

Developing a System Engineering Knowledge Partner

Presented by Anthony (Tony) Komar

Disclaimer:

The views and opinions expressed in this presentation are those of the presenter and do not necessarily reflect the official policy or position of Siemens. While Siemens has supported the presenter's participation in this event, the content presented does not imply any commitment by Siemens to deliver specific capabilities or future products. This presentation is for informational purposes only.

Agenda

Part 1 :

- **Current state**
 - **General-Purpose Large Language Models with SysML and SysML V2**

Part 2:

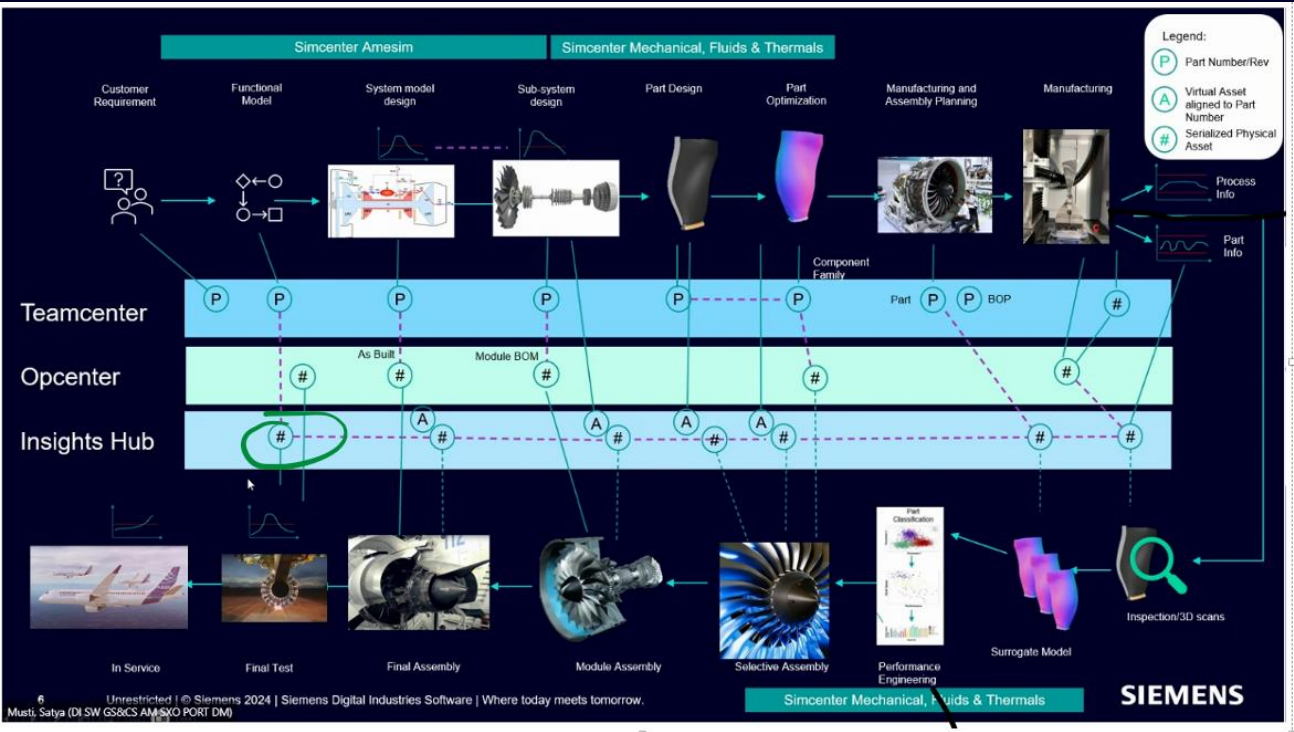
- **Retrieval Augmented Generation Notebook Knowledge Partner based on Capella**

Part 3:

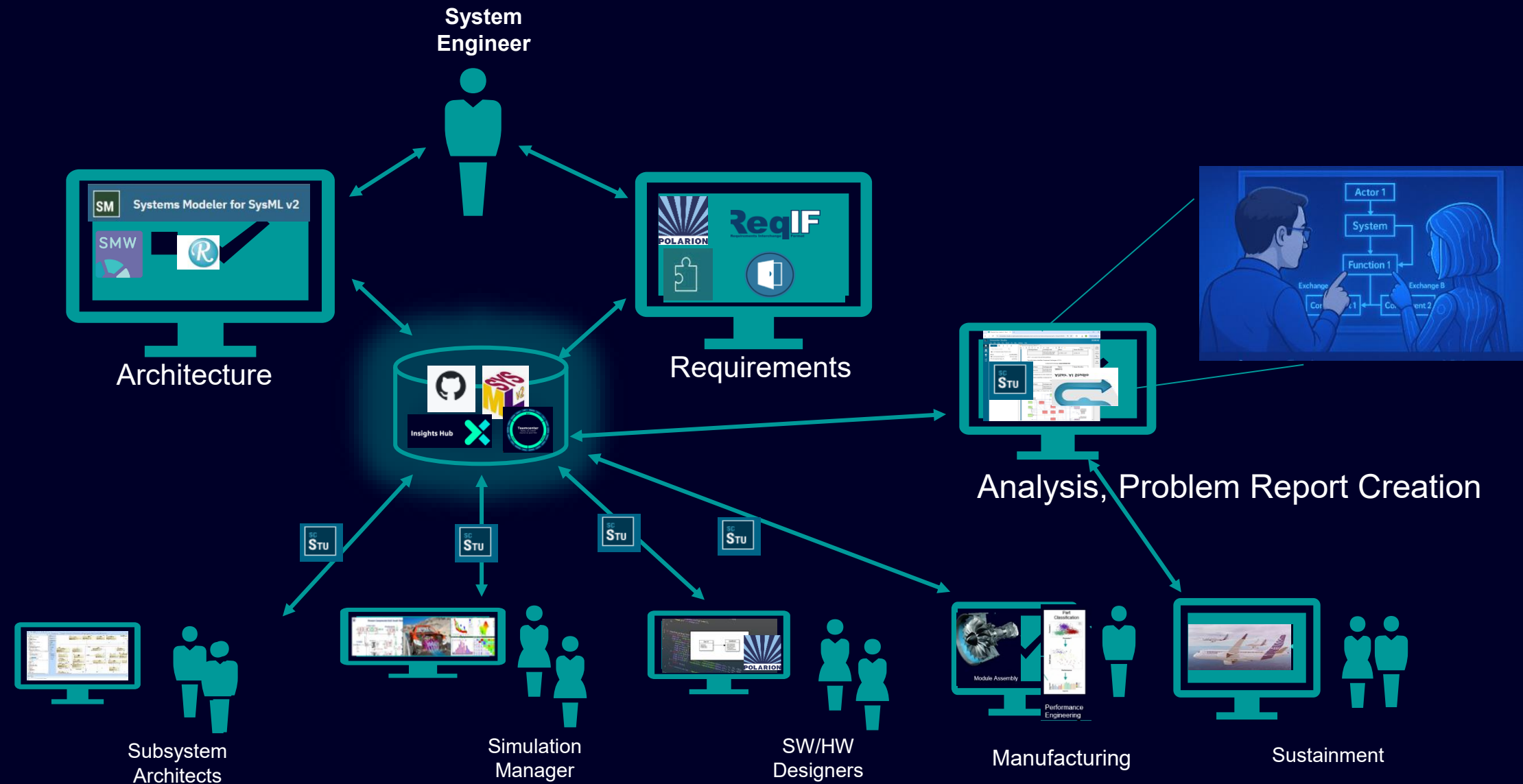
- **Developing Agent based Knowledge Partner leveraging a Capella model.**
 - **Capabilities**
 - **Architecture**
 - **Demo**

First this.. Setting the Stage

Discuss with Satya why an Agent Based Solution for Sustainment. People wanting to use the model without intimate knowledge.



System Engineering Knowledge Partner role in Sustainment System Engineering



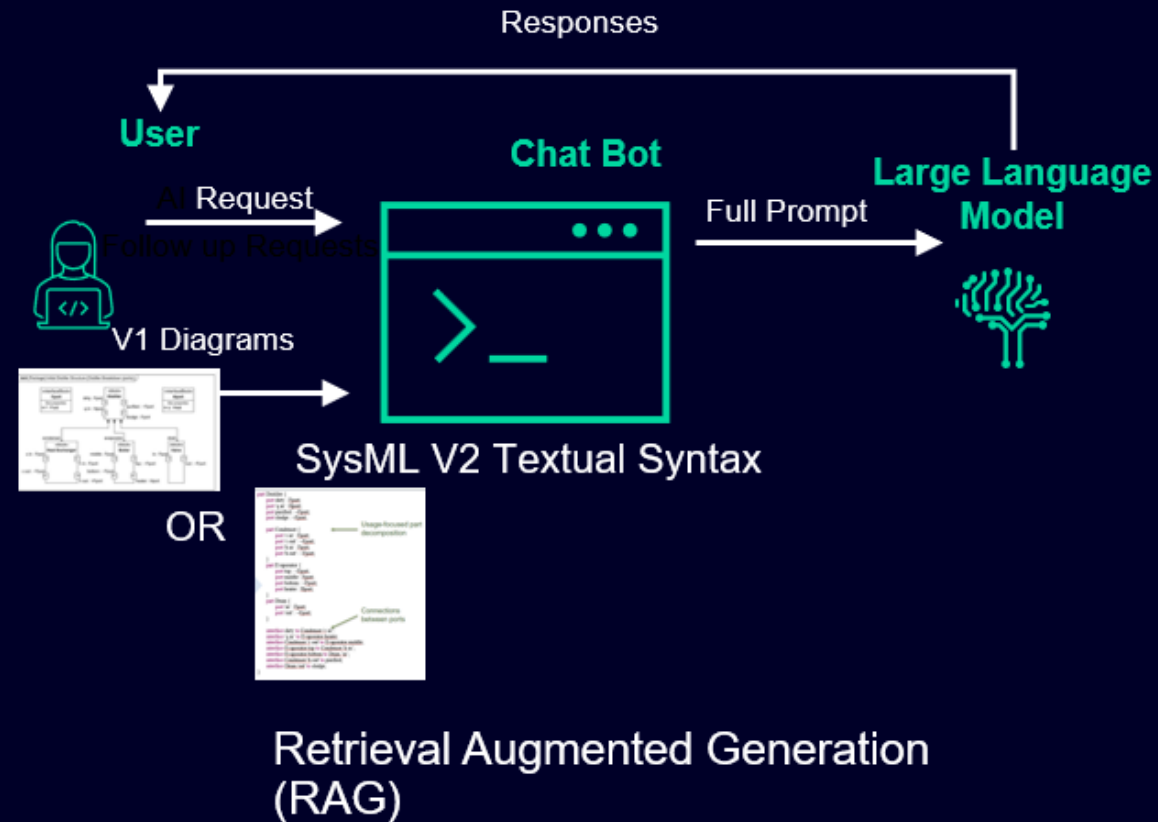
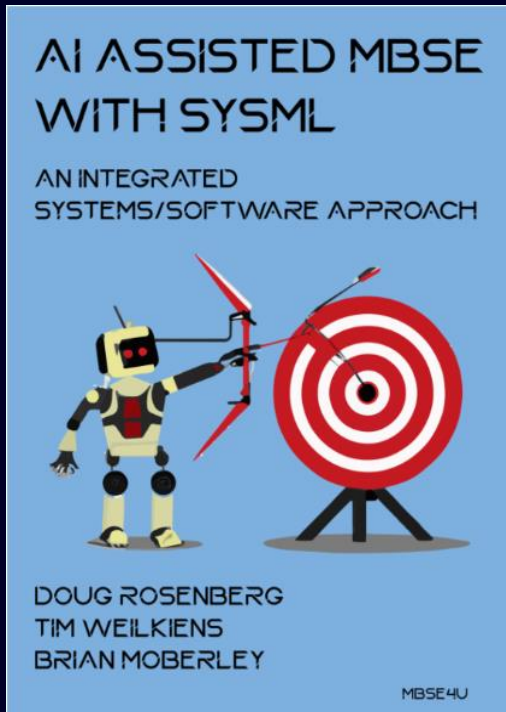
Part 1

