hours, 4 minutes.

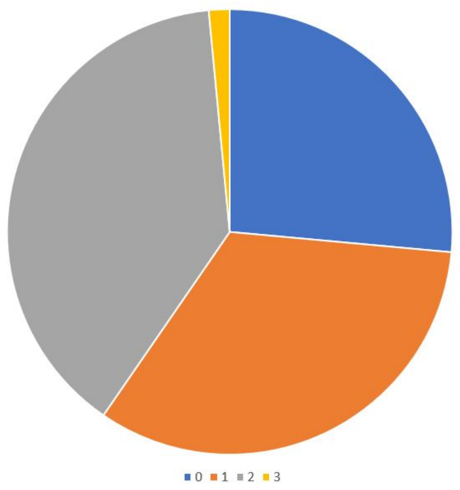


Figure 13: Number of Waiting Aircraft by Aggregate Time

Based on the results of this experiment, Cambridge Bay would be a good choice for an intermediate logistical support base due to its central location in the Arctic. Additional experimentation could be conducted to determine the correct mix of aircraft for re-supply to minimize the time spent waiting on the ground.

6.9 Model Conclusions

Results from experimentation indicate a number of key areas which could be optimized for when selecting an intermediate logistical support base. These include the following:

(1) A location should be chosen that is geographically center between the final destinations, but not too far from the source location;
(2) This model could be turned into an optimization problem where the objective would be to fly the minimum distance to deliver all the pallets. Minimum distance would translate into a reduction in fuel, maintenance, and number of aircraft and result in the lowest cost option.
(3) Having a robust link to the source location is critical to move pallets from the source to intermediate location as quick as possible. This will minimize the time other aircraft spend waiting for pallets to transport.

The optimal solution for this specific instance of the problem is to put the intermediate logistical supply base in Cambridge Bay, Nunavut and use two Hercules and one Globemaster for resupply from Cambridge Bay to the final destination. At least two Globemaster aircraft would be required to maintain the flow of pallets from Trenton to Cambridge Bay.

A future modification to this model could include having aircraft transition between different routes. In this instance it would allow the scenario to start with two Globemasters aircraft from Trenton to quickly supply Cambridge Bay and then have one Globemaster be re-tasked to a different mission and one Globemaster continue with two Hercules to supply the final destinations.

7 FUTURE WORK

A number of enhancements to the current model and system were considered during the development of Sim Manager and drafting of this paper. Below is a list of future work items.

- Logging - The visualizations perform poorly in Sim Visualizer when there is a large volume of logs. This can be improved by further exploration and definition of the JSON logging format to ensure it is flexible enough to cover all use cases. After the final format is developed, simulators such as Cadmium will require modifications to ensure they comply with the new format. Lastly, the logging process will need to be extended to do pre-processing of the log files. This can be accomplished by parsing the log files to a database after the simulator executes the simulation and only provide a specific time period when requested through an ajax call. This will allow for lazy loading of one time period at a time in Sim Visualizer instead of the whole data set. This should improve performance in Sim Visualizer and allow for larger log files to be visualized.
- The strategic airlift model introduced in this paper is coarse and refinements could be introduced to reduce assumptions or capture additional details. Some examples of this are:
  - Include maintenance and crew rest as part of the aircraft atomic model;
  - Expand the routing algorithm to account for airspace boundaries and other non-straight line routing;
  - Allow for multiple types of aircraft on a given route; and
  - Expand support for limiting the number of aircraft a given location can handle.
- The modular nature of Sim Manager allows for extension by adding different user interfaces. Instead of having the operator define their experiment through XML and text files, a front end map interface could be created. The operator would place pins on a map and the interface would generate the input files for Sim Manager automatically.

8 CONCLUSION

Operationalizing constructive simulation for problems such as strategic airlift, allow the modeling and simulation community to further engage operators. This engagement is enabled by tools such as the one developed in this paper. These tools enable the rapid definition and implementation of experiments for operators with limited knowledge and experience in modeling and simulation.

The Arctic logistical support base problem was used as a simple example to demonstrate the power of Sim Manager. Non-technical operators could easily use the XML format defined in this paper to construct new experiments and execute them in simulation software. The flexible JSON logging format defined in this paper allows for the rapid exchange of data between simulation software and a visualization system. Communicating output to the user through maps, images, and text allows users to quickly convert simulation results into useful knowledge. These visualizations help bridge the gap for the operator between the technical and business world.

With an easy to use system such as Sim Manager and a well thought through model such as strategic airlift, there are fewer and fewer gaps for operators to become involved in modeling and simulation. Using the tools, frameworks, and models presented in this paper an operator can easily extend these to other problem domains such as search and rescue, sea transport, and even non-military domains such as commercial supply chain.

ACKNOWLEDGEMENT

I would like to thank Dr Peter Dobias from Defence Research and Development Canada (DRDC) for providing the initial idea for the project. I would also like to thank Dr Cristina Ruiz and Bruno St-Aubin from Carleton University for their help with defining the XML format and guidance on developing the map visualization.

REFERENCES

[1] Osman Balci. 2012. A life cycle for modeling and simulation. *SIMULATION* 88, 7 (2012), 870–883.
[2] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. 2016. DevOps. *IEEE Software* 33, 3 (2016), 94–100.
[3] Margaret L. Loper. 2015. *The Modeling and Simulation Life Cycle Process.* Springer London, London, 17–27.
[4] Government of Canada. 2019. *Strong, Secure, Engaged.* https://www.canada.ca/en/department-national-defence/corporate/policies-standards/canada-defence-policy.html
[5] Alen Simec and Magdalena Maglicic. 2014. Comparison of JSON and XML Data Formats. , 272-275 pages.
[6] Steven Vaughan-Nichols. 2016. Containers vs. virtual machines: How to tell which is the right choice for your enterprise. *Network World (Online)* (May 10 2016).
[7] G. Wainer and Sixuan Wang. 2017. MAMS: Mashup Architecture with Modeling and Simulation as a Service. *Journal of Computational Science* 21 (05 2017).
[8] Colin Ware. 2012. *Information Visualization: Perception for Design* (3 ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[9] Public Works and Translation Bureau Government Services Canada. 2019. strategic airlift [1 record] - TERMIUM Plus® - Search - TERMIUM Plus®. https://www.btb.termiumplus.gc.ca/tpv2alpha/alpha-eng.html?lang=eng&i=1&srchtxt=strategic+airlift&index=alt&codom2nd_wet=1#resultrecs
[10] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. 2000. *Theory of Modeling and Simulation* (2nd ed.). Academic Press, Inc., Orlando, FL, USA.