The Space Congress® Proceedings

1993 (30th) Yesterday's Vision is Tomorrow's Reality

Apr 27th, 2:00 PM - 5:00 PM

Paper Session I-A - Modeling Current and Future Launch Vehicle Processing Using Object-Oriented Simulation Techniques

D. G. Linton
Dept Engineering Bldg-407 Univ of Central Florida

S. Khajenoori
Dept Engineering Bldg-407 Univ of Central Florida

M. Heileman
Space Systems Division

K. Halder
Graduate Research Assists, UCF

H. Cat
Graduate Research Assists, UCF

See next page for additional authors

Presenter Information

D. G. Linton, S. Khajenoori, M. Heileman, K. Halder, H. Cat, G. Hebert, and V. Bullington

Session: **Advanced Technology Development**

Modeling Current and Future Launch Vehicle Processing Using Object-Oriented Simulation Techniques

by

D. G. Linton
Assoc Prof
Elec & Comp Engr Dept
Engineering Bldg-407
Univ of Central Florida
P. O. Box 162450
Orlando, FL 32816-2450
Voice:407-823-2320
FAX:407-823-5835
E-mail: dgl@engr.ucf.edu

S. Khajenoori
Assist Prof
Elec & Comp Engr Dept
Engineering Bldg-407
Univ of Central Florida
P. O. Box 162450
Orlando, FL 32816-2450

M. Heileman
Advanced Pgms Engr
Space Systems Division
MS ZK11
Rockwell Int'l Corp
P. O. Box 21105
KSC, FL 32815

K. Halder, H. Cat,
V. Bullington, G. Hebert
Graduate Research Assists
Elec & Comp Engr Dept
Univ of Central Florida
P. O. Box 162450
Orlando, FL 32816-2450

Abstract

STARSIM, an acronym for Space Transportation Activities and Resources Simulation, is an object-oriented, menu-driven, user-friendly, decision support system for simulating National Space Transportation System (NSTS) processing, as well as Personnel Launch System (PLS)-National Launch System (NLS), PLS-Proton, PLS-Titan IV, Hermes-Ariane 5 and Cargo Transfer Return Vehicle (CTRV) processing. For each launch system modeled, output is displayed numerically (for global statistical information), in pie chart form (to visualize percentages of subcategories associated with a main category) and in Gantt chart form (for visualizing when and where each launch vehicle experiences waiting, processing, blocking and maintenance periods, and the reasons for blocking). Users may input a comprehensive set of system parameters (e.g., number of launch vehicles, processing times at each facility, number of bays at a particular facility) using a window-based environment, or by supplying an existing input data file. Data for existing launch systems and representative data for proposed systems are used to illustrate output for the models mentioned above. The object-oriented methodology employed in the initial model (i.e., NSTS processing) permitted additional models to be implemented in a minimum amount of time and effort.

INTRODUCTION

STARSIM is an object-oriented, discrete-event simulation tool to model scenarios such as the space-shuttle ground processing activities at the Kennedy Space Center. Originally designed to model only the critical path portion of the space-shuttle turn-around flow, STARSIM was later easily adapted to include the processing of the Solid Rocket Boosters (SRBs) and the External Tank (ET), as well as models of processing for PLS/NLS, PLS-Proton, PLS-Titan IV, Hermes-Ariane 5 and CTRV. It is the object-oriented design technique incorporated during the initial development of STARSIM which allows models of existing and future space-vehicle processing scenarios to be developed within a relatively short period of time.