Current state of General-Purpose Large Language Models with System Models

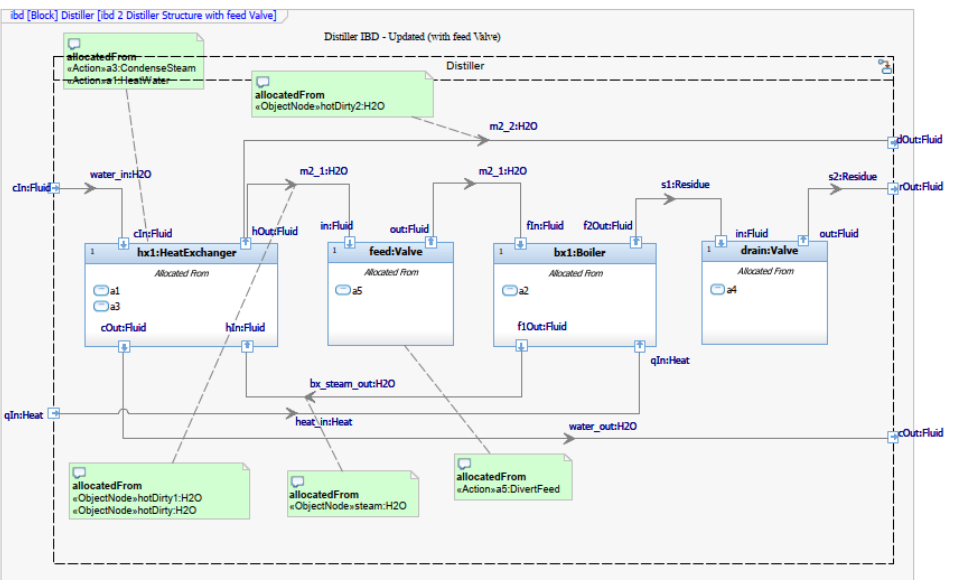
AI Explosion on the scene and its impact

General Purpose Large Language Models

- Can write and read code.
- Can read Sysml v1 diagrams
- Can reason on SysML V2 textual Syntax
- Can create SysML V2



Example: Analysis of SysML V1 Distiller Diagram.



Your prompt: Perform and analysis on the attached .pdf file. It is using SysML V1 Notation. Describe in detail the content of the IBD diagram.

IBD Diagram Analysis for "Distiller Structure with Feed Valve"

Overview of the Diagram

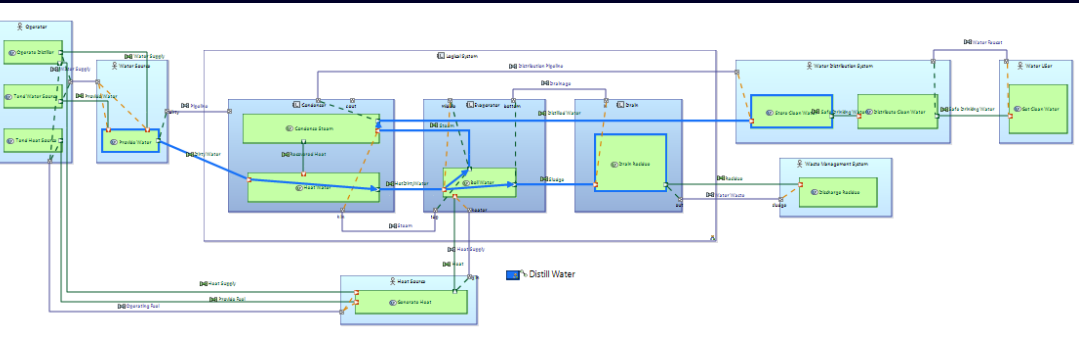
The Internal Block Diagram (IBD) provided depicts the structure and interactions within the distiller system, enhanced with a feed valve. This diagram is drawn using SysML V1 notation and highlights the internal connectivity between system components such as valves, heat exchangers, boilers, and fluid flows. Each component in the diagram holds specific functions crucial for the distillation process, and its interconnections are methodically designed to support the flow of water, residue, heat, and steam throughout the system.

Core Components and Interactions

- **Distiller:** Central to the system that assimilates various components including different types of valves, fluid connections, and processing elements.
- **Valves - drain & feed:**
 - Drain Valve 1 facilitates the egress of residues and processed fluids out of the system.
 - Feed Valve 1 serves as an entry point for source water into the distilling system.
- **Boiler 1 (bx1):** Allocated for the action 'a2', this component is integral in boiling the water to generate steam (H2O bx_steam_out).
- **Heat Exchanger 1 (hx1):** Utilized for 'a1: HeatWater' and 'a3: CondenseSteam' actions to facilitate heat transfer processes within the system. It manages incoming and outgoing fluids to maximize heat efficiency.
- **Fluid Ports & Connections:** The IBD highlights various fluid connections:
 - *f1Out*: Outflow for fluids, indicating treated water or by-products forwarded to a particular location or stage.
 - *f2Out*, *fIn*: Indicate bidirectional fluid exchanges within certain elements of the distiller.
 - *cOut*, *cIn*, *hIn*, *hOut*: Fluid ports on heat exchanger for cooling (c) and heating (h) purposes, ensuring optimal fluid transition and thermal regulation.

Example: Analyzing SysML V2 is also possible with this approach

Capella Distiller Project



Siemens SMW to SysML V2 Converter



Sysml V2

Library : ARCADIA
File: Distiller Project

OA – SA – LA



File: Distiller Project
Logical Architecture
LA

Using the attached SysML V2 files, ARCADIA.sysml and Project Logical Architecture.sysml. Use ARCADIA.txt as a library that defines the terms of ARCADIA Domain found in the Project Logical Architecture.txt. Perform analysis on the Logical Component Evaporator and display the result as .html and specifically provide insights on:

The purpose of the component.

1. Any components that make it up.
2. A list of allocated functions and their potential roles.
3. How the ports relate to the functions.
4. Any traceability artifacts linked to the component and the artifacts url.
5. Suggestions for improving the component description.
6. Create a tabular table with columns for the component and its ports and the port interfaces, the interface source and the interface target.
7. List any property values and the related model element.

Note: The domain terms of ARCADIA can be used on SysML V2 due to Library

SysML V2 Analysis no training – with a Domain Specific ARCADIA library.

Using the attached SysML V2 files, ARCADIA.sysml and Project Logical Architecture.sysml. Use ARCADIA.txt as a library that defines the terms of ARCADIA Domain found in the Project Logical Architecture.txt. Perform analysis on the Logical Component Evaporator and display the result as .html and specifically provide insights on:

The purpose of the component.

- Any components that make it up.
- A list of allocated functions and their potential roles.
- How the ports relate to the functions.
- Any traceability artifacts linked to the component and the artifacts url.
- Suggestions for improving the component description.
- Create a tabular table with columns for the component and its ports and the port interfaces, the interface source and the interface target.
- List any property values and the related model element.

Analysis of Logical Component: Evaporator

1. Purpose of the Component

The "Evaporator" is a logical component in the distillation process, primarily involved in heating water to its boiling point to facilitate steam generation. It acts as a critical element in the water distillation process by supporting the transition from liquid water to vapor.

2. Components That Make It Up

The Evaporator consists of various ports that allow it to interact with other components in the system, facilitating fluid and heat interactions necessary for the distillation process. It does not have sub-components that make it up but is a part of a larger logical architecture in a distillation system.

3. Allocated Functions and Their Potential Roles

- Boil Water:** The primary function of the Evaporator. It transforms input water into steam.

4. Port Relations to Functions

- Port 'bottom':** Connected to the function output port FOP 1 of the "Boil Water" logical function. Responsible for expelling residues/waste.
- Port 'top':** Connected to the function output port FOP 2 of the "Boil Water" logical function. It emits generated steam towards the Condenser component.
- Port 'middle':** Serves as an input port, receiving hot water for further boiling.
- Port 'heater':** Provides inlet for heating mediums, enabling the heating process in the Evaporator.

5. Traceability Artifacts

- No specific traceability artifacts with URLs are associated with the Evaporator from the provided document.

6. Suggestions for Improving the Component Description

To enhance clarity, the component's description should include operational constraints, performance metrics, such as efficiency and throughput rate, and the specific range of temperatures it supports. This specific context is invaluable for engineers integrating and maintaining the component in complex systems.

7. Port Table

Component	Port	Interface	Interface Source	Interface Target
Evaporator	bottom	Drainage	Evaporator	Drain
Evaporator	top	Steam	Evaporator	Condenser
Evaporator	middle	Hot Water	Diverter Assembly	Evaporator
Evaporator	heater	Heat Supply	Heat Source	Evaporator

8. Property Values and Related Model Element

The Evaporator component doesn't contain additional specified property values outside of its assigned roles and interactions within the logical architecture.

