



Supporting the Reuse of Algorithmic Simulation Models

Nicholas Keller (presenter), Bernard Zeigler, Doohwan Kim, Chase
Anderson, James Cenev

RTSync Corp & nou Systems Inc

www.rtsync.com www.nou-systems.com

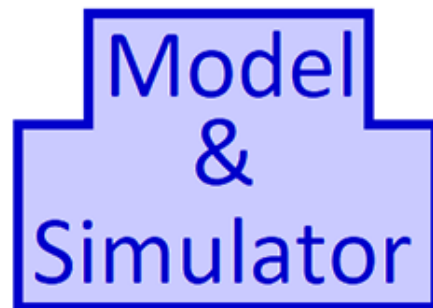
July 20, 2020



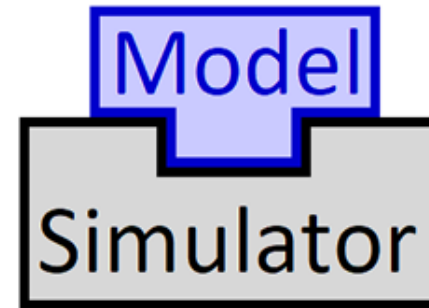
Research problem: simulation model reuse remains elusive

- ▶ Custom monolithic simulation models are the norm in government and industry
- ▶ Model and simulator not kept separate
- ▶ Simulation models can't be shared
 - Modeling teams are constantly 'reinventing the wheel'
- ▶ DEVS modeling formalism allows for model reuse, but has a steep learning curve that discourages use outside of academia
- ▶ Algorithmic models are specialized DEVS atomic models that return an output in the same (simulation) time instant their inputs are received
 - They can be thought of as memoryless functions
- ▶ Our solution: reduce the steep learning curve of DEVS through a custom DEVS M&S IDE; start with algorithmic models then generalize so that non-experts can reap the rewards of model reuse.

Current practice



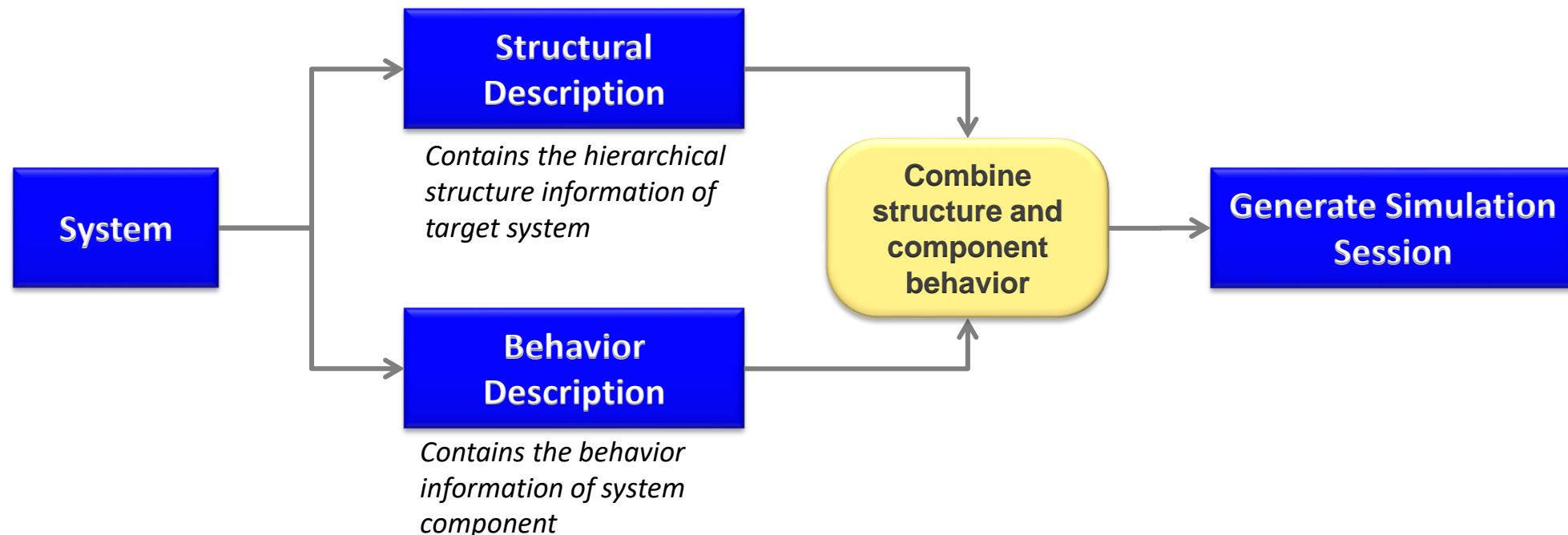
The ideal



- ▶ **DEVS (Discrete Event System Specification)**
- ▶ **MS4 Me**
- ▶ **Algorithmic model reuse case study**
- ▶ **Future work**