OVERVIEW OF THE NATIONAL SPACE TRANSPORTATION SYSTEM (NSTS) MODEL

A flow diagram of NSTS processing is shown in Figure 1 (oval shapes represent servers, parallel lines indicate queues, rectangles are resource stores and arrows show the path of an entity with, where appropriate, the name of the entity adjacent to the arrow). An alphabetized list of important acronyms is shown at the conclusion of this paper. As depicted in Fig. 1, space-shuttle orbiters or orbiter vehicles (OVs) which are launched from Kennedy Space Center (KSC) may land at Edwards Air Force Base (EAFB) or KSC. OVs landing at EAFB are serviced by a crew which, using a ferry kit, prepares the OV for return to KSC aboard a shuttle carrier aircraft (SCA). Ferry kits return to EAFB by truck and SCAs are flown back to EAFB. An OV which either arrives at KSC from EAFB aboard an SCA, or which lands at KSC, is first linked with a Firing Room (FR) at the Launch Control Center (LCC) and then begins processing in a bay at the Orbiter Processing Facility (OPF). As OPF servicing is initiated, the OV's main engines and hypergolic-fueled engines (HGUs, for hypergolic units) begin parallel processing at the Engine Shop (ES) and Hypergolic Maintenance Facility (HMF), respectively. OPF processing is completed when the OV main engines and hypergolic-fueled engines are reinstalled and checked out. If a mobile launch platform (MLP) is available, OVs which complete OPF processing join with an MLP in one of the bays in the Vehicle Assembly Building (VAB). MLP availability implies that the solid rocket boosters and external tank have been integrated with the MLP. After VAB processing, if one of two launch pads are empty, the OV-MLP pair begins processing at the launch pad (LP). If at least twenty-one days has elapsed since the last launch (a NASA requirement) and no other OVs are on a mission, shuttles which finish servicing at the pad are launched. During a mission, an OV is under control of the Mission Control Center (MCC), which selects the eventual landing site (EAFB or KSC). The OV completes its mission and lands at either KSC or EAFB, while the MLP begins post-launch processing in preparation for an eventual match with an OV at the VAB, and the solid rocket boosters (SRBs) splash down in the ocean for eventual recovery by ship. After the SRBs are brought to Port Canaveral via ship, SRB disassembly begins. After disassembly and refurbishment (at the Refurbishment Subassembly Facility (RSF)), the SRB components, the parachute and the Solid Rocket Motor (SRM) are processed at the Rotatation Processing and Surge Facility (RPSF) in preparation for integration with the External Tank (ET), which was checked out in the VAB, and the previously mated OV-MLP-FR triple.

Prior to entering the OPF, each OV is attached to a Firing Room (FR) associated with the Launch Control Center (LCC). After a launch, the FR becomes available for use by other OVs. Each OV which enters the OPF after three continuous years of operation experiences an Orbiter Maintenance Down Period (OMDP). Only one OV may undergo OMDP at a time and each OMDP requires about four additional months of processing in the OPF. A FR is not linked with an OV during OMDP but must be re-linked with the OV before normal OPF processing begins.

By incorporating the object-oriented approach during the design of the STARSIM version of NSTS shown in Fig. 1, the adaptation of models for the PLS/NLS, PLS-Proton, PLS-Titan IV, Hermes-Ariane 5 and CTRV systems was completed in about four man-months. An outline of the design process is described below.

DESIGN ASPECTS OF STARSIM

The object-oriented approach used to design STARSIM resulted in the creation of eight basic class types named ENTITY, QUEUE, SERVER, RESOURCE, RANDOM, GENERATOR, MANAGER and a special graphics object, REPORTER, for producing graphical output. REPORTER Object was designed especially to display Pie Chart and Gantt Chart output from any model in a Windowed environment. For example, with OV index numbers (1, 2, ...) and processing facility acronyms (EAFB, SCA, ...) on the vertical axis and time (in days) along the horizontal axis, Fig. 2 shows all waiting, servicing, blocking and OMDP times at each facility for all OV flows through the NSTS model during the first 150-plus days of a 3650-day simulation run. The user may scroll up or down (to see data for other OVs), scroll right or left (to view visible OVs at different periods of time) and may change the time scale (to zoom-in or zoom-out on a particular area of interest). By consulting the accompanying legend for color or black-and-white terminals

(see Fig. 2), notice that OV 3 experiences blocking at the OPF. By clicking a mouse on this area of blocking, the specific reason for the blocking is displayed (see Fig. 3); namely, the 45-day MLP preparation time in the VAB is not completed. A sample Pie Chart (based on a cost model for NSTS) is shown in Fig. 4.

The eight base object-classes may be employed in a cookie-cutter-like approach to stamp out similar objects. A description of the base classes used in STARSIM and the adapted models for PLS/NLS, PLS Titan IV, PLS Proton, Hermes/Ariane 5, and CTRV are described below:

ENTITY:

An ENTITY is an item that flows through the simulation network with a pre-determined path encompassing various activities. There are three different types of entities. The first type is used to represent reusable components like OVs and SRBs. The second type, ET for example, is destroyed after launch. The third type (e.g., Solid Rocket Motor (SRM), Parachute, and Engine) flows through the model network as a part of the first type of entity, is separated at some location to follow its own route, and then is re-assembled with the parent entity.

SERVER:

A location (e.g., OPF, RPSF, Parachute Facility) where other entities (e.g., OV, SRB, Parachute) spend time being processed by a server. Any delay experienced by an entity is modeled by passing the entity to a server. Servers do not have to be fixed (e.g., movable servers include SCA, SRB Ship and ET Barge).