Part 2

Retrieval Augmented Generation Notebook Knowledge Partner based on Capella

Pugh Analysis of Options Led to Capella Based Architecture

Desired Capability	SysML V2	SysML V1	Capella/ARCADIA
Provide the LLM a subset of content relevant for the analysis being requested. (keep tokens down)	0 <i>No Semantic Aware APIs.</i>	— <i>Diagrams are used to scope analysis</i>	+ <i>Rich set of Semantic aware APIs *</i>
Provide a static representation of model that can be easily understood	0 <i>Work on V2 Textual or Abstract Syntax</i>	— <i>Diagrams are the only consistent representation</i>	+ <i>Public APIs can be used to create static representation *</i>
Use a well know method, that both the Large Language Model and the System Engineer both understand.	0 <i>ARCADIA support for libraries just coming on-line (Siemens/OBEO)</i>	— <i>Profiles enhance diagrams</i>	+ <i>ARCADIA user base exists and is known to LLMs</i>

The RAG solution built on Simcenter Studio/ Jupyter LAB

```
from IPython import display as diag_display
resources = {
    "Mountain_Bike": "Mountain_Bike/Mountain Bike",
}
```

LAB diagram

Select: [LAB] Braking (Diagram)
[LAB] Comfort (Diagram)
[LAB] Powering and Shifting (Diagram)
[LFD] Root Logical Function (Diagram)
[LAB] Mountain Bike All (Messy) (Diagram)
[LAB] Mountain Bike and ACME (Messy) (Diagram)
[LAB] Terrain Handling Steering (Diagram)
[MCS] Capabilities (Diagram)
[LDFB] Ride Trail (Diagram)

Submit Selection
Reset

Selected Object Details:
Name: [LAB] Mountain Bike All (Messy)
Type: Diagram
Phase: Logical Architecture LA
Source Component:
Target Component:
Waiting for selection...

The architecture is analyzed by ChatGPT with following prompt.

Please analyze the VAM, file and display the result as HTML specifically provide insights on:
1. The purpose of the diagram based on its related nodes.
2. Describe what the nodes represent in the ARCADIA method and what the diagram is describing. Please format the analysis in HTML suitable for Jupyter Notebook display operation.
OpenAI API Key retrieved successfully.

ChatGPT Analysis:
HTML

Analysis of System Model Diagram

1. Purpose of the Diagram

The diagram is titled "LAB Mountain Bike All (Messy)" and is of the type "Diagram". It represents the complex interactions and relationships within the system of a mountain bike. This includes its various components and their respective roles or functions within the mountain biking process. The primary goal of this model visualization is to depict the mountain bike's functionality, how it interacts with its environment (specifically rugged terrain), and how different components support the overall biking activity.
The related nodes of the diagram point out the various sub-functions and components that interact to make a mountain bike operational. These nodes represent key elements of the overall system involved in ensuring effective bicycle operation and rider satisfaction. This includes the interaction with environmental factors, handling, and comfort provided to the user.

Your prompt: Provide a summary of the functional chains for the mountain bike.

Generating a response...

HTML

Summary of Functional Chains for the Mountain Bike

1. Stopping

The **Stopping** functional chain is focused on the actions and mechanisms involved in bringing the mountain bike to a halt. This encompasses:
• Provide Stopping (Logical Function)
• Split 14 (Logical Function)
• Transfer Propulsion and Stopping Force (Logical Function)
• Provide Friction to Terrain (Logical Function)
• Brake Controls (Functional Exchange)

2. Powering

The **Powering** functional chain emphasizes the efficient transfer and utilization of energy required to propel the mountain bike forward, including functions such as:
• Provide Friction to Terrain (Logical Function)
• Transfer Propulsion and Stopping Force (Logical Function)
• Convert Pedal Force to Wheels (Logical Function)
• Functional Exchanges related to Force (Functional Exchange)

Type your prompt...

Execute Exit

Load file:

Waiting for chat interactions...

Identifies the project for analysis

Prompts for model content for analysis

Creates structured content based on model content and related data

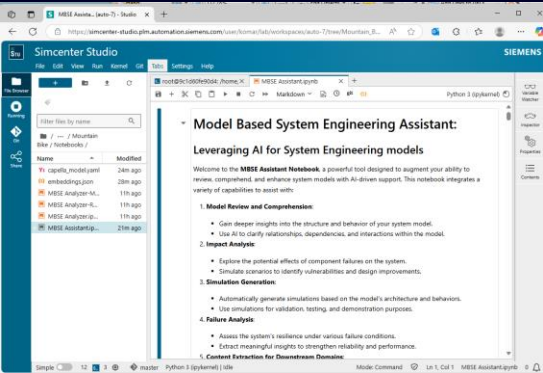
Displays the content

Execute default AI prompts based on content selected

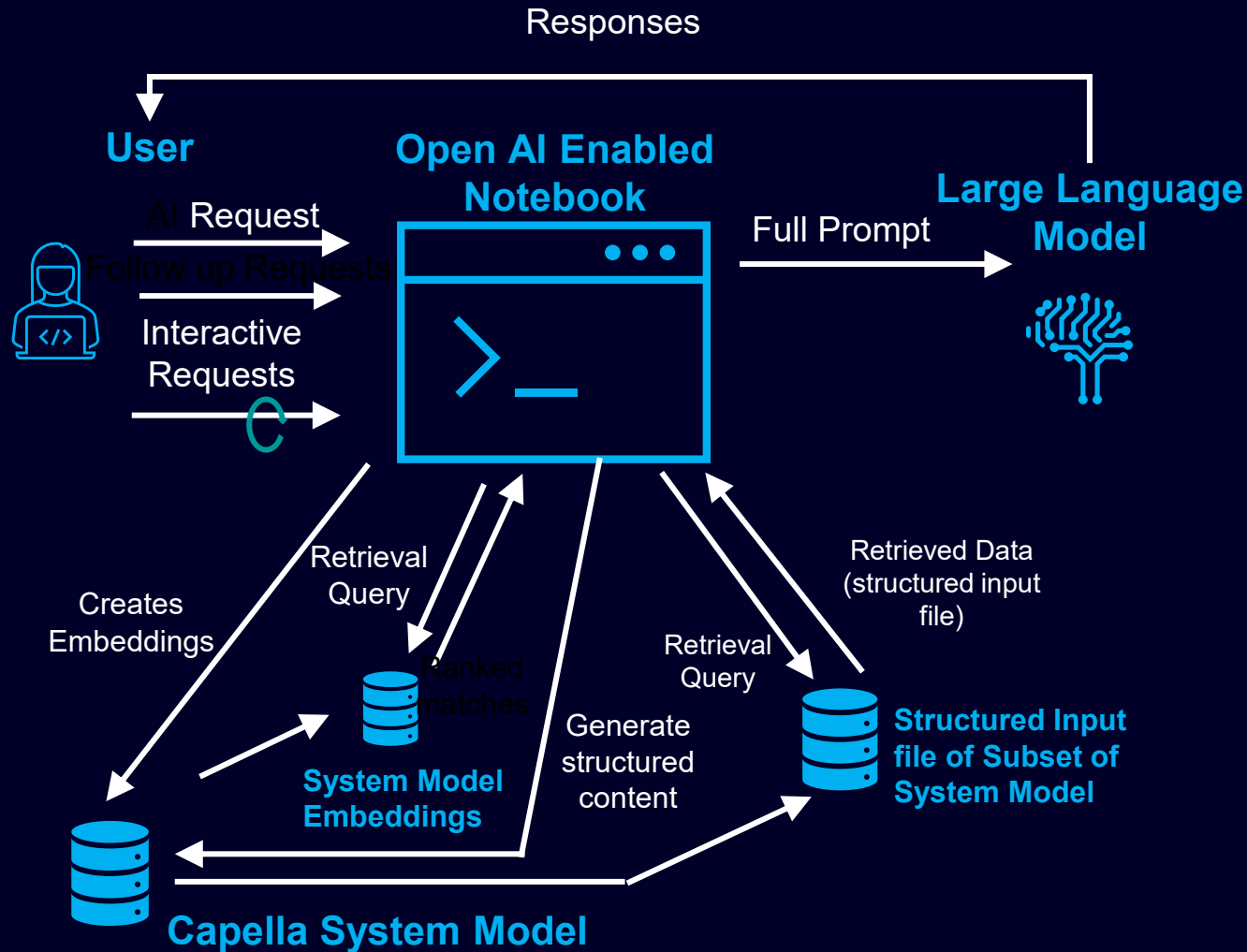
Executes follow-on tailored prompts

Interactive chat on model content

<https://www.youtube.com/@innovatingwithcapella>



Details of RAG based notebook solution



Notebook

- Model converted to embeddings
- User interacts with prompt that leverages embedding to locate subset of model.
- Subset of model converted to yaml “structured input”
- User executes pre-defined and user defined prompts on “structured input”
- Utilizes the capabilities of LLMs (Open AI 4o)
- Provides interactive experience

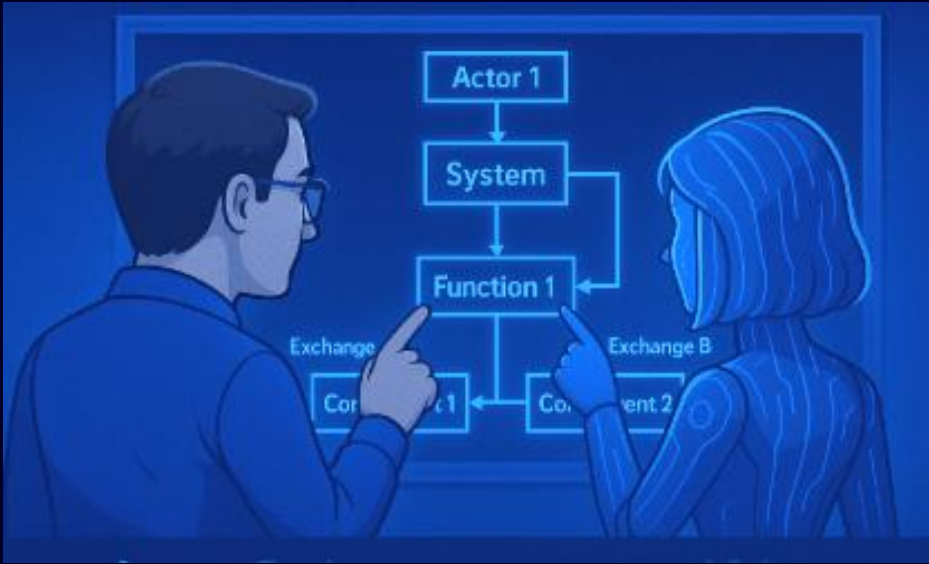
Embeddings

- Allows search like capability to find model content
- Does not require precision of “code”