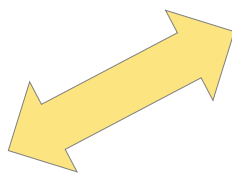
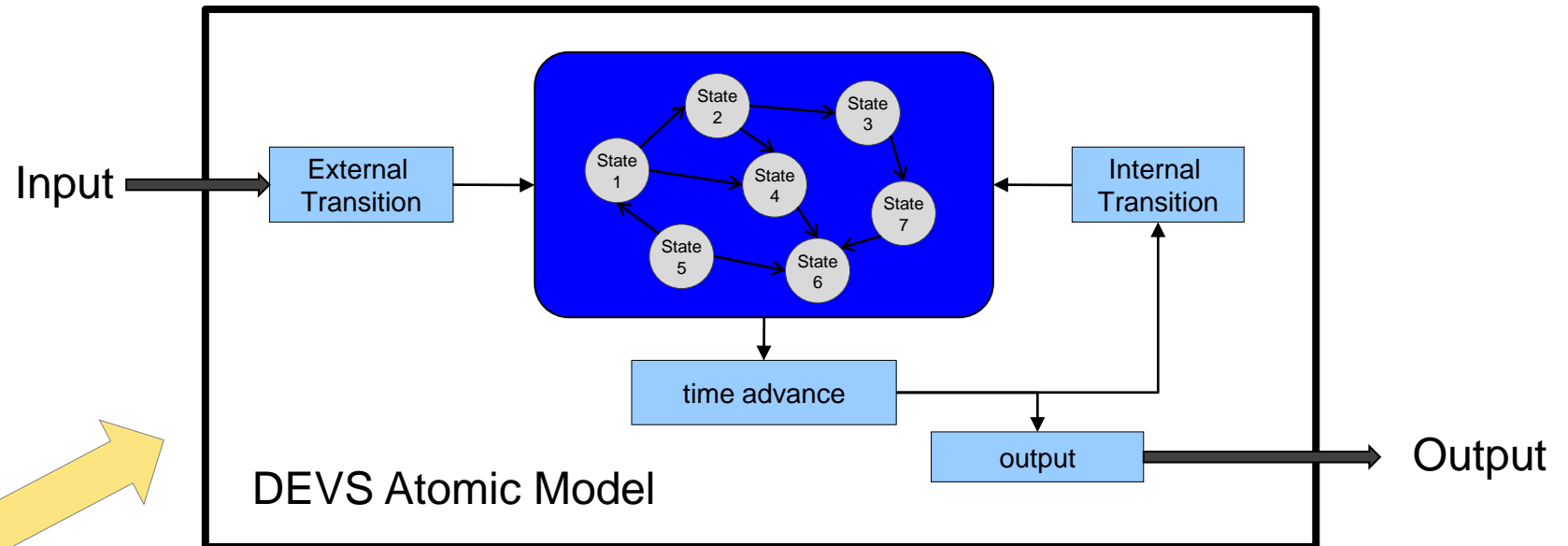
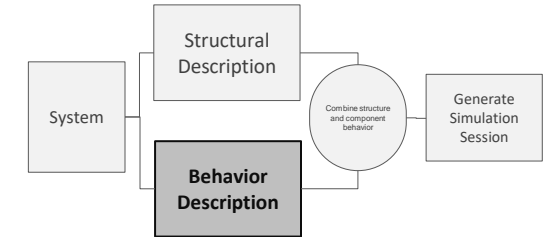
DEVS Modeling & Simulation Approach

- ▶ **DEVS (Discrete Event System Specification) formalism provides an engine for advanced Modeling & Simulation (M&S) technology to support “build and test”**
 - DEVS evaluates different design alternatives of a system, where information of system structure as well as component behavior can be implemented in an integrated simulation environment.



Behavior Description : DEVS Model

- ▶ DEVS model describes the behavior of a component based on state transition that displays discrete event activity of system.