RESOURCE:

A resource (e.g., MLP and the Ferry Kit) is needed by a server in order to process an entity. The server may have to prepare the required resource before being able to process the next entity. Some amount of post-processing time may also be needed.

QUEUE:

A QUEUE models a location where entities may wait for an available server. The capacity of a queue can range from any non-negative number to infinity; a zero-size queue is used to hold entities at blocked server locations when no server is available at the following location along a route.

RANDOM:

This class is used to instantiate the Pseudo-Random Number Generator (PRNG) for the different models. The PRNG is used to generate random numbers from various user-defined distributions.

GENERATOR:

Entities are created by generators. Different types of generators are needed in order to generate different types of entities (e.g., OV, ET, SRB). Generators can be attached at any point in a model, resulting in the created entities entering the system at that location.

MANAGER:

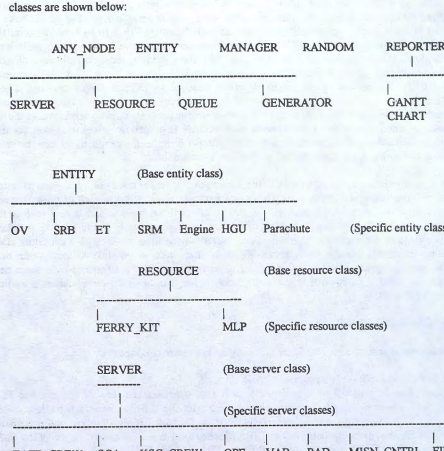
Only one manager object exists in a model. The simulation is controlled by this manager object in the sense that all the scheduling, as well as invoking of objects associated with a specific event, is performed by the manager.

REPORTER:

As discussed above, this object permits graphical output to be displayed.

There is a common ancestor class, ANY_NODE, from which the SERVER, RESOURCE, QUEUE and GENERATOR classes have been inherited. The ANY_NODE class contains the common attributes (e.g., an integer-valued identifier used to differentiate between OVs) present in the derived classes and allows

the derived classes to be considered generic types. An abbreviated set of hierarchical charts for the basic classes are shown below:



In order to instantiate (or create an instance of) the entities, facilities/servers and resources belonging to any specific group, a separate class for that group must be inherited from the base object-class. For example, OVs are not instantiated directly from the ENTITY class. Instead, a class named OrbiterVehicle is first derived from ENTITY, and then all the OVs are instantiated from this derived class. In this way, features belonging exclusively to any particular group of entities can be incorporated into the appropriate derived class. Any method (e.g., update_cum_served, a subprogram which updates the cumulative number of OVs processed by a server group) can be easily overridden to fit the needs of each individual group.

A graphical representation of the above description was developed for documentation purposes. Figure 5 is the legend for all graphical depictions of objects/classes. Referring to Fig. 5, ENTITY is an example of a SUPER_CLASS_NAME and OrbiterVehicle is an example of a CLASS_NAME. The legend briefly defines public and private methods and attributes, shows how these methods and attributes will appear inside a rectangle representing a class, defines the concepts of static variables and virtual methods and depicts interface representations for all methods.

Figure 6 shows the graphical version of the base class ENTITY. Since the intent of this paper is to give an overview of the object-oriented approach, details (e.g., definitions of all parameters in Fig. 6) will be ignored in the ensuing discussions. As seen in the heirarchial diagram for ENTITY, there are seven objects (OV, SRB, ET, SRM, Engine, HGU, and Parachute) which may be inherited from the ENTITY

base class. Figure 7 is a graphical version of one of these inheritable classes, OrbiterVehicle (used to model OVs in the NSTS STARSIM model). Some of these attributes (see Fig. 7), e.g. no_units, time_param and firing_room_type, are declared as static -- this permits a single copy of these attributes to be maintained for all the OVs instantiated from this class. The attribute firing_room_type identifies the group of firing rooms needed for the NSTS simulation. Note that even though this variable could have been made a global variable, placing it in the OV class as a static member reduces the chance of side-effects. The attribute time_param is a data structure used to hold the different types of timing parameters associated with OVs for generating the length of service required at different server locations. These timing parameters include the fixed-length service time for a deterministic-mode simulation, or minimum, maximum and mode parameters for a triangular distribution used to generate random service times during a stochastic-mode simulation. There are also some new methods (e.g., get_fir_room) used only for this class of entity, while some other methods (e.g., clear_history) have been overridden to gain extra or completely different functionality than obtainable from the base methods.