Structured Input File

- Establishes guardrails and grounds responses

The result is System Engineering Knowledge Partner via Notebook



Deep knowledge in System Engineering domain concepts

- Languages: V1, V2 (growing)
- Methods - ARCADIA, UAF OOSEM

Peer in invention

- Vast knowledge on Problem Domains

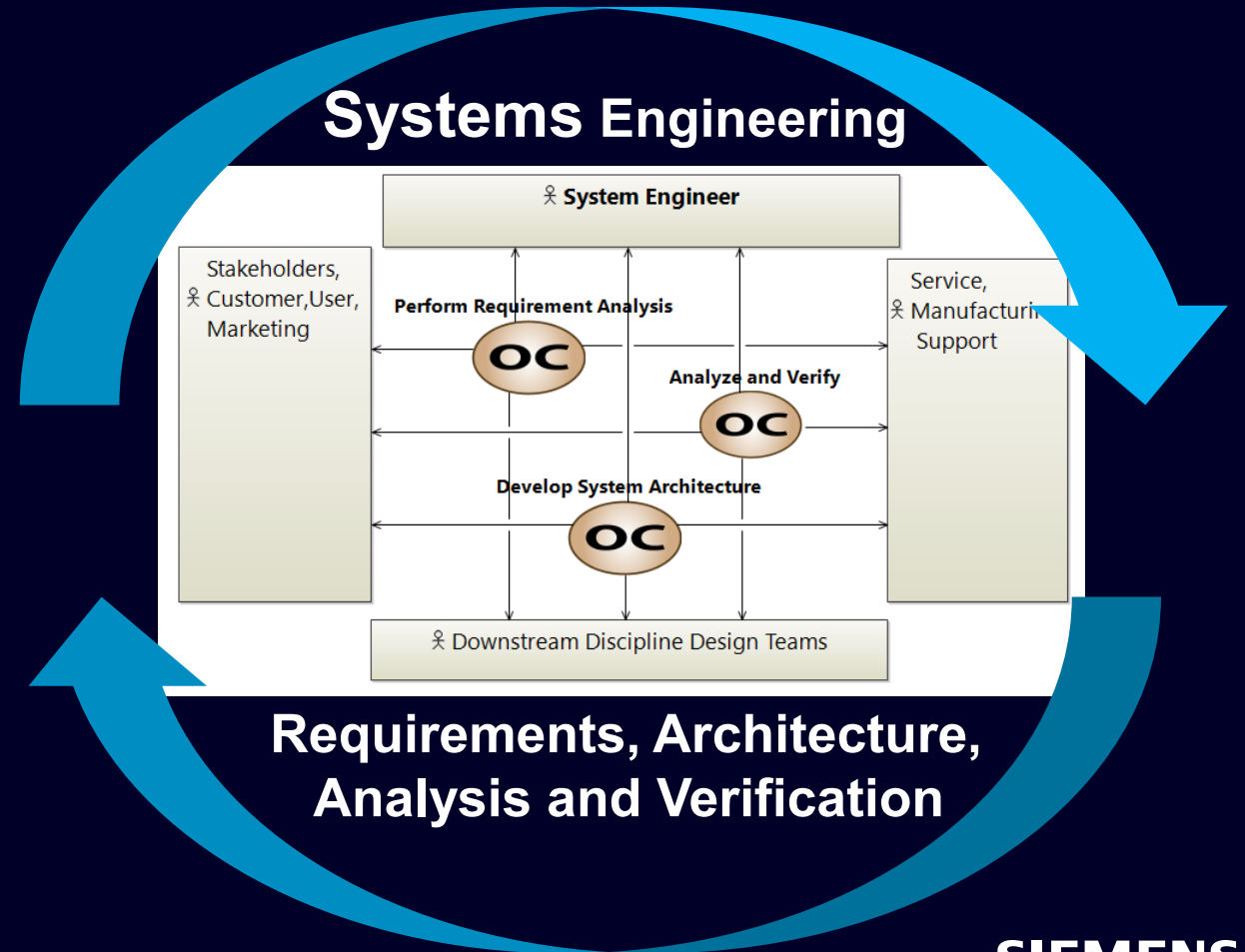
An assistant in analysis

- Extensive analytical skills for impact analysis

A peer in communicating to downstream users

- Capable in variety of languages and notations.
- Skilled in translation engineering domains.
- Deep analytical ability.

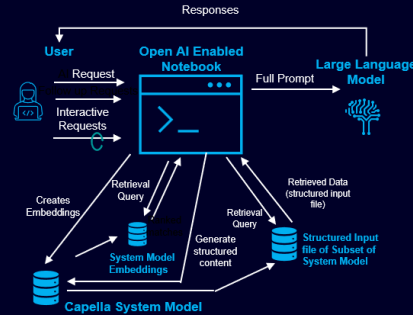
A peer in Verifying the Model and Requirements



Part 3

Agent based Knowledge Partner leveraging a Capella Model

RAG and Notebook Approach advantages and limitations as a Knowledge Partner



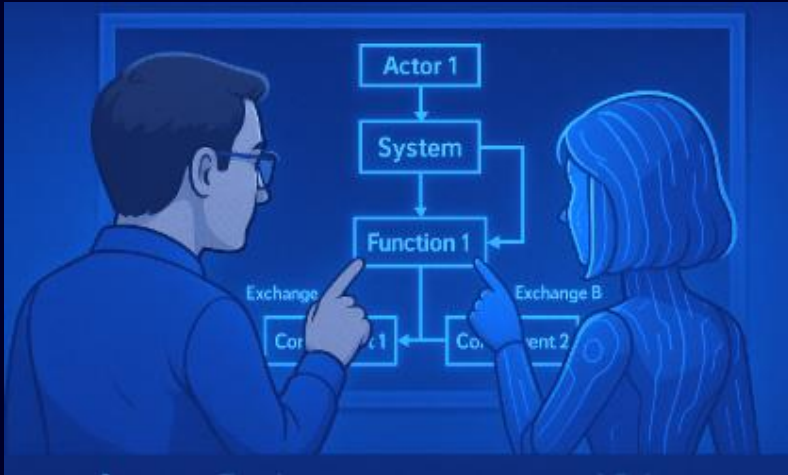
Advantages

- Notebook could be easily replayed as model matures.
- Jupyter Notebooks and Python are widely available to support AI and System Engineering work.
- The notebook could be used to bridge across tools
 - Requirements
 - PLM
 - Risk Management

Limitations

- Jupyter notebook python required for advanced use cases.
- Code is frequently required for common activities of extracting and loading data.
- Code required to interact with other systems related to System Engineering.
 - Requirements
 - PLM
 - Risk Management
- Adoption Curve

Agentic Technology as a Solution to Knowledge Partner

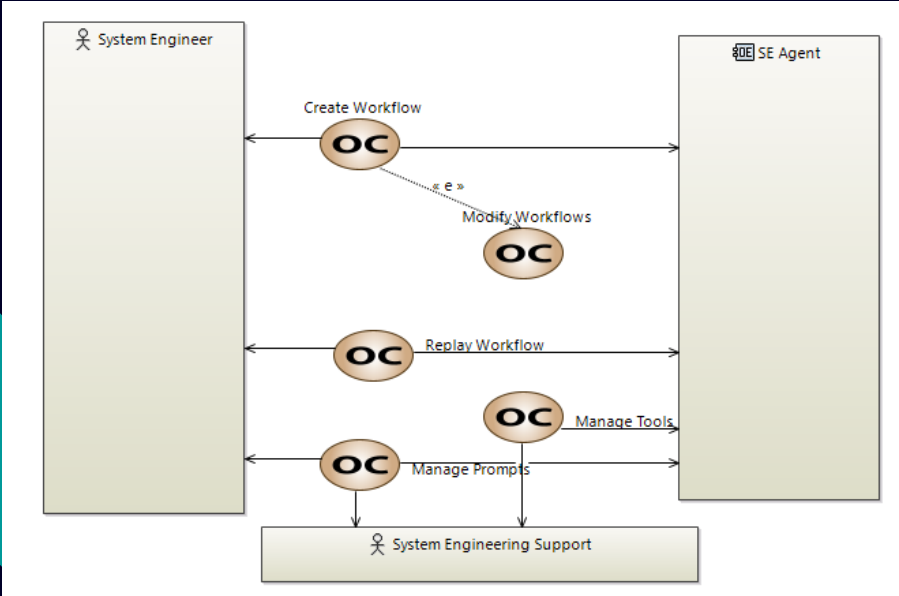
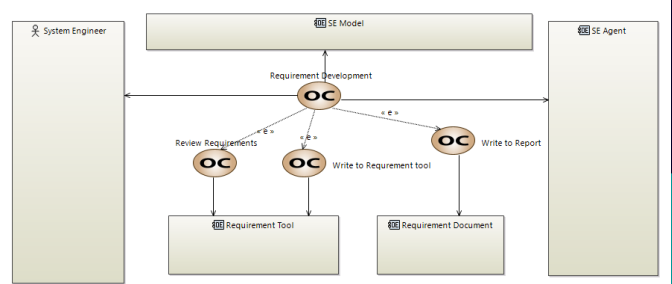


- **Agents can assist users**
- **They can guide and facilitate use.**
- **Agents can act autonomously.**
- **They can make decisions or take actions without constant human instruction.**
- **They integrate multiple tools or APIs.**
- **Agents can call external systems—search, databases, models, or scripts to perform a task..**
- **They can collaborate with humans or other agents.**
- **Multi-agent systems share tasks or negotiate outcomes.**
- **They can support for repetitive or knowledge-intensive tasks.**
- **Examples include research, summarization, analysis, and automation.**

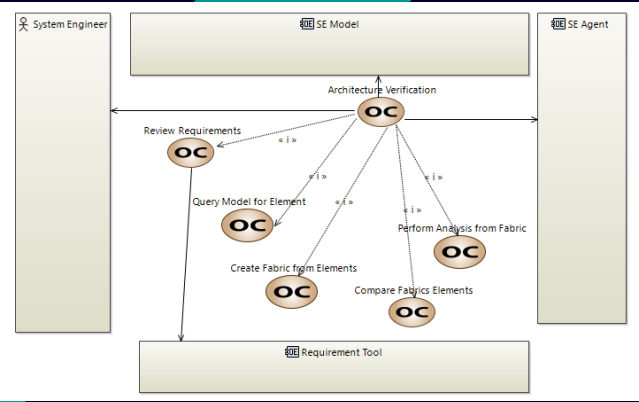
Designing an SE Agent started with Operational Capabilities

