

Weapon System Model Integration

Rachael Wiseman



CAPT Mark Oesterreich, USN
Commanding Officer



Dr. Brett Seidle, SES
Technical Director

Digital Eng → Weapon System Model Integration

- **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.



Current: Stove-piped models and data sources

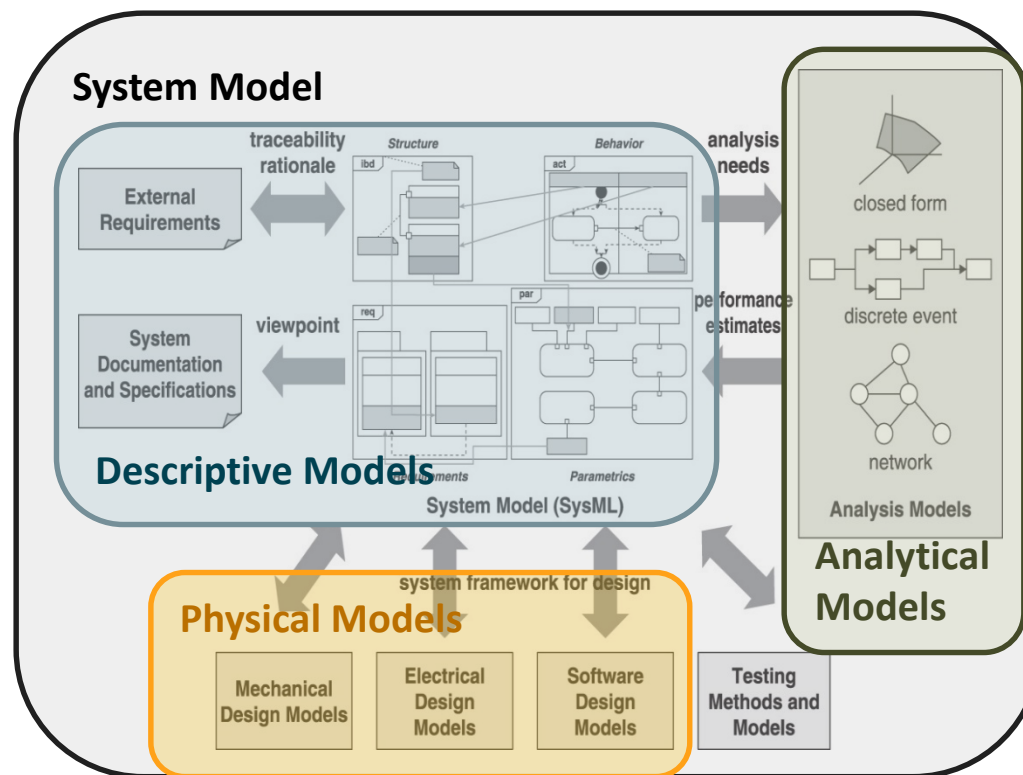
Future: Digital Engineering Ecosystem

Statement A: Approved for Public Release; Distribution Unlimited

Utilizing An Integrated Approach

- **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.