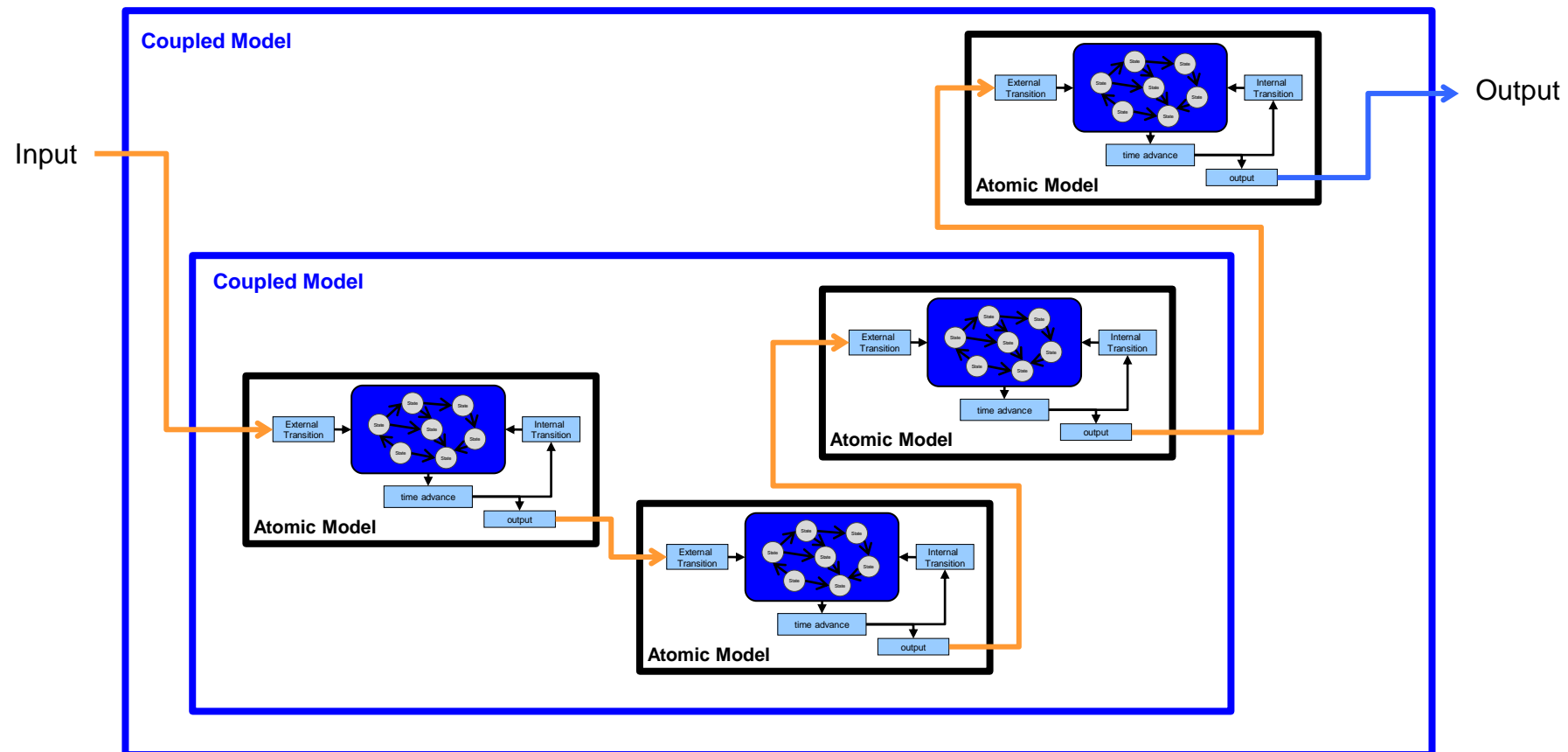
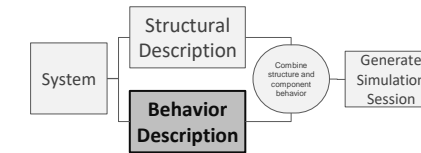
The atomic DEVS Model for Ping-Pong Players
 The atomic DEVS model for player A of Fig. 1 is given $\text{PlayerA} = \langle X, Y, S, \theta_0, \tau_a, \delta_{ext}, \delta_{int}, \lambda \rangle$ such that

$$\begin{aligned}
 X &= \{?receive\} \\
 Y &= \{!send\} \\
 S &= \{(d, \sigma) \mid d \in \{Wait, Send\}, \sigma \in \mathbb{T}^\infty\} \\
 s_0 &= (Send, 0.1) \\
 \tau_a(s) &= \sigma \text{ for all } s \in S \\
 \delta_{ext}(((Wait, \sigma), ?receive) &= (Send, 0.1) \\
 \delta_{int}(Send, \sigma) &= (Wait, \infty) \\
 \delta_{int}(Wait, \sigma) &= (Send, 0.1) \\
 \lambda(Send, \sigma) &= !send \\
 \lambda(Wait, \sigma) &= \phi
 \end{aligned}$$

Both Player A and Player B are atomic DEVS models.

Behavior Description : DEVS Model

- ▶ Multiple model constructs a bigger coupled model, where each model interfaced via input and output port.

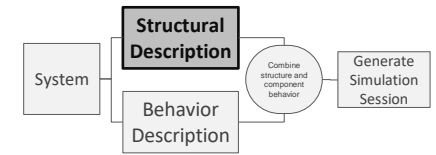


Structural Description : System Entity Structure (SES)

The coupled DEVS model

The ping-pong game of Fig. 1 can be modeled as a coupled DEVS model

$N = \langle X, Y, D, \{M_i\}, C_{xx}, C_{yx}, C_{yy}, Select \rangle$ where $X = \{\}$; $Y = \{\}$; $D = \{A, B\}$; M_A and M_B is described as above; $C_{xx} = \{\}$; $C_{yx} = \{(A. !send, B. ?receive), (B. !send, A. ?receive)\}$; and $C_{yy}(A. !send) = \phi, C_{yy}(B. !send) = \phi$.



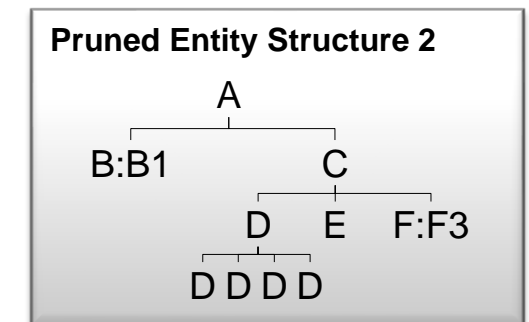
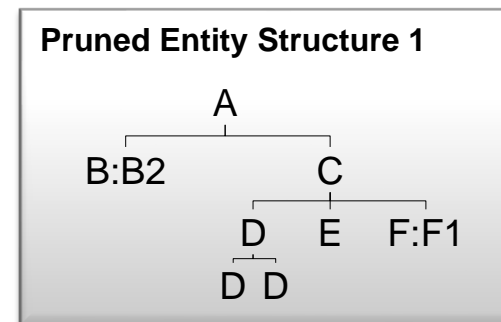
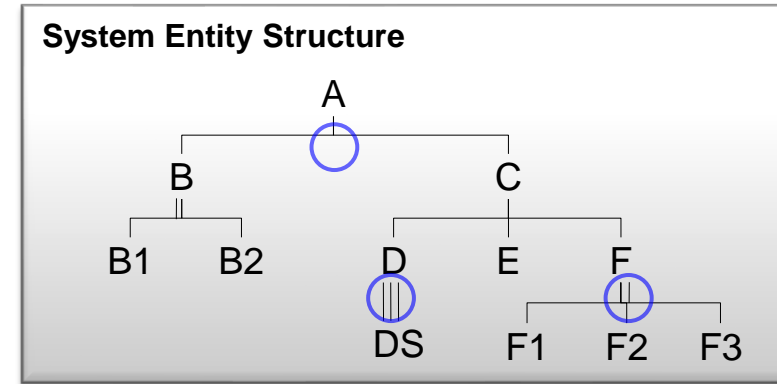
► **The System Entity Structure (SES) is a formal framework to represent the elements of a system (or world) and their relationships in hierarchical manner.**