Workflows

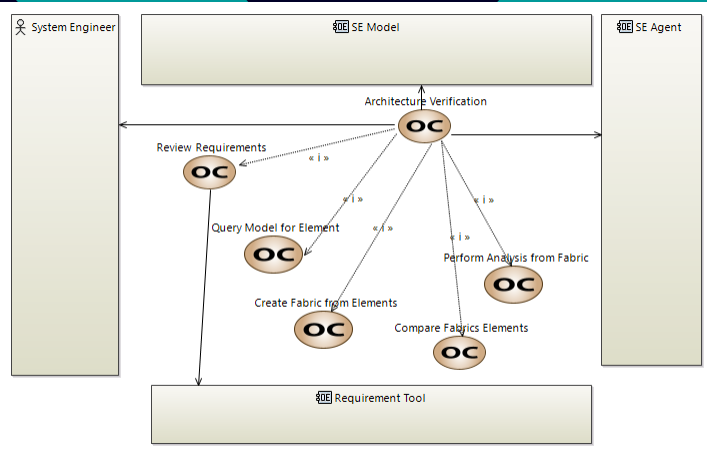
Requirements



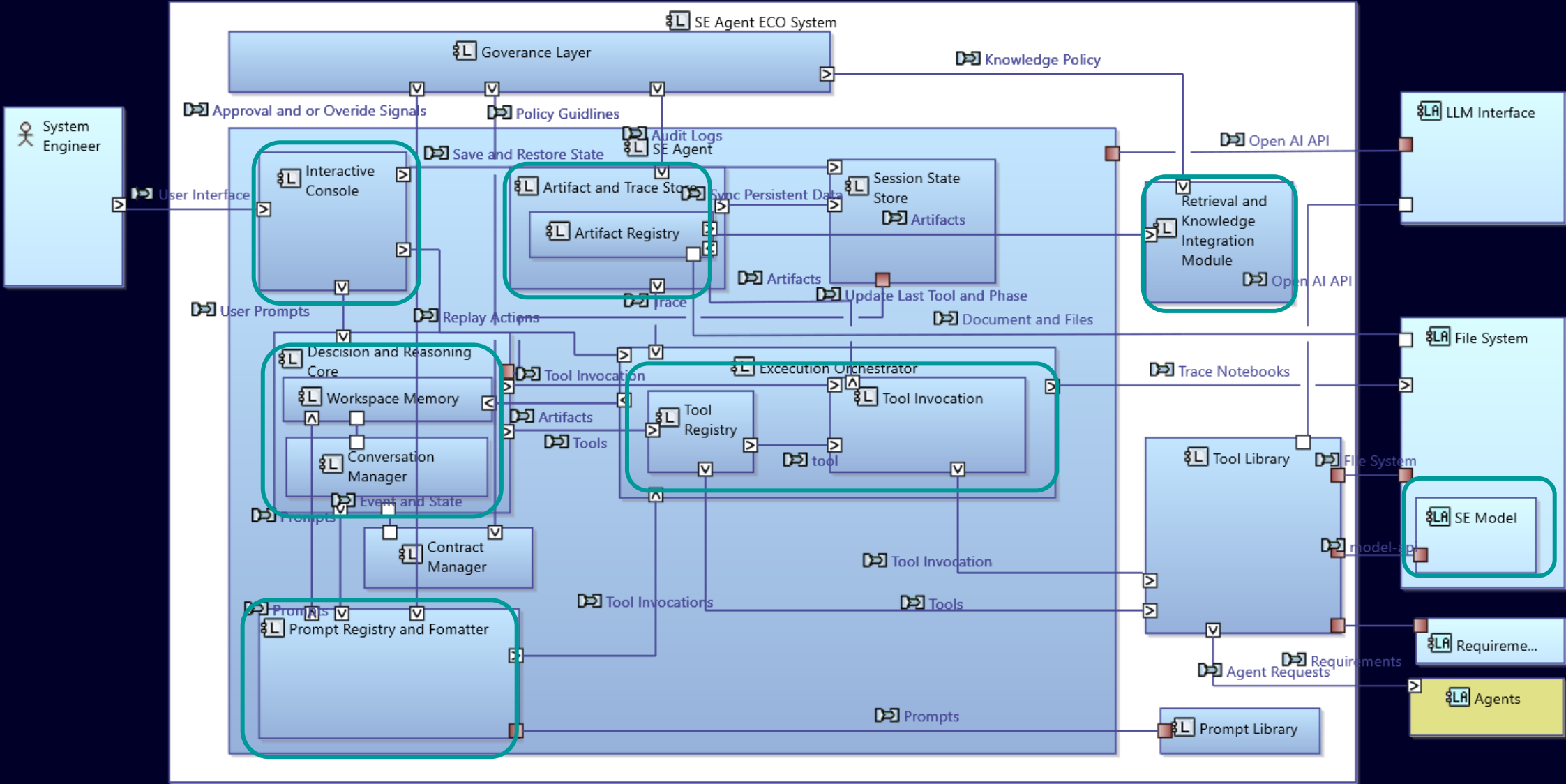
Verification



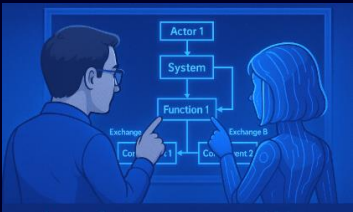
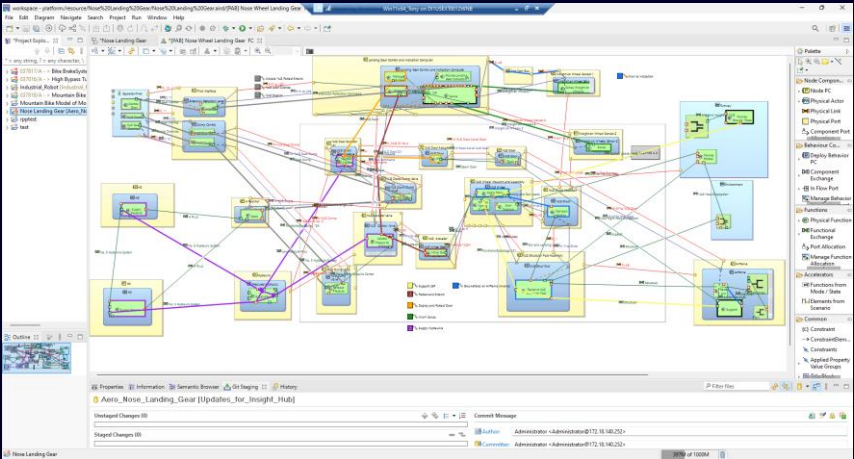
Architecture



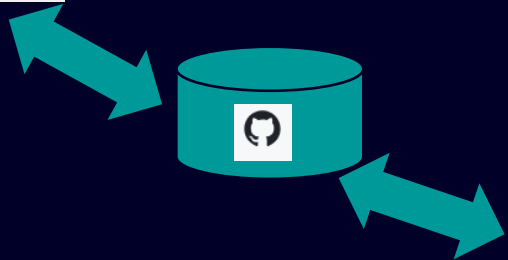
Architecture of System Engineering Knowledge Partner



Sustainment Demonstration using Agent based Knowledge Partner



This screenshot shows the 'SE Knowledge Partner' interface, which is a code editor for Python. The code defines an 'AgentCore' class and an 'Agent' class. The 'Agent' class has methods for 'create_capability_model_artifact', 'load_prompt_path', and 'load_prompt_path'. The code is written in a clean, professional style with syntax highlighting. The interface includes a sidebar with a project tree and a bottom panel showing properties and messages.



This screenshot shows the 'SE Knowledge Partner' interface, which is a code editor for Python. The code defines an 'AgentCore' class and an 'Agent' class. The 'Agent' class has methods for 'create_capability_model_artifact', 'load_prompt_path', and 'load_prompt_path'. The code is written in a clean, professional style with syntax highlighting. The interface includes a sidebar with a project tree and a bottom panel showing properties and messages.

The interactive session concludes by creating a notebook capturing the interactive workflow capable of being replayed.

Agent Replay Notebook

Generated from agent history.

Step 1: Run create_capella_model_artifact

Input: {'path_to_model': '/home/simcenter/studio/Aero_Nose_Landing_Gear/Nose_Landing_Gear.aid', 'resources': {'SE_Agent': '/home/simcenter/studio/Aero_Nose_Landing_Gear/}}

[]: _ = agent.run('create_capella_model_artifact', input_data={'path_to_model': '/home/simcenter/studio/Aero_Nose_Landing_Gear/Nose_Landing_Gear.aid', 'resources': {'SE_Agent': '/home/simcenter/studio/Aero_Nose_Landing_Gear/'}})

Step 2: Run load_prompt_path

Input: {'prompt_dir_path': '/home/simcenter/studio/Mountain_Bike/Mountain_Bike/Scalable_Agent_Notebook/My_Prompts/'}

[]: _ = agent.run('load_prompt_path', input_data={'prompt_dir_path': '/home/simcenter/studio/Mountain_Bike/Mountain_Bike/Scalable_Agent_Notebook/My_Prompts/'})

Step 3: Run save_prompt_artifact

Input: {'name': 'System Analysis of Fault Code from PAB', 'text': '#CONTEXT#\nYou know the model based system engineering ARCADIA method, knowledge of aircraft systems as defined in ATA Spec 100.\n\n#ROLE#\nYou are an aircraft system engineer assisting in taking an aircraft fault and finding a diagram that will depict it aircraft system functional chain.\n\n#TASK#\n1. Using the fault message "Nose Wheel Landing Gear is retracting 30 seconds slower then nominal" lookup in ATA Spec 100 a aircraft system that would create such a fault message.\n2. Use that aircraft system it to create a search prompt of format "Physical Architecture Diagram: [PAB] <aircraft_system> FC" replacing <aircraft_system> with the ATA system.\n3. Use the tool create_artifact to create an artifact of type "prompt" and name it "PAB_System_FC" set its content the prompt created.\n4. Use the tool create_artifact to create an artifact of type "rationale" and name it "System Rationale" set its content to rationale on the aircraft system selected from ATA 100 Spec provide details on spec section.', 'source_path': '/home/simcenter/studio/Mountain_Bike/Mountain_Bike/Scalable_Agent_Notebook/My_Prompts/fault_code_PAB.json', 'template_name': 'System Analysis of Fault Code from PAB', 'tags': ['fault', 'code', 'query']}

[]: _ = agent.run('save_prompt_artifact', input_data={'name': 'System Analysis of Fault Code from PAB', 'text': '#CONTEXT#\nYou know the model based system engineering ARCADIA method, knowledge of aircraft systems as defined in ATA Spec 100.\n\n#ROLE#\nYou are an aircraft system engineer assisting in taking an aircraft fault and finding a diagram that will depict it aircraft system functional chain.\n\n#TASK#\n1. Using the fault message "Nose Wheel Landing Gear is retracting 30 seconds slower then nominal" lookup in ATA Spec 100 a aircraft system that would create such a fault message.\n2. Use that aircraft system it to create a search prompt of format "Physical Architecture Diagram: [PAB] <aircraft_system> FC" replacing <aircraft_system> with the ATA system.\n3. Use the tool create_artifact to create an artifact of type "prompt" and name it "PAB_System_FC" set its content the prompt created.\n4. Use the tool create_artifact to create an artifact of type "rationale" and name it "System Rationale" set its content to rationale on the aircraft system selected from ATA 100 Spec provide details on spec section.', 'source_path': '/home/simcenter/studio/Mountain_Bike/Mountain_Bike/Scalable_Agent_Notebook/My_Prompts/fault_code_PAB.json', 'template_name': 'System Analysis of Fault Code from PAB', 'tags': ['fault', 'code', 'query']})

Step 4: Run llm_chat

Prompt: show artifact System Analysis of Fault Code from PAB

[]: agent.run('llm_chat', input_data={'prompt': 'show artifact System Analysis of Fault Code from PAB'})

Step 5: Run show_artifact

Input: {'name': 'System Analysis of Fault Code from PAB'}

[]: _ = agent.run('show_artifact', input_data={'name': 'System Analysis of Fault Code from PAB'})

Step 6: Run llm_chat

Prompt: Load artifact named "System Analysis of Fault Code from PAB" into workspace memory.