Reference: A Practical Guide to SysML: The Systems Modeling Language

Statement A: Approved for Public Release; Distribution Unlimited

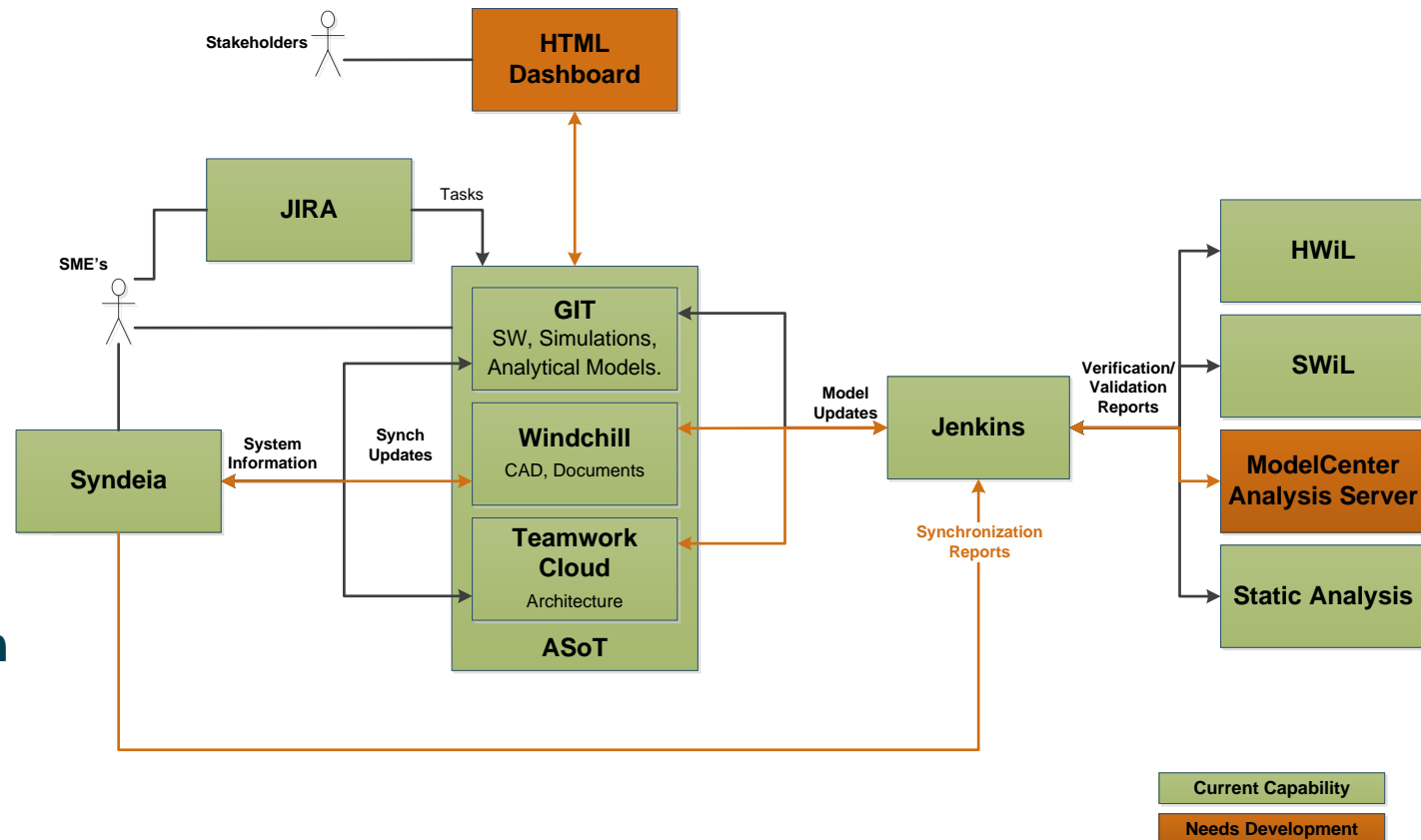
- **Physical Model:** Concrete vice abstract (logical/analytical) representation
- **Descriptive Model:** Describes a logical relationship, typically describing a functional or physical architecture of a system.
 - Defines component interconnections/relationships
- **Analytical Model:** Describes a mathematical 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 hybrids 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

- 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.