Aspects represent ways of taking things apart into more detailed ones and labeled decomposition relation between the parent and the children

Specialization categorizes things in specific forms and is a labeled relation that expresses alternative choices.

Multi-aspects are aspects for which the components are all of the same kind.

Pruning:
An operation to cut off structure in a SES that is not needed to meet particular problem specifications.



Why was MS4 Me* developed for DEVS based modeling and simulation?

- ▶ **DEVS representation formalism needs to be easier for non-experts to understand**
- ▶ **Users do not have to learn formal representation of DEVS models and simulation at the beginning**
- ▶ **Mapping DEVS to DEVS Natural Language (DNL) provides abstraction layer and makes communication easier**
- ▶ **Graphic User Interface can be built on top of the abstraction layers**
- ▶ **Flow of information from Top-Down to Bottom-up, High Level and Low Level (codes and GUI)**

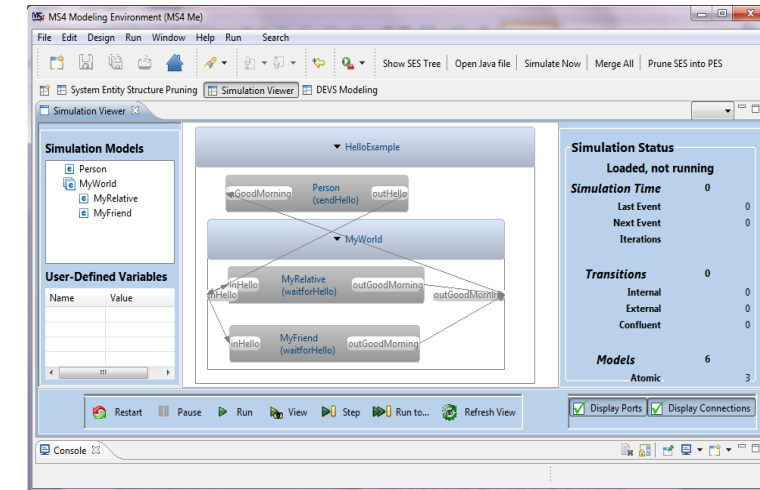
* MS4 Me is a commercial M&S Integrated Development Environment (IDE) maintained by MS4 Systems since 2011.

MS4 Modeling Environment

- ▶ **Software Tool for DEVS Modeling and Simulation**
 - Based on Java and Eclipse RCP environment
- ▶ **Top down modeling methodology**
 - Sequence Diagram Designer generates a SES file and DNL files
- ▶ **Bottom up modeling methodology**
 - State Diagram Designer generates DNL files

Features

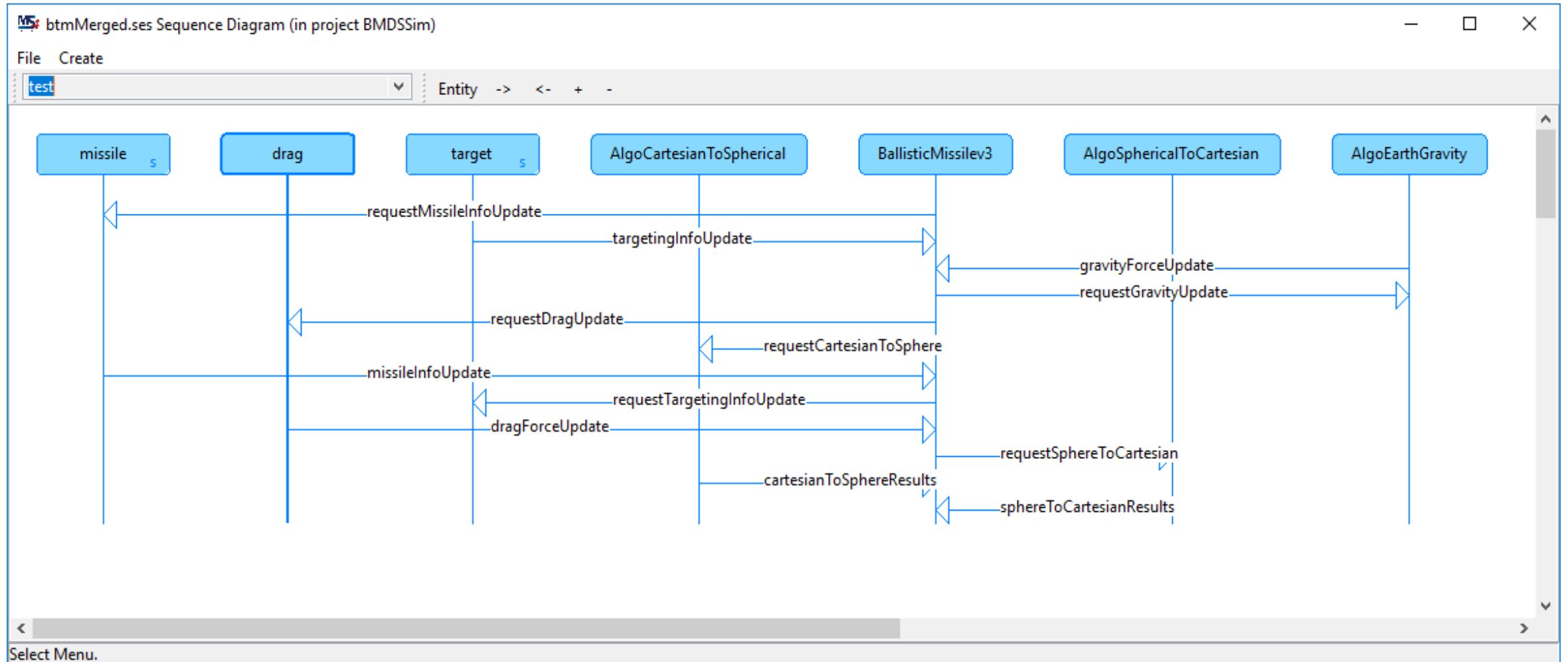
- ▶ **Highlight keywords capability when editing SES and DNL files**
- ▶ **Automatic converting DNL files to Atomic Java Models**
- ▶ **Tree Viewer for a SES file**
- ▶ **Pruning process from Pruning SES document to generate coupled Java models**
- ▶ **Launch page to help directly access to the SES and PES files**
- ▶ **Support dynamic structure model**
- ▶ **Graphing capability**
- ▶ **1 year free trial license: ms4systems.com/pages/downloads.php**



