Digital Engineering is defined as an integrated digital approach that uses an authoritative source of systems’ data and models as a continuum across disciplines to support lifecycle activities from concept through disposal as defined by the Office of the Deputy Assistant Secretary of Defense Office for Systems Engineering (ODASD(SE)).

Digital Engineering / Model Based Systems Engineering (MBSE) Benefits:
- Accelerated Learning Environment / Knowledge Capture / Speed / Cost – Increase speed of design convergence; reduce associated costs related to unnecessary design iterations, and reduce learning curves
- Capabilities Based Acquisition – Rapid delivery of integrated capabilities
- Sustainment Vision – Predictive, integrated sustainment operations
- Digital Business Operations – Integrated business systems “apps” at the desktop

In today’s Digital Engineering age, there’s no shortage of challenges and opportunities related to weapon system model integration and infrastructure.
• An integrated approach ties data across the lifecycle, and amongst the different engineering domains into a connected digital engineering ecosystem.

• This ecosystem is the interconnected infrastructure, environment, and methodology (process, methods, and tools) used to store, access, analyze, and visualize evolving systems' data and models to address the needs of the stakeholders.

• It enables engineers and stakeholders to collaborate and integrate their efforts to reduce rework and errors throughout the lifecycle.
Today there are many different types of models that can be illustrated in a multitude of modeling languages and tool sets. A model ontology must be developed to define how the models work together to be traceable back to system requirements, and for verification and validation efforts. Starting by defining a classification of models is helpful in selecting the necessary model based on purpose and scope.
• **Physical Model:** Concrete vice abstract (logical/analytical) representation
• **Descriptive Model:** Describes a logical relationship, typically describing a functional or physical architecture of a system.
  – Defines component interconnections/relationships
• **Analytical Model:** Describes a mathematical relationship, such as differential equations that support quantifiable analysis about the system parameters.
  – Used to determine component property values to meet system requirements
• **Domain Specific Models:** The descriptive and analytical models discussed above can also classified into the domain that they represent – i.e. system properties, design implementations, subsystem, system application, etc.
• **System Models:** These models are most likely hybids of both descriptive and analytical models, and can span multiple domains that will need to be integrated (and here in lies the challenge) to ensure a comprehensive and cohesive presentation of the entire system.
  – Specifies components of the system
Standards and Guidelines

• Program standards and guidelines defining model-based business rules and processes must be established and adhered to, to ensure that an attribute in one model has the same meaning in another model.

• This becomes especially important when working in multiple modeling tools. In addition to establishing business rules/processes, a program must also have a means by which to exchange the model data – i.e. file exchange, use of application program interfaces (API), shared repository, etc. not to mention model classification / portion marking, partitioning, etc. considerations.

• The use of modeling standards / guidelines, and model-based business rules and processes are vital components to successfully executing proper integration across multiple domains.
Model Trace / Authoritative Source of Truth

- Working in the Digital Engineering / Model-Based Systems Engineering (MBSE) environment, many different types of models may be developed, as categorized on the previous slide. In addition, there are domain specific models typically also created, for example, component level design, analysis and simulation.

- All of these different models must be integrated in order to have an authoritative source of truth, as well as to fully realize the benefits of working in a model based environment. The different models each represent different facets of the same system. As a result, there must be model integration to ensure that the models are traceable back to the requirements, and ultimately ensuring proper integration for a cohesive system solution.
- Using SW Engineering approach for model management
  - Static analysis
  - Test on commit
  - Frequent commits
- Integrating current capabilities that are disparate
- Using R&D funding to develop needed application capabilities and integrations between tools not available today
- Attempting to use COTS or open source as much as possible
• Multiple System Modeling Language (SysML) tools exist for Systems Engineers to utilize for their associated systems tasking, such as: MagicDraw, GENESYS, Rational Rhapsody, Enterprise Architect, Etc.

• SysML supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems can include hardware, software, information, processes, personnel, and facilities. SysML is an enabling technology for MBSE.

• Some translators do exist for use between the different tools (i.e. import/export through Excel). However much information is lost in this translation.

• The following will detail a model integration example, discussing a solution developed to integrate between MagicDraw and GENESYS models, moving towards a tool agnostic/open source solution in the second phase.
Conversion Process Creation

1. Investigate GENESYS & SysML MetaModels
2. Develop Hypersonics Metamodel
3. Map Required GENESYS & SysML
   - Generate Data for Element Creation
   - Import Data into SysML
4. Parse XML Files
5. Link Data Between Files
6. Generate Data for Element Creation
7. Import Data into SysML
8. Review Import

Updates as Needed

Automate in future with custom No Magic plugin
### Metamodel Example

#### Element Mapping

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Maps to Sys ML Element</th>
<th>Sys ML Mapping Notes</th>
<th>Maps to GENESYS Element</th>
<th>GENESYS Mapping Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System Element</td>
<td>Block</td>
<td></td>
<td>Component</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>System Part</td>
<td>Property</td>
<td>SysML Property must be typed by a block. For MagicDraw, a specific stereotype, &quot; Part Property&quot; is also</td>
<td>Component</td>
<td>Convert GENESYS Component name to lower case.</td>
</tr>
</tbody>
</table>