The same type of discussion as above applies to the derivation (i.e., inheritance) of other types of entity classes. It is worth mentioning that the time_param data structure is contained in each of these instance classes. If these times (or parameters), necessary to obtain service times, were included with each of the specific servers, any of these servers processing more than one type of entity would have to maintain separate data structures for each type of entity. Thus, including the time_param with each entity class relieves the server of this burden. This design approach also makes it possible to incorporate new parameters for generating service times with a minimum amount of additional effort and code, since new parameters can be added to the specific type of entity class, rather than to all the server classes attending that particular type of entity.

ADDITIONAL MODELS DEVELOPED

Using the class-objects designed for NSTS, the following systems were also modeled:

- 1.5 STAGE PLS/NLS-2 (without Boosers):
  This is a model of a proposed system that simulates the flow of the manned PLS (Personnel Launch System) vehicle along with the ET-like core-stage module called NLS (National Launch System). Included in this model is also the flow and processing for another entity called ADAPTER, which is used to mount the PLS on top of the NLS. The PLS is a reusable entity, while the NLS and ADAPTER units are destroyed upon each launch.

- PLS/NLS-1 (with Boosters):
  An extension to the previous model, PLS/NLS-1 includes the detailed processing activities at the various SRB facilities. The SRB is mated to the NLS and ADAPTER assembly on top of the MLT (Mobile Launch Transporter) in the VAB bay, before a PLS can be attached to them. Similar to the NSTS model with SRB processing, this model also includes parachute as well as SRM processing.

- 2-STAGE PLS/NLS-2:
  This model is also an extension to the 1.5-STAGE PLS/NLS-2 model. The model processes an extra module called the 2nd stage, which is mounted on top of an NLS core stage (hence the term NLS 2-stage). The PLS can then be attached on top of this assembly via an ADAPTER unit. This 2nd stage unit is an expendible.

- PLS/TITAN-IV:
  This model simulates another proposed launch activity which includes the PLS as the major entity which is launched on top of the core-stage called the TITAN-IV rocket. After being created at the manufacturer's site, the core stage is brought via a barge to KSC, where it is assembled on an MLP in the Vehicle Integration Building (VIB).

Afterwards, it is sent to the SRM Mate Assembly Building (SMAB) where it is mated to a Solid Rocket Motor (SRM) that has been processed by the SRM Preparation Facility after being created by the manufacturer. This assembly is then passed to the launch complex where a PLS is mated via the processed ADAPTER and then launched. All the entities in this model except the PLS are expendible and are terminated after launch.

- PLS/PROTON:
  This models another proposed scenario with the PLS as the reusable, main entity. The other entity modules present in this model are the 1st STAGE, 2nd STAGE, 3rd STAGE, Strap-on-tank and ADAPTER. All these entities, upon finishing their own processing chore, are mated with the PLS in the Horizontal Integration and Testing Facility, sent via a transporter to the pad for further processing and finally launched. Only the PLS is recovered after a mission.

- HERMES/ARIANE-5:
  This is also a model simulating proposed scenarios including the European HERMES space plane on an ARIANE-5 rocket as the main entity. The other required entities are the P230 Booster Pairs, Central Body or CORE and Resource Module or ADAPTER. Every type of entity except the HERMES is expendible.

- Medium Reusable CTRV:
  This is the most recent model adapted from the STARSIM project. The main entity is the CTRV (Cargo Transport and Return Vehicle) which is used to ferry the different types of cargoes to and from the space-station.

First, flow charts similar to Fig. 1 were developed for each of the above systems. Then the appropriate STARSIM base classes were used to inherit objects with the required characteristics. Output analagous to Figures 2-4 may be obtained for each of the above systems.

SUMMARY AND CONCLUSIONS

The nature of the physical situations to be simulated (e.g., various kinds of servers, resources, and entities) resulted in the choice of the object-oriented language C++ for implementing STARSIM and the adapted models. The processing scenarios of all systems are similar to one another in their overall structure, differing only in the behavior of the different objects. These characteristics require code and interface inheritance for the objects in each of the simulation models. As a result, the core of STARSIM consists of a set of base objects needed by each model. Inheriting code within core object modules helps avoid redundancy. Inheriting the interfaces of the base object classes allows the derived specialized classes to obtain additional and/or completely different behaviors. These capabilities would not be possible in a non-object-oriented implementation.