Algorithmic model reuse case study

- ▶ **Ballistic missile trajectory simulation built by a programmer (Chase Anderson) with no prior experience in DEVS or MS4 Me**
 - A key goal is to make DEVS more accessible
- ▶ **24 (3*2*2*2) possible trajectory simulation variations corresponding to different algorithmic model combinations**
 - Algorithmic models
 - 3 drag algorithmic models
 - 2 air density algorithmic models
 - 2 ballistic missile options
 - 2 possible targeting options
- ▶ **MS4 Me makes it easy to define the models and compose and simulate different model combinations**

Interaction diagram for a missile trajectory simulation



Complete model in MS4 Me

The screenshot displays the MS4 Modeling Environment (MS4 Me) interface. The main window shows a simulation model diagram with several interconnected components:

- btm** (bottom) container:
 - AlgoEarthGravity (passive)**: Receives `inrequestGravityUpdate` and outputs `outgravityForceUpdate`.
 - drag** (highlighted in yellow) container:
 - AirDensityTableConst_airDensityCalc (passive)**: Receives `inrequestAirDensity` and outputs `outairDensity`.
 - AlgoDragEqnNone_dragCalc (passive)**: Receives `inrequestDragUpdate` and outputs `outdragForceUpdate` and `outgetAirDensity`.
- AlgoSphericalToCartesian (waitForConversionRequest)**: Receives `inrequestSphereToCartesian` and outputs `outsphereToCartesianResults`.
- Titan2_missile (waitForInfoRequest)**: Receives `inrequestMissileInfoUpdate` and outputs `outmissileInfoUpdate`.
- SouthTarget_target (waitForInfoRequest)**: Receives `inrequestTargetingInfoUpdate` and outputs `outtargetingInfoUpdate`.
- AlgoCartesianToSpherical (waitForConversionRequest)**: Receives `inrequestCartesianToSphere` and outputs `outcartesianToSphereResults`.

Connections between components include `inrequestGravityUpdate`, `inrequestDragUpdate`, `inrequestAirDensity`, `inrequestSphereToCartesian`, `inrequestMissileInfoUpdate`, `inrequestTargetingInfoUpdate`, `inrequestCartesianToSphere`, `outgravityForceUpdate`, `outairDensity`, `outdragForceUpdate`, `outgetAirDensity`, `outsphereToCartesianResults`, `outmissileInfoUpdate`, `outtargetingInfoUpdate`, and `outcartesianToSphereResults`.

Simulation Status
Loaded, not running

Time

Last Event	
Next Event	Infinity
Iterations	139

Transitions

Internal	0
External	0
Confluent	0

Models

Atomic	8
Coupled	3

At the bottom, there is a control bar with buttons for Restart, Pause, Run, View, Step, Run to..., and Refresh View. On the right, there are checkboxes for Display Ports and Display Connections, both of which are checked.