#### Property Mapping

<table>
<thead>
<tr>
<th>#</th>
<th>Owner</th>
<th>Relationship</th>
<th>Type</th>
<th>Maps to GENESYS Property</th>
<th>GENESYS Property Source</th>
<th>GENESYS Mapping Notes</th>
<th>Maps to Sys ML Property</th>
<th>Sys ML Property Source</th>
<th>Sys ML Mapping Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System Part</td>
<td>has type</td>
<td>System Element</td>
<td>Component</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Property</td>
</tr>
<tr>
<td>2</td>
<td>System Part</td>
<td>allocated from</td>
<td>Function</td>
<td>+performs : Function</td>
<td>Component</td>
<td></td>
<td></td>
<td></td>
<td>Allocate</td>
</tr>
<tr>
<td>3</td>
<td>System Part</td>
<td>is Element of</td>
<td>System Element</td>
<td>-builtin : Component</td>
<td>Component</td>
<td></td>
<td></td>
<td></td>
<td>Property</td>
</tr>
<tr>
<td>4</td>
<td>System Element</td>
<td>contains</td>
<td>System Part</td>
<td>+built from : Component</td>
<td>Component</td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
</tr>
</tbody>
</table>
General Structure Conversion Example

GENSYS
Fast Food System
10 Components

Part I - Component List

- C - System Context
- C1 - Customer
- SYS - Fast Food System
- SYS1 - Building
- SYS1.1 - Bathrooms
- SYS1.2 - Eating Area
- SYS1.3 - Kitchen
- SYS1.4 - Service Area
- SYS2 - Parking Lot
- SYS3 - Staff
- SYS3.1 - Cleanup Crew
- SYS3.2 - Cooks
- SYS3.3 - Service Staff

MagicDraw
Imported Fast Food System
- Most entities treated as blocks
- Processed “built from” relationship
Functional Flow Block Diagram Conversion Ex

GENSYS FFS Activity Diagram

- Implemented Node Creation
- Implemented Control Flow connections

MagicDraw FFS Activity Diagram
<table>
<thead>
<tr>
<th>Aspect</th>
<th>Metamodel</th>
<th>GENESYS Mapping</th>
<th>SysML Mapping</th>
<th>Import &amp; Review Unclass</th>
<th>Import &amp; Review Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Basic Structure</td>
<td>In Progress</td>
<td>Complete</td>
<td>Complete</td>
<td></td>
<td>Complete</td>
</tr>
<tr>
<td>Value Properties</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>FFBD Control Flows</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>FFBD Item Flows</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Interfaces</td>
<td>Complete</td>
<td>Complete</td>
<td>In Progress</td>
<td>Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Parametrics</td>
<td>On Deck</td>
<td></td>
<td>On Deck</td>
<td>On Deck</td>
<td>On Deck</td>
</tr>
<tr>
<td>Use Cases</td>
<td>On Deck</td>
<td></td>
<td>On Deck</td>
<td>On Deck</td>
<td>On Deck</td>
</tr>
<tr>
<td>Sequences</td>
<td>On Deck</td>
<td></td>
<td>On Deck</td>
<td>On Deck</td>
<td>On Deck</td>
</tr>
<tr>
<td>State Machine</td>
<td>On Deck</td>
<td></td>
<td>On Deck</td>
<td>On Deck</td>
<td>On Deck</td>
</tr>
</tbody>
</table>
**Deliverables:**
- Process for integrating models between MagicDraw and GENESYS
- Documentation on the process to complete the integration

**Accomplishments to Date:**
- MagicDraw chosen as Tool of Choice
- Partial Mapping of GENESYS ontology to SysML
- Model Integration Methods Evaluated
  - Export data form GENESYS for import into MagicDraw
  - Extract data directly from GENESYS model file
- Evaluated relationships within GENESYS model files
- Evaluated functional control flow constructs in GENESYS model file
- Extracted data manually from GENESYS and used a relational database to export data
- Manually imported data into MagicDraw using the CSV Import Plugin
- Developing a MagicDraw Plugin to read the GENESYS model file and create elements in MagicDraw via JAVA API calls
  - Created a way to read and query GENESYS data from a GENESYS model file
  - Converted data packages, blocks, model items, system parts, item parts, generalizations, value types, part properties, interface if blocks, proxy ports, hierarchy for interfaces, links, requirements, activities, and activity diagrams with call behavior actions, control flows and item flows

**Lessons Learned:**
- Creating MagicDraw models from the GENESYS model's export file using the MagicDraw API's methods
- Created and tested how the MagicDraw Plugin will be created which will enable a distributable solution
- Improved Plugin development time by using Eclipse built in debugger to reload java classes instead of restarting the entire program
- Created a collaborative development environment using GitForWindows with a shared folder on the network