[]: agent.run('llm_chat', input_data={'prompt': 'Load artifact named "System Analysis of Fault Code from PAB" into workspace memory.'})

Step 7: Run load_artifact_into_workspace_memory

Input: {'artifact_name': 'System Analysis of Fault Code from PAB'}

[]: _ = agent.run('load_artifact_into_workspace_memory', input_data={'artifact_name': 'System Analysis of Fault Code from PAB'})

Step 8: Run llm_chat 1

Prompt: Plan and act upon the workspace memory item "System Analysis of Fault Code from PAB" content.

[]: agent.run('llm_chat', input_data={'prompt': 'Plan and act upon the workspace memory item "System Analysis of Fault Code from PAB" content.'})

Step 9: Run create_artifact

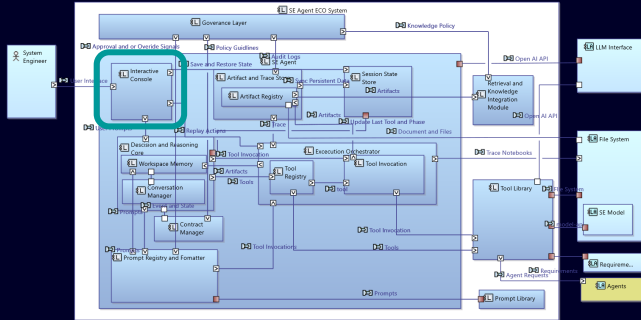
Input: {'name': 'PAB_System_FC', 'type': 'prompt', 'content': 'Physical Architecture Diagram: [PAB] Nose Wheel Landing Gear FC'}

[]: _ = agent.run('create_artifact', input_data={'name': 'PAB_System_FC', 'type': 'prompt', 'content': 'Physical Architecture Diagram: [PAB] Nose Wheel Landing Gear FC'})

[]: Using the fault message "Nose Wheel Landing Gear is retracting 30 seconds slower then nominal" lookup in ATA Spec 100 a aircraft system that would create such a fault message.\n2. Use that aircraft system it to create a sea

The notebook could be easily modified to automate the interactive task.

Review of what we saw: Interactive Console



Allows interaction with the Agent and view the state of agent and tools.

Type your prompt...

Execute

Exit

Load file:

Interactive chat started. Use ``run:tool {json}`` to call tools. Exit button to close.

State

Prompts

Refresh

Workspace (Newest – Oldest)

(none)

Artifacts (Newest – Oldest)

name	type
6e39938a-ed7c-4756-8565-d41b330062dd	capella_model
My_Prompts_prompt_path	prompt_path

Start Guided Session

Prompt (required)

Attach artifacts (optional)

_workspace_store

None

My_Prompts_prompt_path

Start Session (Agent)

Runnable Tools (A → Z)

None

name
create_artifact
create_capella_model_artifact
generate_arcadia_fabric
import_file_artifact
interactive_chat
list_artifacts
list_workspace
llm_chat
load_artifact_into_workspace_memory
load_from_workspace_memory_to_artifact
load_prompt_path
name_artifact
notebook_export

[illegible]

10

- ```
from se_agent.core.agent import AgentCore

agent = AgentCore()
agent.create_package("SE_Agent")
agent.use_package("SE_Agent")

agent.run('create_capella_model_artifact', input_data=
{'path_to_model': '/home/simcenter/studio/Aero_Nose_Landing_Gear/Nose Landing Gear/Nose Landing Gear.aird',
 'resources': {'SE_Agent': '/home/simcenter/studio/Aero_Nose_Landing_Gear/Nose Landing Gear/'}})
```

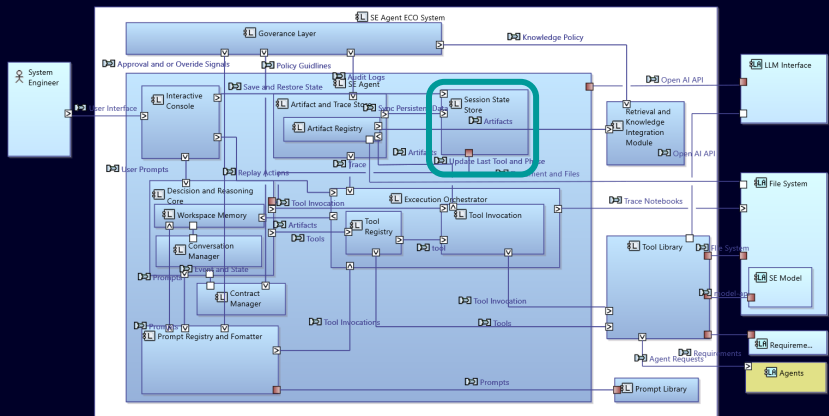


- | Artifacts (Newest → Oldest)            |               |
|----------------------------------------|---------------|
| name                                   | type          |
| System Analysis of Fault Code from PAB | prompt        |
| 20b662ff-e105-4b1a-9909-263c5c389080   | capella_model |
| My_Prompts_prompt_path                 | prompt_path   |



- | Workspace (Newest → Oldest)            |        |
|----------------------------------------|--------|
| name                                   | type   |
| System Analysis of Fault Code from PAB | prompt |

# Replay Notebook



Agent Replay Notebook

Generated from agent history.

Step 1: Run create\_capella\_model\_artifact

Input: {'path\_to\_model': '/home/simcenter/studio/Aero\_Nose\_Landing\_Gear/Nose\_Landing\_Gear/Nose\_Landing\_Gear.aird', 'resources': {'SE\_Agent': '/home/simcenter/studio/Aero\_Nose\_Landing\_Gear/Nose\_Landing\_Gear/'}}

[ ]: \_ = agent.run('create\_capella\_model\_artifact', input\_data={'path\_to\_model': '/home/simcenter/studio/Aero\_Nose\_Landing\_Gear/Nose\_Landing\_Gear/Nose\_Landing\_Gear.aird', 'resources': {'SE\_Agent': '/home/simcenter/studio/Aero\_Nose\_Landing\_Gear/'}})

Step 2: Run load\_prompt\_path

Input: {'prompt\_dir\_path': '/home/simcenter/studio/Mountain\_Bike/Mountain\_Bike/Scalable\_Agent\_Notebook/My\_Prompts/'}

[ ]: \_ = agent.run('load\_prompt\_path', input\_data={'prompt\_dir\_path': '/home/simcenter/studio/Mountain\_Bike/Mountain\_Bike/Scalable\_Agent\_Notebook/My\_Prompts/'})

Step 3: Run save\_prompt\_artifact

Input: {'name': 'System Analysis of Fault Code from PAB', 'text': '#CONTEXT#\nYou know the model based system engineering ARCADIA method, knowledge of aircraft systems as defined in ATA Spec 100.\n\n#ROLE#\nYou are an aircraft system engineer assisting in taking an aircraft fault and finding a diagram that will depict it aircraft system functional chain. \n\n#TASKS#\n1. Using the fault message "Nose Wheel Landing Gear is retracting 30 seconds slower then nominal" lookup in ATA Spec 100 a aircraft system that would create such a fault message.\n2. Use that aircraft system it to create a search prompt of format "Physical Architecture Diagram: [PAB] <aircraft\_system> FC" replacing <aircraft\_system> with the ATA system. \n3. Use the tool create\_artifact to create an artifact of type "prompt" and name it "PAB\_System\_FC" set its content the prompt created.\n4. Use the tool create\_artifact to create an artifact of type "rationale" and name it "System Rationale" set its content to rationale on the aircraft system selected from ATA 100 Spec provide details on spec section.', 'source\_path': '/home/simcenter/studio/Mountain\_Bike/Mountain\_Bike/Scalable\_Agent\_Notebook/My\_Prompts/fault\_code\_PAB.json', 'template\_name': 'System Analysis of Fault Code from PAB', 'tags': ['fault', 'code', 'query']}

[ ]: \_ = agent.run('save\_prompt\_artifact', input\_data={'name': 'System Analysis of Fault Code from PAB', 'text': '#CONTEXT#\nYou know the model based system engineering ARCADIA method, knowledge of aircraft systems as defined in ATA Spec 100.\n\n#ROLE#\nYou are an aircraft system engineer assisting in taking an aircraft fault and finding a diagram that will depict it aircraft system functional chain. \n\n#TASKS#\n1. Using the fault message "Nose Wheel Landing Gear is retracting 30 seconds slower then nominal" lookup in ATA Spec 100 a aircraft system that would create such a fault message.\n2. Use that aircraft system it to create a search prompt of format "Physical Architecture Diagram: [PAB] <aircraft\_system> FC" replacing <aircraft\_system> with the ATA system. \n3. Use the tool create\_artifact to create an artifact of type "prompt" and name it "PAB\_System\_FC" set its content the prompt created.\n4. Use the tool create\_artifact to create an artifact of type "rationale" and name it "System Rationale" set its content to rationale on the aircraft system selected from ATA 100 Spec provide details on spec section.', 'source\_path': '/home/simcenter/studio/Mountain\_Bike/Mountain\_Bike/Scalable\_Agent\_Notebook/My\_Prompts/fault\_code\_PAB.json', 'template\_name': 'System Analysis of Fault Code from PAB', 'tags': ['fault', 'code', 'query']})

Step 4: Run llm\_chat

Prompt: show artifact System Analysis of Fault Code from PAB

[ ]: agent.run('llm\_chat', input\_data={'prompt': 'show artifact System Analysis of Fault Code from PAB'})

Step 5: Run show\_artifact

Input: {'name': 'System Analysis of Fault Code from PAB'}

[ ]: \_ = agent.run('show\_artifact', input\_data={'name': 'System Analysis of Fault Code from PAB'})

Step 6: Run llm\_chat

Prompt: Load artifact named "System Analysis of Fault Code from PAB" into workspace memory.

[ ]: agent.run('llm\_chat', input\_data={'prompt': 'Load artifact named "System Analysis of Fault Code from PAB" into workspace memory.'})

Step 7: Run load\_artifact\_into\_workspace\_memory

Input: {'artifact\_name': 'System Analysis of Fault Code from PAB'}

[ ]: \_ = agent.run('load\_artifact\_into\_workspace\_memory', input\_data={'artifact\_name': 'System Analysis of Fault Code from PAB'})

Step 8: Run llm\_chat 1

Prompt: Plan and act upon the workspace memory item "System Analysis of Fault Code from PAB" content.