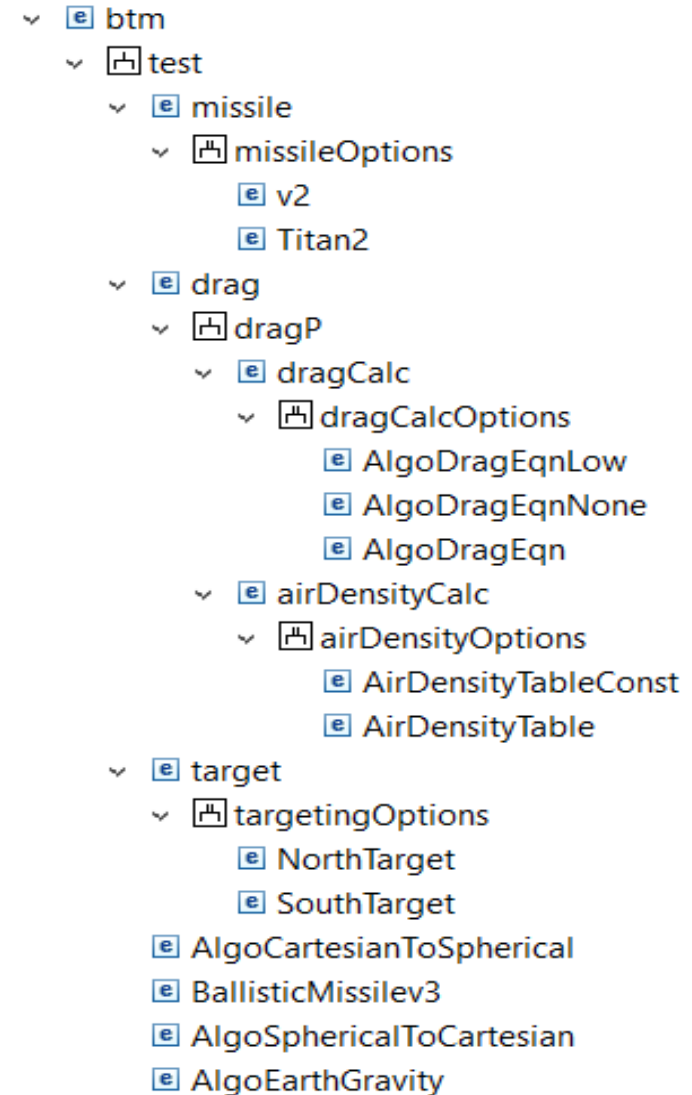
SES outline of a ballistic trajectory missile model

▶ 24 (3*2*2*2) possible trajectory simulation variations corresponding to different algorithmic model combinations

- Algorithmic models
 - 3 drag algorithmic models
 - 2 air density algorithmic models
 - 2 ballistic missile options
 - 2 possible targeting options

▶ MS4 Me provides a graphic pruning interface

▶ Users can also manually specify the pruning using a text file



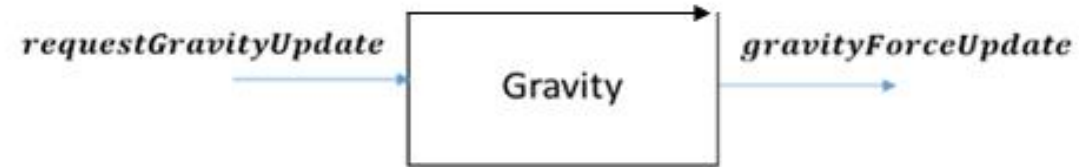
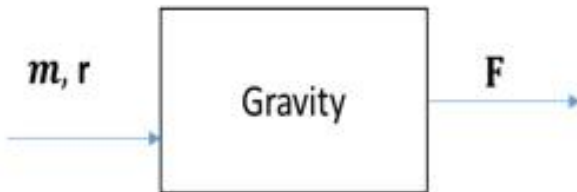
Gravity Algorithmic model