- **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.**

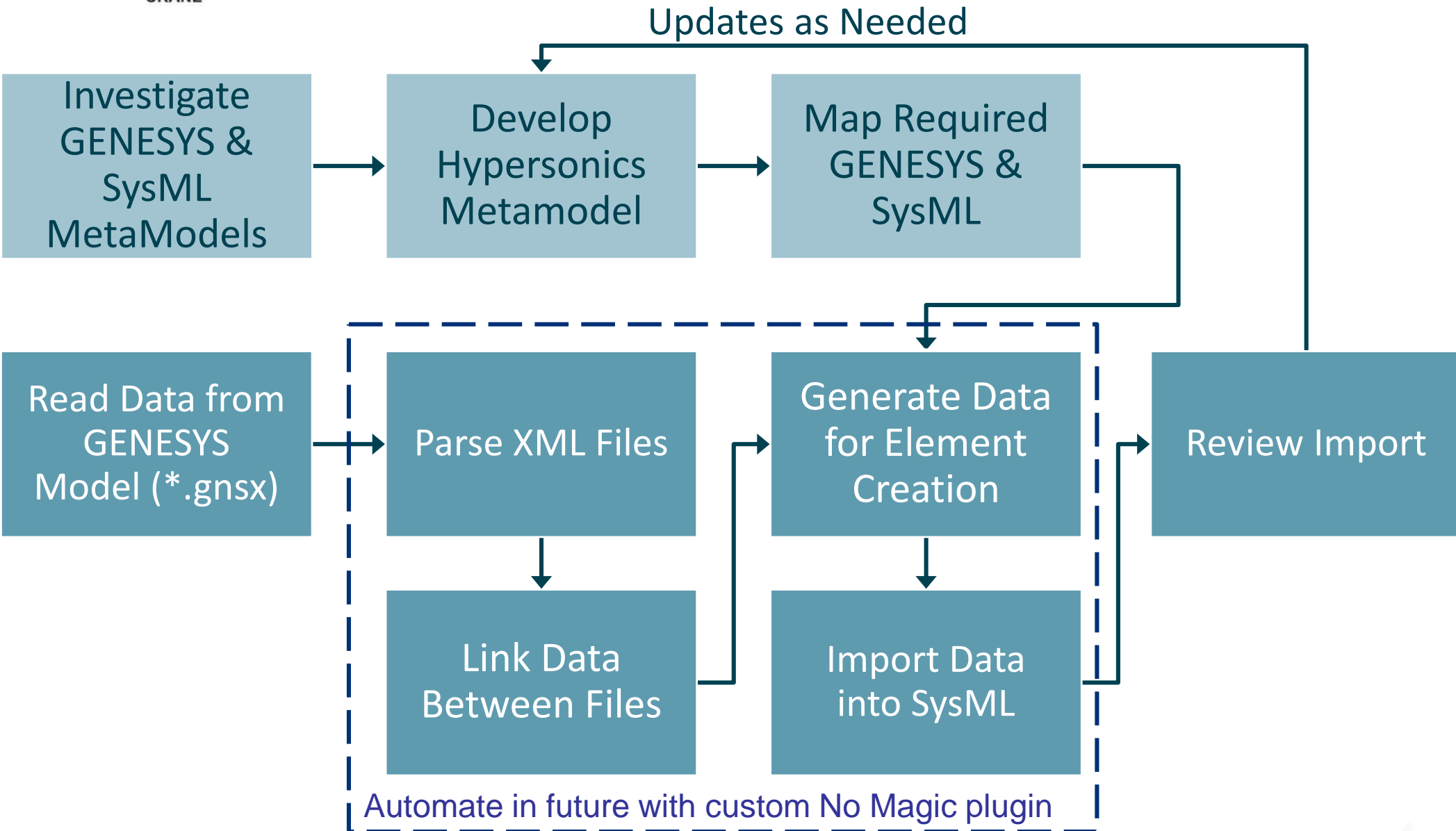
Model Development Environment

- 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



• Element Mapping

#	Name	Maps to Sys ML Element	Sys ML Mapping Notes	Maps to GENESYS Element	GENESYS Mapping Notes
1	 System Element	 Block		 Component	
2	 System Part	 Property	SysML Property must be typed by a block. For MagicDraw, a specific stereotype, "Part Property" is also	 Component	Convert GENESYS Component name to lower case.