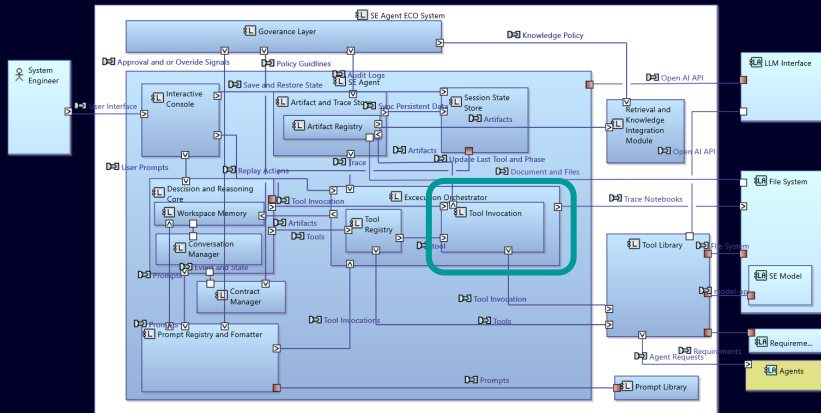
[ ]: agent.run('llm\_chat', input\_data={'prompt': 'Plan and act upon the workspace memory item "System Analysis of Fault Code from PAB" content.'})

Step 9: Run create\_artifact

Input: {'name': 'PAB\_System\_FC', 'type': 'prompt', 'content': 'Physical Architecture Diagram: [PAB] Nose Wheel Landing Gear FC'}

[ ]: \_ = agent.run('create\_artifact', input\_data={'name': 'PAB\_System\_FC', 'type': 'prompt', 'content': 'Physical Architecture Diagram: [PAB] Nose Wheel Landing Gear FC'})

# Review of what we saw: Tools



Tools are:

Extensible: Easy to add additional tools.

Discoverable : Agent can discover new tools without code change.

Artifact Driven : Tools expect artifacts as inputs and deliver as output. (IO Schema)

## Artifact Management

- **create\_artifact**: Create a new artifact with the given name, type, and content in the target package.
- **name\_artifact**: Assign a name to an existing artifact.
- **import\_file\_artifact**: Register a local file as a 'file\_reference' artifact (path + metadata).

## Capella Model Operations

- **create\_capella\_model\_artifact**: Create a Capella model artifact from a path and resources.
- **query\_capella\_model**: Query a Capella model using embeddings and save a selection artifact.
- **generate\_arcadia\_fabric**: Generate a YAML-based knowledge fabric for selected Capella objects and save it as an artifact.

## Prompt Management

- **load\_prompt\_path**: Load a directory of notebook-defined prompts and save a pointer artifact for other tools to use.
- **save\_prompt\_artifact**: Render a prompt from a prompt specification artifact and save it.
- **execute\_prompt\_for\_session**: Start a Guided Session from a required prompt artifact, optionally attaching other artifacts.
- **show\_prompt\_spec**: Display a prompt specification artifact with title, version, and JSON preview.

## Data Import/Export

- **read\_csv**: Import a CSV and create a 'table' artifact (columns + records).
- **read\_leveled\_csv**: Import a leveled CSV file and create a hierarchy artifact.
- **notebook\_export**: Export agent history as a Jupyter notebook for replay.

## Workspace and Memory Management

- **list\_artifacts**: List artifacts in the active (or specified) package with a Markdown summary.
- **list\_workspace**: Display the current workspace contents: artifacts and memory entries.
- **load\_artifact\_into\_workspace\_memory**: Load an artifact into workspace memory by name or latest type.
- **load\_from\_workspace\_memory\_to\_artifact**: Load a workspace memory entry as a typed artifact.

## Interactive and Chat Tools

- **interactive\_chat**: Launch an interactive chat session; uses an internal chat setup.
- **llm\_chat**: Conduct a tool-aware chat with the language model and store conversation turns.

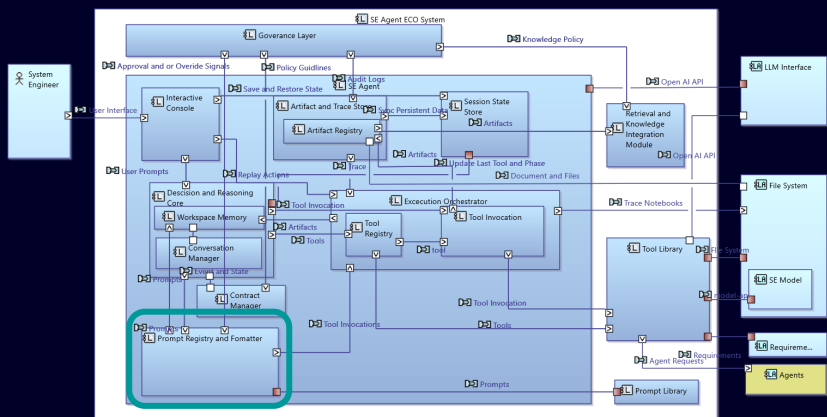
## Analysis and Reasoning

- **reason\_on\_arcadia\_fabric\_or\_files**: Use ChatGPTAnalyzer to reason on an arcadia\_fabric and/or file\_reference artifacts.

## Utility

- **search\_prompts**: List prompt notebooks within a prompt\_path artifact, with optional filtering.
- **show\_artifact**: Display a stored artifact (raw content).
- **show\_artifact\_memory**: Summarize artifacts in a package with counts and examples.

# Review of what we saw: Prompts



## #CONTEXT#

You know the model based system engineering ARCADIA method, knowledge of aircraft systems as defined in ATA Spec 100.

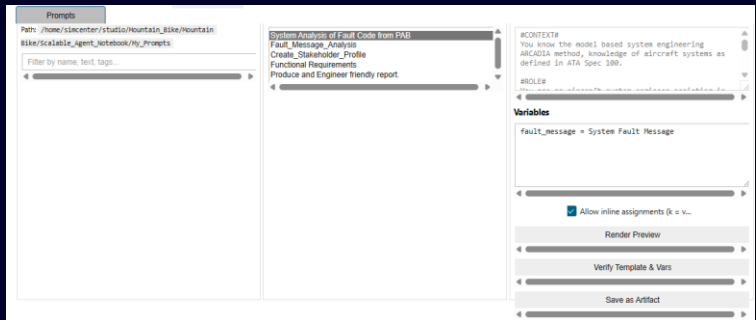
## #ROLE#

You are an aircraft system engineer assisting in taking an aircraft fault and finding a diagram that will depict it aircraft system functional chain.

## #TASKS#

1. Using the fault message "{{fault\_message}}" lookup in ATA Spec 100 a aircraft system that would create such a fault message.
2. Use that aircraft system it to create a search prompt of format "Physical Architecture Diagram: [PAB] <aircraft\_system> FC" replacing <aircraft\_system> with the ATA system.
3. Use the tool create\_artifact to create an artifact of type "prompt" and name it "PAB\_System\_FC" set its content the prompt created.
4. Use the tool create\_artifact to create an artifact of type "rationale" and name it "System Rationale" set its content to rationale on the aircraft system selected from ATA 100 Spec provide details on spec section.

## Dedicated UI



What would be the cause of this {{fault\_message}}?

Indicate specific impacted functional chains and components. Be verbose basing answer on model content and aircraft knowledge.

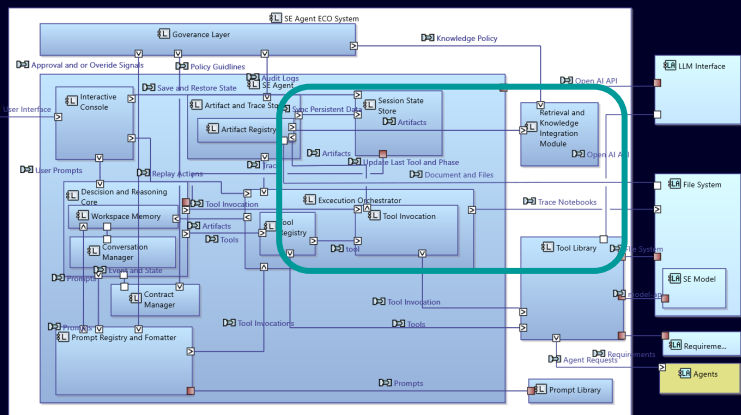
Explain the relationship of components and delivered function based on the model content.

What would be long term problem it issue persists? Give a discussion based on specific components and long term impact if not addressed.

## Extensible and Configurable

```
agent.run('load_prompt_path', input_data={'prompt_dir_path':
'/home/simcenter/studio/Mountain_Bike/Mountain
Bike/Scalable_Agent_Notebook/My_Prompts/'})
```

# Review of what we saw: Tool invoking Retrieval and Knowledge Integration



Spins up a separate LLM to support tasks

## Nose Wheel Landing Gear Deployment Delay Analysis

### Cause of Delay

The delay in the deployment of the nose wheel landing gear could be attributed to several components and functional chains within the system model. Based on the YAML model content, the following components and functional chains are likely impacted:

- **Hydraulic System:** The *SupplyPressure* function, which is part of the *Pressured Hydraulic* component, is responsible for providing hydraulic pressure to the landing gear system. If there is a delay in hydraulic pressure supply, it could directly impact the deployment time.
- **Control Pressure Function:** The *Control Pressure* function, owned by the *NLG Doors Selector Valve*, manages the pressure required for gear deployment. Any malfunction or delay in this function could lead to slower gear deployment.
- **Functional Chain - Retract and Extend:** This chain involves the *Control Pressure to Wheel* and *Manage NLG* functions. Any delay in these functions could result in a longer deployment time.

### Relationship of Components and Delivered Function

The deployment of the nose wheel landing gear involves a series of interconnected components and functions:

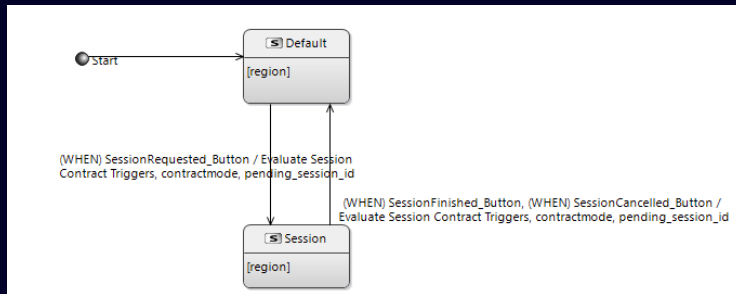
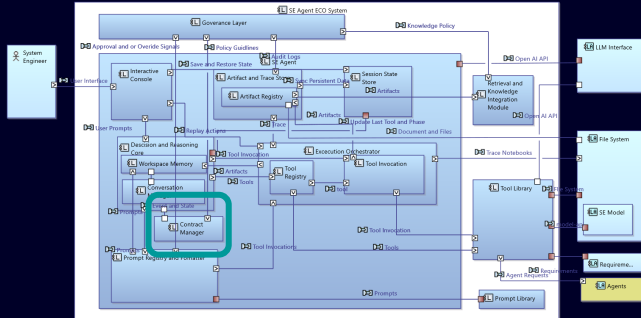
- The *Landing Gear Control and Indication Computer* manages the overall control of the landing gear system, including the *Manage NLG* function.
- The *NLG Control Valve* and *NLG Selector Valve* are critical in directing hydraulic pressure to the landing gear actuator, enabling the physical movement of the gear.
- The *SupplyPressure* function ensures that sufficient hydraulic pressure is available for the gear deployment process.