Input: Altitude and mass of object

Output: Force in direction to Earth's center

Parameters: Gravitational constant, earth mass and

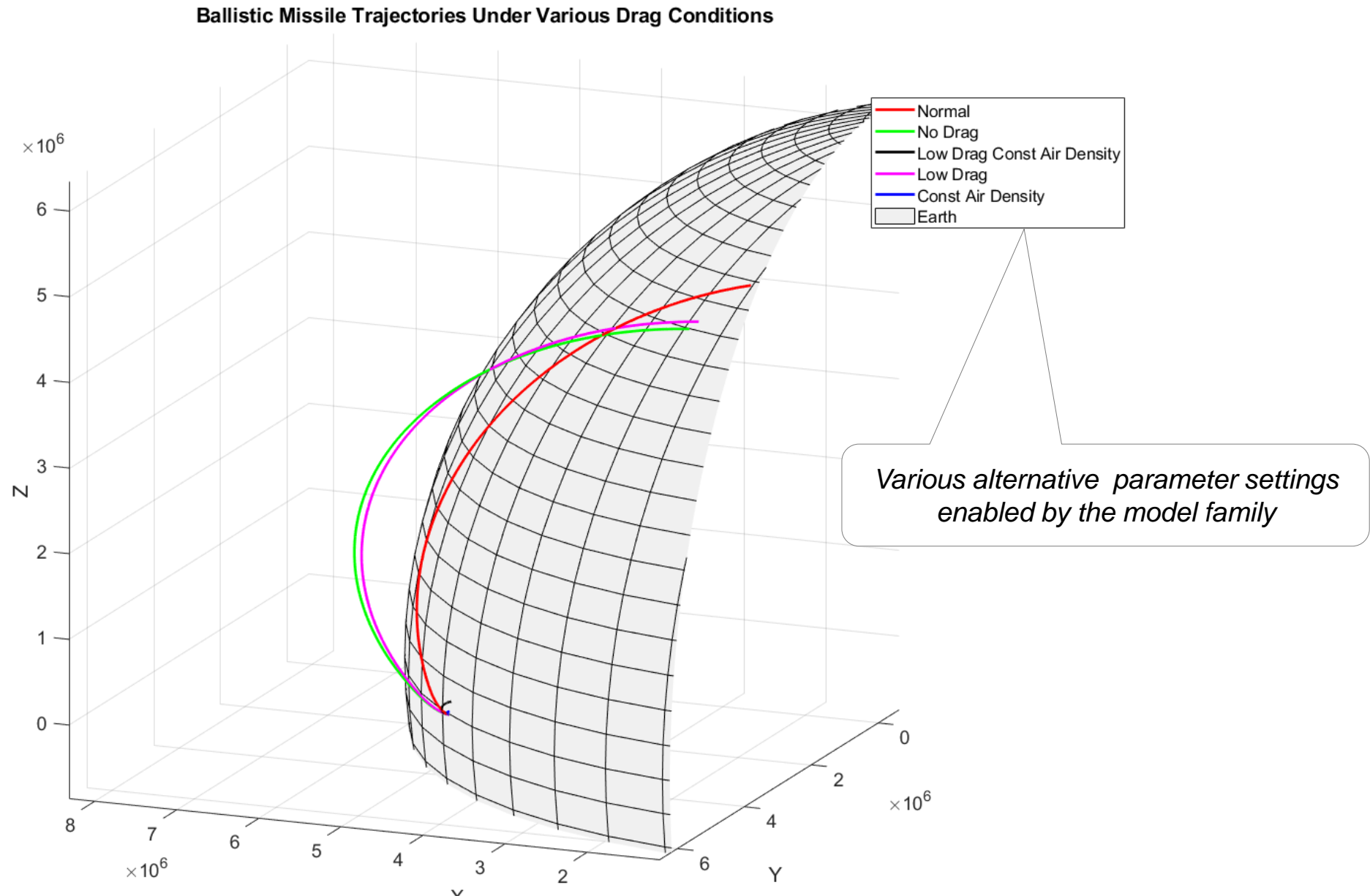
$$F = G \cdot m_e \cdot m_o / r^2$$



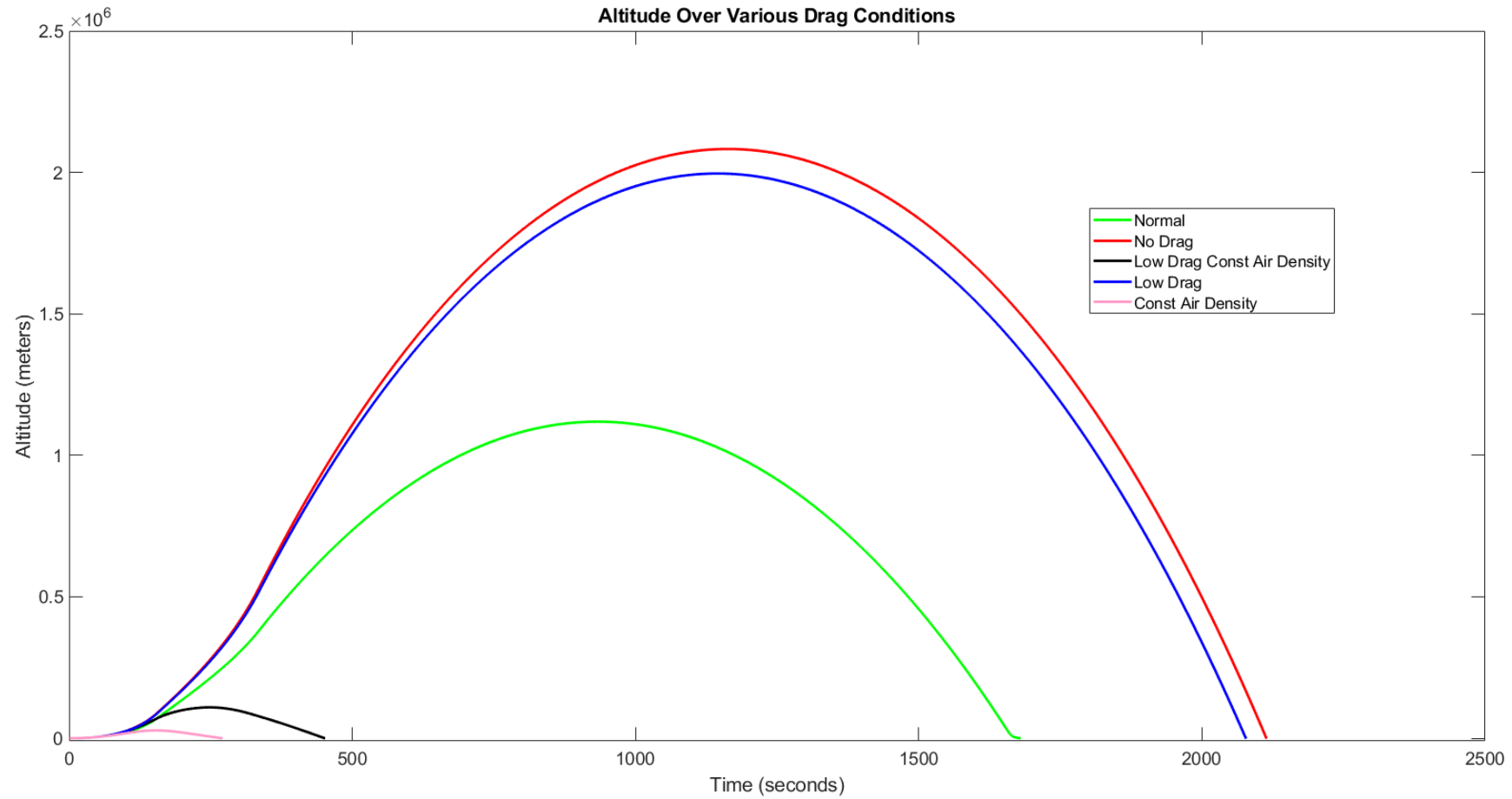
$$m, r \rightarrow G \cdot m_e \cdot m_o / r^2 \rightarrow F$$