Another advantage of using an object-oriented design methodology resulted in the locality of code. For instance, any modification to be performed to a specific model can be achieved by making changes to a single location within one method. If a change is to be made to a specific type of derived object, only the methods of that particular instance class need be altered. If a change is to be made which will affect all of the derived classes of some base class, then the base class method is the place to make those changes, since they will be propagated through the hierarchy to the derived classes. In short, the object-oriented approach resulted in more reliable code that was easier to maintain and, consequently, created an environment which enhanced productivity (i.e., seven distinct systems were adapted and verified in four man-months)

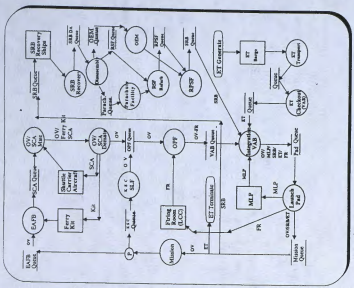


Figure 1 STARSIM Flow Diagram with P=PR (Landing at KSC).

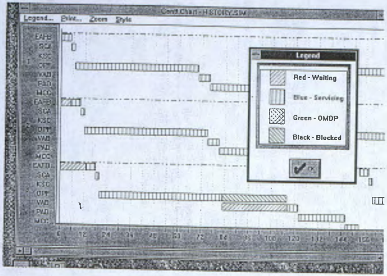


Figure 2. Gantt chart output for 150-plus days (time scale = 6 days).

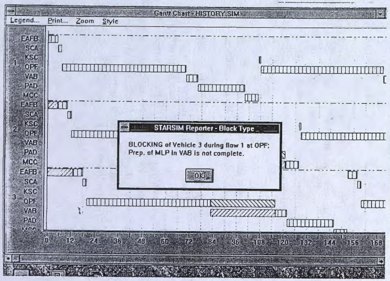


Figure 3. Gantt chart with reason for blocking of OV 3 at OPF displayed.

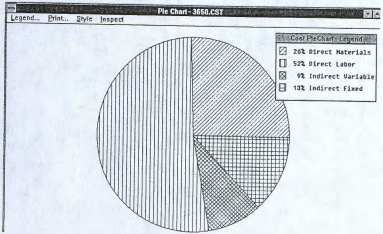


Figure 4. Pie chart of cost percentages of four cost categories.

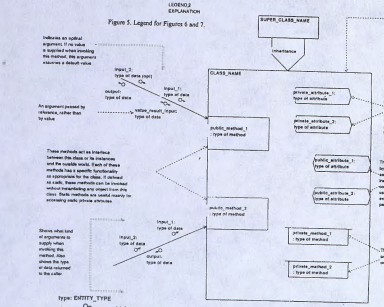


Figure 5. Legend for Figures 6 and 7.

ALPHABETIZED LIST OF IMPORTANT ACRONYMS

CTRV	Cargo Transfer Return Vehicle
EAFB	Edwards Air Force Base
ES	Engine Shop
ET	External Tank
FR	Firing Room
HGU	Hypergolic Unit
HMF	Hypergolic Maintenance Facility
HPF	Hypergolic Processing Facility
KSC	Kennedy Space Center
LCC	Launch Control Center (where FRs are located)
LP	Launch Pad
MCC	Mission Control Center
MLP	Mobile Launch Platform
MLT	Mobile Launch Transporter
NASA	National Aeronautics and Space Administration
NLS	National Launch System
NSTS	National Space Transportation System
OEM	Original Equipment Manufacturer
OMDP	Orbiter Maintenance Down Period
OPF	Orbiter Processing Facility
OV	Orbiter Vehicle
PLS	Personnel Launch System
RO	Reporter Object
RPSF	Rotation, Processing and Surge Facility
RSF	Refurbishment Subassembly Facility
SCA	Shuttle Carrier Aircraft
SLF	Shuttle Landing Facility (at KSC)
SRB	Solid Rocket Booster
SRM	Solid Rocket Motor
STARSIM	Space Transportation Activities and Resources SIMulation
VAB	Vehicle Assembly Building

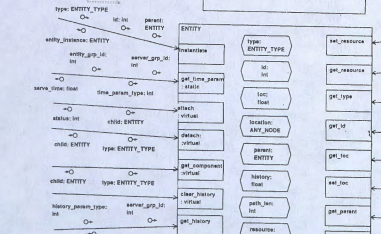


Figure 6. Graphical depiction of methods, attributes and interfaces of base class ENTITY

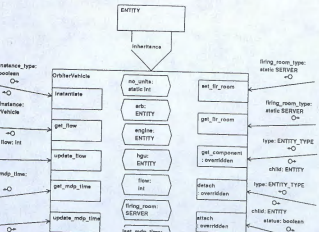


Figure 7. Graphical depiction of methods, attributes and interfaces of inherited object OrbiterVehicle.