### Long-term Impact if Issue Persists

If the delay in gear deployment persists, it could lead to several long-term issues:

- **Increased Wear and Tear:** Prolonged deployment times can increase the mechanical stress on the landing gear components, leading to accelerated wear and potential failure.
- **Safety Risks:** Delayed gear deployment can pose significant safety risks during landing, especially in emergency situations where timely gear deployment is critical.
- **Operational Delays:** Persistent delays can lead to operational inefficiencies, affecting flight schedules and increasing maintenance costs.

# Contracts: Used to guide the agent behavior.



## # 3) Interaction contract

```
contract = [
```

```
 "You are a systems engineering assistant with access to tools.",
```

### # How to think (silently) before acting

```
"Before proposing actions, silently check the ToolRegistry IO_SCHEMA for each tool you intend to use.",
```

### # What to output when actions are possible

```
"When actions can be taken, reply ONLY with a JSON object containing an 'actions' list. Do NOT include any other text.",
```

```
"Each action must be of the form: {\\"tool\\": <tool_name>, \\"input\\": {<fields>}}",
```

### # Hard rules for inputs (teach, don't fix)

```
"INPUT RULES:",
```

- " - Include ONLY input fields that are defined in the tool's IO\_SCHEMA.inputs.",
- " - NEVER invent fields (e.g., 'version' unless IO\_SCHEMA includes it).",
- " - Omit keys with empty values (\\", null) entirely.",
- " - Satisfy all required inputs. If a required input is missing, first add an action to create or fetch it.",
- " - Use artifact identifiers that exist: prefer 'name' OR 'id' as specified by the tool; do not include both unless required.",
- " - Match value types suggested by IO\_SCHEMA (string, integer, boolean, dict, list, path).",
- " - If a tool expects a single-asset artifact (e.g., 'capella\_model'), reference it by name or id as the tool specifies.",
- " - Do not include package in inputs unless the tool explicitly defines a 'package' input; the runtime provides package scope.",

### # Planning & chaining

```
"PLANNING:",
```

- " - Plan from available artifacts to desired outputs. If prerequisites are missing, create them first.",
- " - Use the tool's declared outputs (IO\_SCHEMA.outputs) to chain to the next action.",
- " - Prefer minimal, correct sequences over long plans. Keep actions ≤ 3 unless the task truly requires more.",
- " - If no tool applies or required data cannot be created without user input, reply naturally and ask only for the missing essentials.",

### # Output format and cleanliness

```
"FORMAT:",
```

- " - Return ONLY the JSON with 'actions'. No markdown fences, no json prefix, no commentary, no 'run:' lines.",
- " - Do not propose {\\"tool\\":\\"interactive\_chat\\"} recursively.",
- " - Do not echo previous conversations or artifacts in the JSON.",

### # Examples (schema-compliant)

```
"EXAMPLES:",
```

```
' {"actions":[{"tool":"read_leveled_csv","input":{"filename":"drone.csv"}}]}' ,
' {"actions":[{"tool":"name_artifact","input":{"type":"hierarchy","name":"BOM"}}]}' ,
' {"actions":[{"tool":"show_artifact","input":{"name":"SEA_Capella_Model"}}]}' ,
' {"actions":[{"tool":"query_capella_model","input":{"capella_model_name":"BikeModel","query":"brake lever","top_n":25}}]}' ,
```

### # Final reminder

```
"If no tool applies, respond naturally."
```

# Conclusion: Agent Technology Delivers on System Engineering Knowledge Partner

- **Agents can assist users**
  - They can guide and facilitate use.
- **Agents can act autonomously.**
  - They can make decisions or take actions without constant human instruction.
- **They integrate multiple tools or APIs.**
  - Agents can call external systems—search, databases, models, or scripts to perform a task..
- **They can collaborate with humans or other agents.**
  - Multi-agent systems share tasks or negotiate outcomes.
- **They can support for repetitive or knowledge-intensive tasks.**
  - Examples include research, summarization, analysis, and automation.

## Demonstration Example

Agent can help a user use tools.

Execution a multi-step prompt.

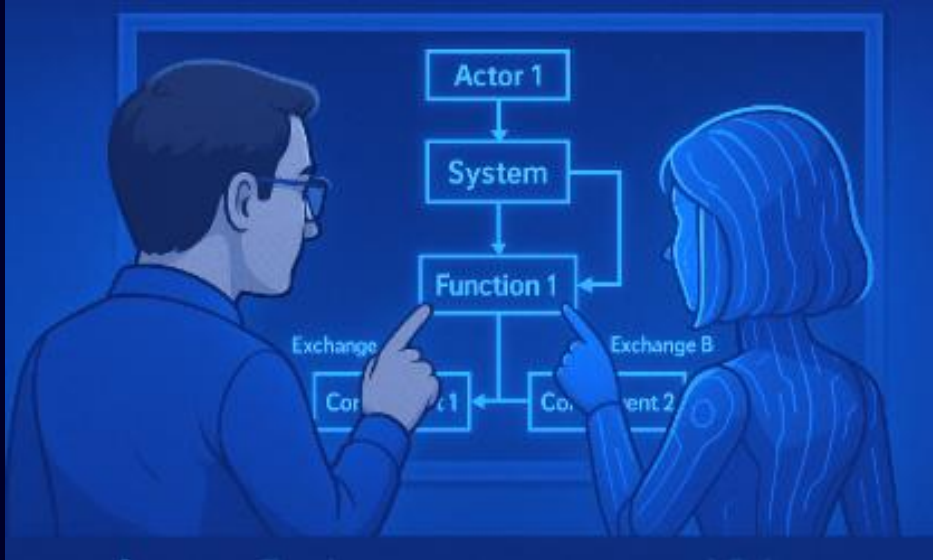
✓ Building a prompt based on ATA Spec  
Creating Artifacts (Rationale and Prompt)

✓ Creating of Capella Fabric using pycapellambse  
api

Agent has shown to help user do task and perform operations in a “session”

Agent technology is being used to create a replayable scenario, and to chain together a set of tasks.

## Next Steps



Extending Knowledge Partner from what we see here.

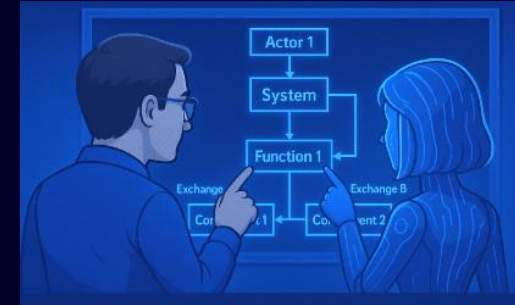
- updates to models
- update requirements
- create verification tests
- translate to/from other downstream down stream domains and tools

## Key take aways from today's discussion

An AI based Knowledge Partner is possible and can be implemented on a Capella Model.

Large Language Models combined with System Model has huge potential, grounding analysis in the content of the model.

Take aways from today's discussion  
Capella can help you design you AI Agent.



# You need to get ready for using AI with System Engineering with models

## If you want to learn more and stay informed.

- Follow me on Linked In : <https://www.linkedin.com/in/tony-komar/>
- Please subscribe on in YouTube: <https://www.youtube.com/@innovatingwithcapella/>
- Email me at [tony.komar@siemens.com](mailto:tony.komar@siemens.com)
- Email me at [ajk5442@psu.edu](mailto:ajk5442@psu.edu)
- Look for future Tutorials with INCOSE
- Siemens Products and Contacts noted in presentation
  - Simcenter Studio – Mike Nicolai – Product Manager - [mike.nicolai@siemens.com](mailto:mike.nicolai@siemens.com)
  - System Modeling Workbench – Albino Pereira Teixeira – Product Manager - [albino.pereira@siemens.com](mailto:albino.pereira@siemens.com)
  - SMW/Capella to SysML V2 Converter – Karen Ryan - [karen.ryan@siemens.com](mailto:karen.ryan@siemens.com)



# Source used as basis on Agent Architecture

## **Annotated Bibliography: Modular / Contract-Driven Agent Architectures**

### **AgentSquare: Automatic LLM Agent Search in Modular Design Space**

*Shang, Yu et al. (2024) — arXiv preprint*

Defines a modular design space for LLM agents (dividing logical blocks into Planning, Reasoning, Tool Use, Memory) and proposes module evolution & recombination plus a performance predictor to automatically search for high-performing agent configurations. Aligns closely with the concept of 'contracts' as swappable modules.

Link: <https://arxiv.org/abs/2410.06153>

### **LLM-based Agent Unified Modeling Framework (LLM-Agent-UMF)**

*Hassouna, Chaari, Belhaj, et al. (2024) — arXiv preprint*

Proposes a unified modeling framework for LLM agents that clearly separates core functional modules (planning, memory, profile, action, security) and introduces a 'core agent' coordinator. Good reference for formalizing how 'contracts' fit into an agent's architecture.

Link: <https://arxiv.org/abs/2409.11393>

### **Agent Design Pattern Catalogue: A Collection of Architectural Patterns**

*(2024) — arXiv review / pattern catalogue*

Presents 17 architectural patterns for foundation model-based agents, discussing forces, tradeoffs, and contexts. Demonstrates that 'contracts' fit into broader architectural pattern taxonomies.

Link: <https://arxiv.org/html/2405.10467v2>

### **Evaluation-Driven Development of LLM Agents: A Process Model and Reference Architecture**

*Xia, Lu, Zhu, Xing, Zhao, Zhang (2024) — arXiv preprint*

Introduces a design approach analogous to test-driven development for LLM agents, integrating evaluation loops into runtime architecture. Supports the idea that decision contracts should be monitored and iterated.

Link: <https://arxiv.org/abs/2411.13768>

### **Improving Planning with Large Language Models: A Modular Agentic Architecture**

*Webb, Mondal, Momennejad (2023) — arXiv preprint*

Proposes a Modular Agentic Planner (MAP) architecture, where planning is decomposed into interacting modules handled via LLMs. Demonstrates benefits of modular pluggability, useful analogy for 'contract' integration.

Link: <https://arxiv.org/abs/2310.00194>

### **A Concurrent Modular Agent: Framework for Autonomous LLM Agents**

*Maruyama, Yoshida, Sato, et al. (2025) — arXiv preprint*

Introduces Concurrent Modular Agent (CMA), where modules run asynchronously yet share a coherent state via inter-module communication. Relevant when contracts overlap or run concurrently.

Link: <https://arxiv.org/abs/2508.19042>

### **Large Language Model Agent: A Survey on Methodology**

*(2025) — arXiv survey*

Deconstructs LLM agent systems via a methodology-oriented taxonomy: architectural patterns, collaboration, evaluation, and evolution. Provides context for positioning 'contracts' within agent design practices.

Link: <https://arxiv.org/abs/2503.21460>

### **Agentic AI on AWS: Design Patterns & Considerations**

*AWS Community Article*

High-value professional resource articulating agentic design patterns, service mappings, and practical constraints. Good for showing industry alignment with contract-based agent design.

Link: <https://community.aws/content/2qlet7NAUR02ygnurN1O74ClwTN/agentic-ai-framework-on-aws-the-design-patterns-and-considerations>