- ✓ ● Message Types
- ✓ ● gravityInputParams
 - ⊗= mass (double)
 - ⊗= altitude (double)
- ✓ 🚪 Input Ports
 - > 🚪 requestGravityUpdate (gravityInputParams)
- ✓ 🚪 Output Ports
 - > 🚪 gravityForceUpdate (Double)
- > ○ States
- ✓ ● State Variables
 - ⊗= force (double) = 0
 - ⊗= gravConstant (double)
 - ⊗= earthMass (double)
 - ⊗= earthRadius (double)

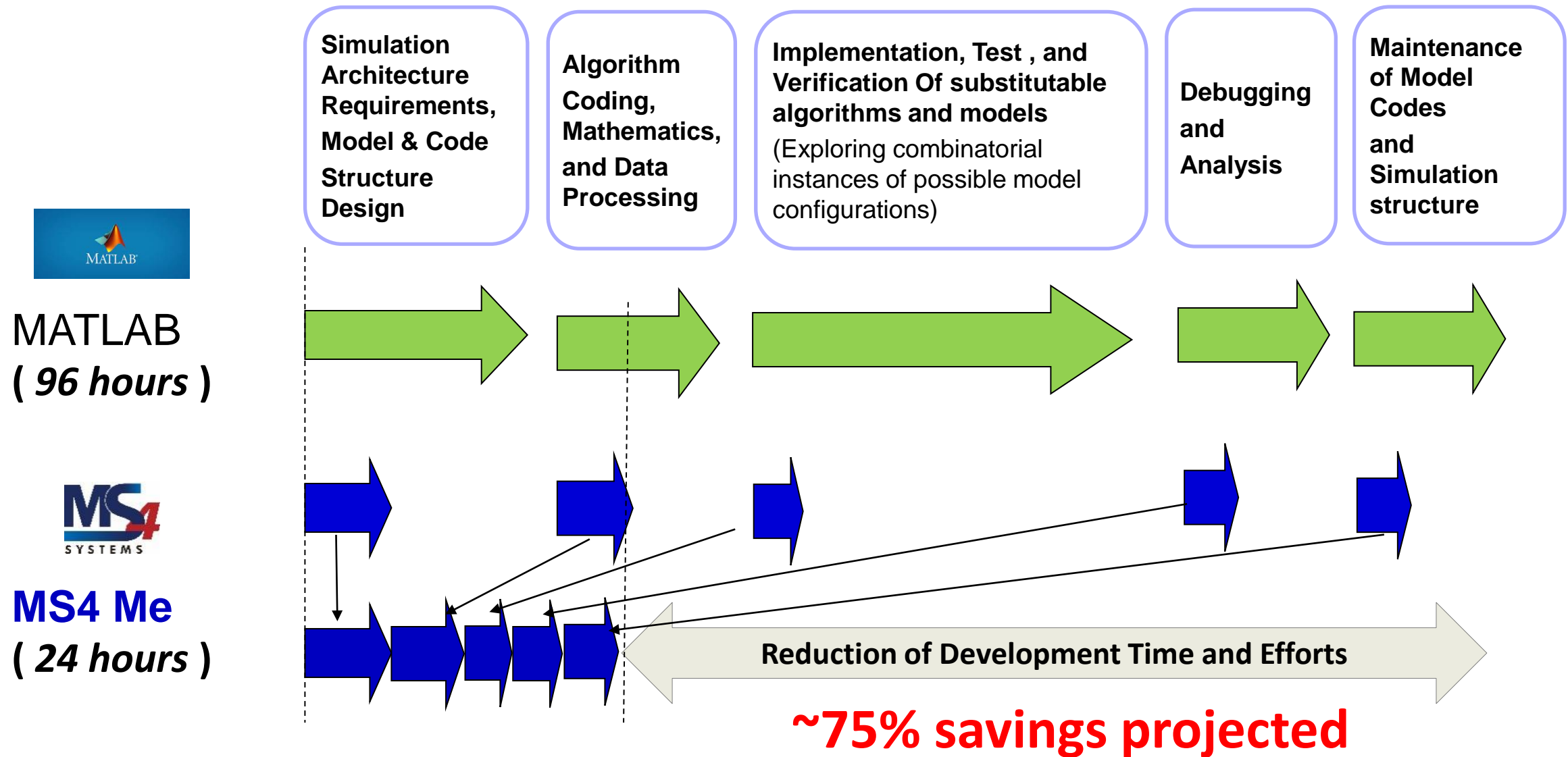
Example Trajectories generated by the model



Altitude Over Time



Comparison of Development Time and Efforts



Continuing and Future work

- ▶ **Expand MS4 Me to better support collaborative model development and reuse: generalize from algorithmic to all DEVS**
- ▶ **Currently input/output ports are Java primitive types or classes, which forces the modeler to document their model to enforce port semantics**
 - We are adding input/output port types, which users can use to semantically define ports
- ▶ **Model reuse requires the ability to effectively search for models**
 - We are adding a model browser which users can use to search for models based on keywords and their input/output port types
- ▶ **Model reuse requires the management of model libraries to ensure trust**
 - We are adding model libraries, which will facilitate the distribution of models and control who can modify existing models