• Property Mapping

#	Owner	Relationship	Type	Maps to GENESYS Property	GENESYS Property Source	GENESYS Mapping Notes	Maps to Sys ML Property	Sys ML Property Source	Sys ML Mapping Notes
1	 System Part	has type	 System Element	 Component		Because GENESYS is not object oriented, there is not a directly translatable property in GENESYS to the Ontology. Instead, this property of a System Part, which traces to a GENESYS component with the components name converted to all lower case, will point to the System Element created as a direct trace of a GENESYS component.	 type : Type [0..1]	 Property	
2	 System Part	allocated from	 Function	 +performs : Function	 Component		 client : NamedElement [1..*]	 Allocate	Only from Allocates with Supplier as the System Part.
3	 System Part	is element of	 System Element	 -built in : Component	 Component		 /owner : Element [0..1]	 Property	
4	 System Element	contains	 System Part	 -built from : Component	 Component		 /part : Property [0..*]	 Block	Specifically typed by a block. For MagicDraw, a specific stereotype, "Part Property" is also used.

GENSYS Fast Food System 10 Components

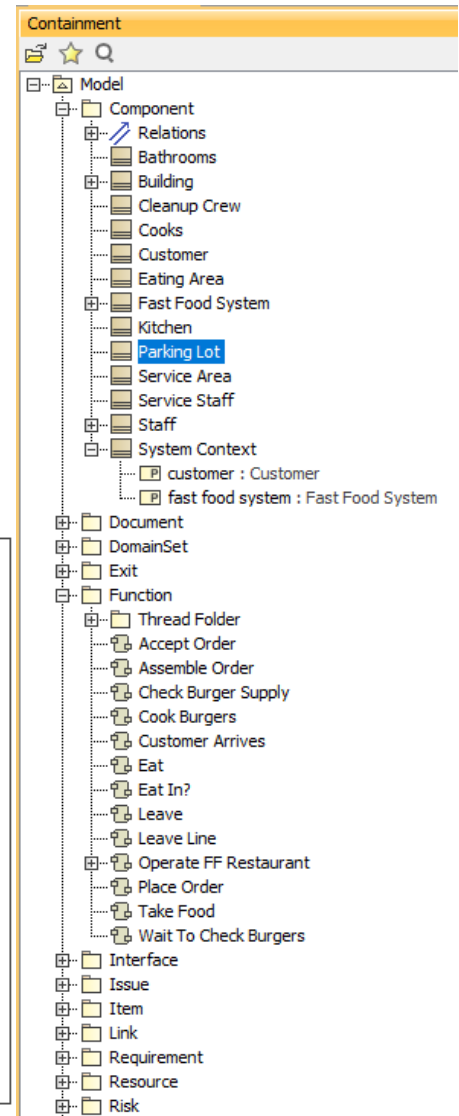
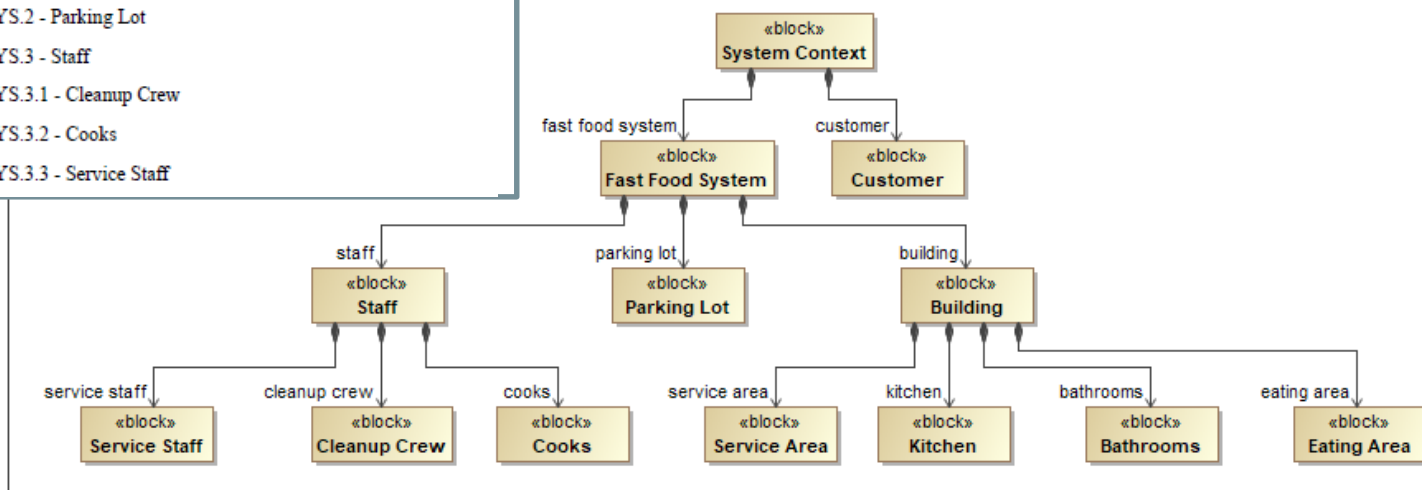
Part I - Component List

C - System Context
C.1 - Customer
SYS - Fast Food System
SYS.1 - Building
SYS.1.1 - Bathrooms
SYS.1.2 - Eating Area
SYS.1.3 - Kitchen
SYS.1.4 - Service Area
SYS.2 - Parking Lot
SYS.3 - Staff
SYS.3.1 - Cleanup Crew
SYS.3.2 - Cooks
SYS.3.3 - 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

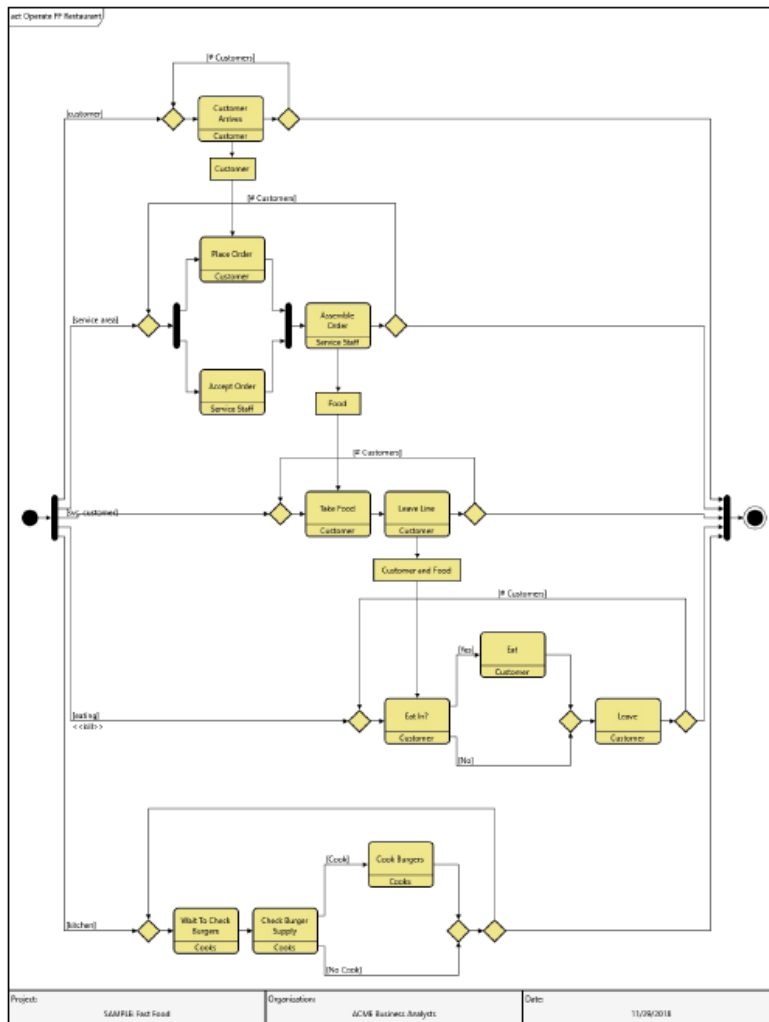
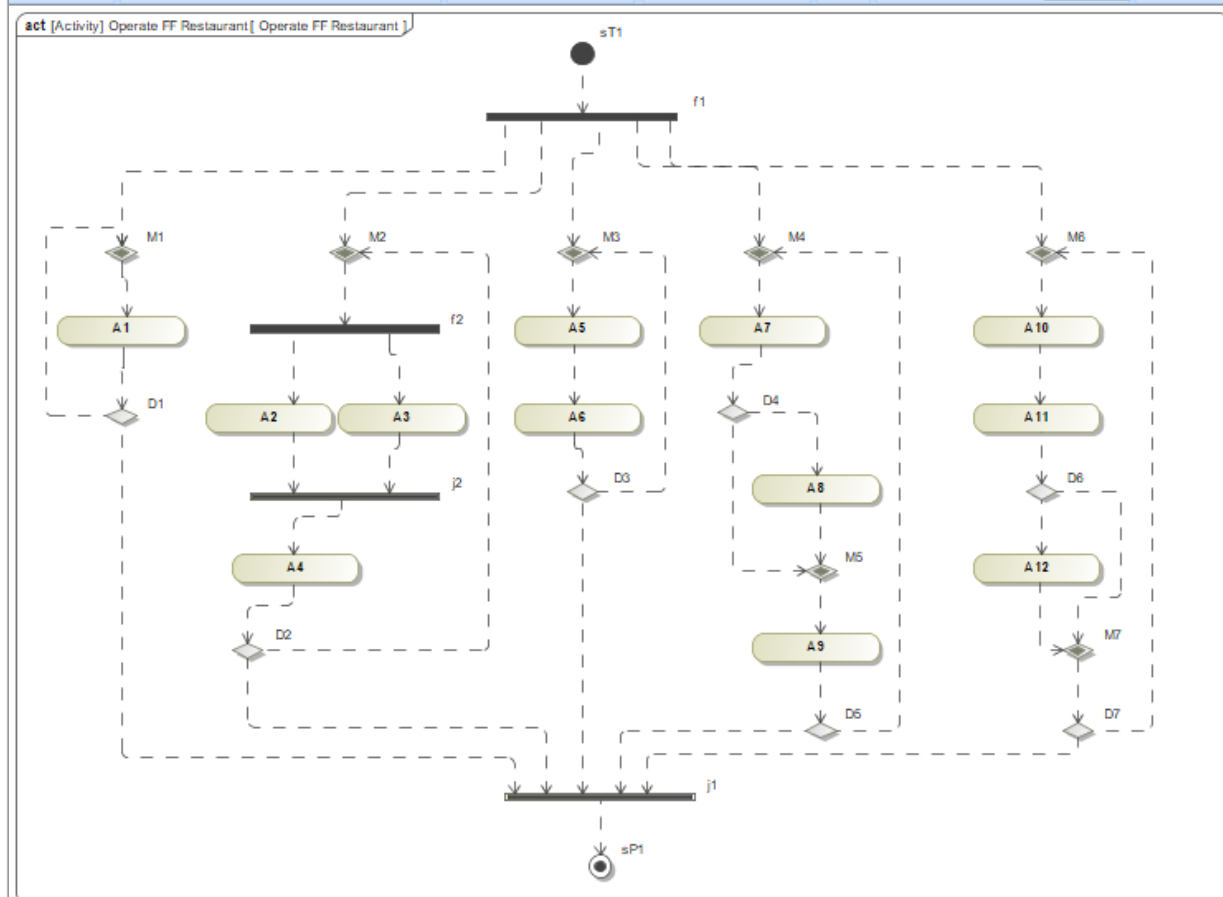


Figure 28. Operate FF Restaurant, Activity Diagram

MagicDraw FFS Activity Diagram

- Implemented Node Creation
- Implemented Control Flow connections



Conversion Process Status

Aspect	Metamodel	GENESYS Mapping	SysML Mapping	Import & Review Unclass	Import & Review Class
Requirements					
Basic Structure					
Value Properties					
FFBD Control Flows					
FFBD Item Flows					
Interfaces					
Parametrics					
Use Cases					
Sequences					
State Machine					
		Complete	In Progress	On Deck	To Do

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 from 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