MBSE CYBER SYSTEMS SYMPOSIUM 2025

REST API'S IN TEAMWORK CLOUD

From Basics to Best Practices

DASSAULT SYSTEMES

MBSE CYBER SYSTEMS SYMPOSIUM 2025

David Fields
Co-Found & Chief Technology Officer
Enola Technologies

Jeff Seufer
Cyber Systems Industry Process Consultant
Dassault Systemes

Table of Contents

INTRODUCTION

ENOLA

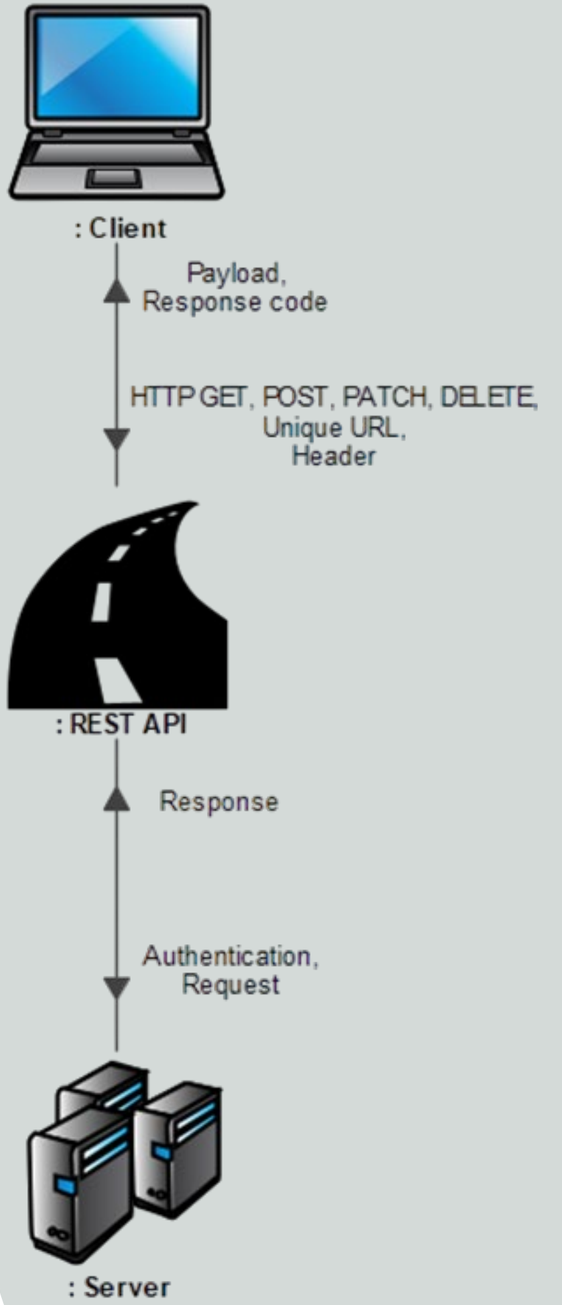
What is REST API?

Definition:

REpresentative **S**tate **T**ransfer (REST)
Application **P**rogramming **I**nterface (API)

Translation:

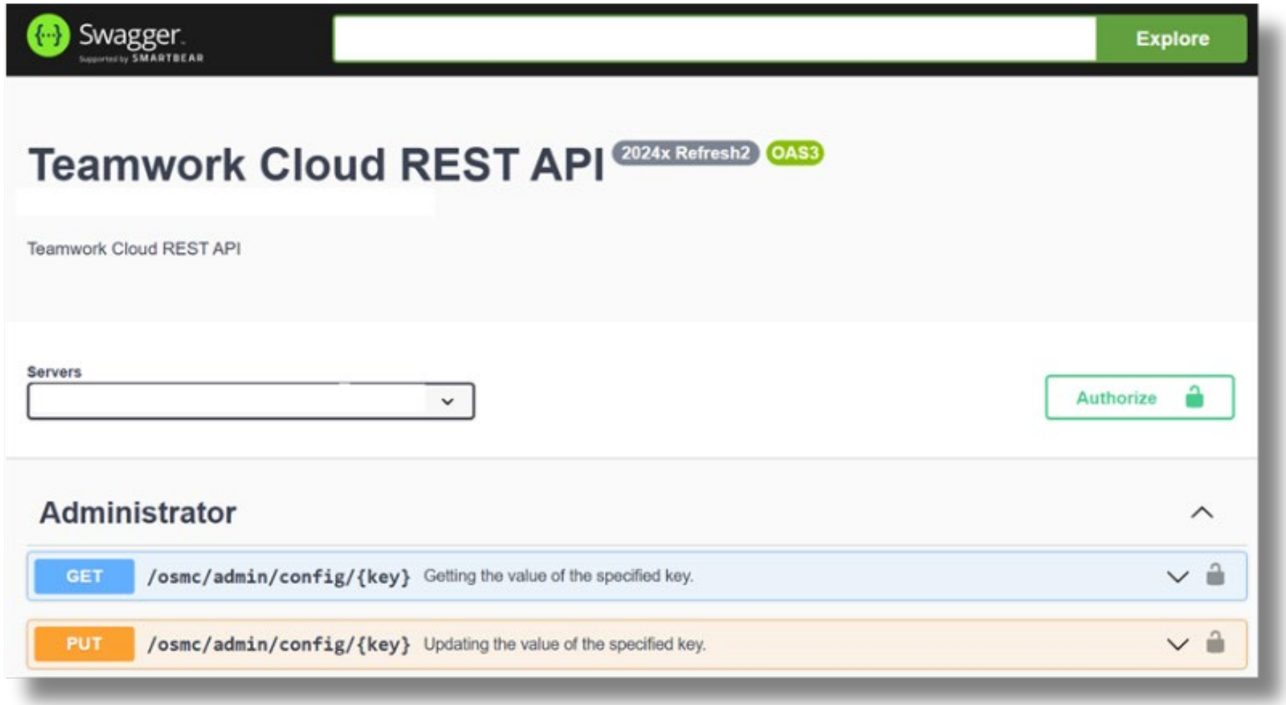
- Provides a set of rules for exchanging data between a client and a server
- Functions using a stateless request-response method
- Utilized HTTP methods for CRUD operations through:
  - Create ⟶ POST
  - Read ⟶ GET
  - Update ⟶ PUT or PATCH
  - Delete ⟶ DELETE
- Each resource in a REST API has a unique URL, which is used to access it
- Includes standard response status codes and response format types like XML and JSON
- Utilizes request Headers for authentication through different protocols



Working with REST APIs

Common Tools

- Postman
- Swagger UI
- Python via Requests Library
- Java HTTP from Java Standard Library
- C# using the built in HttpClient
- cURL through many terminal tools

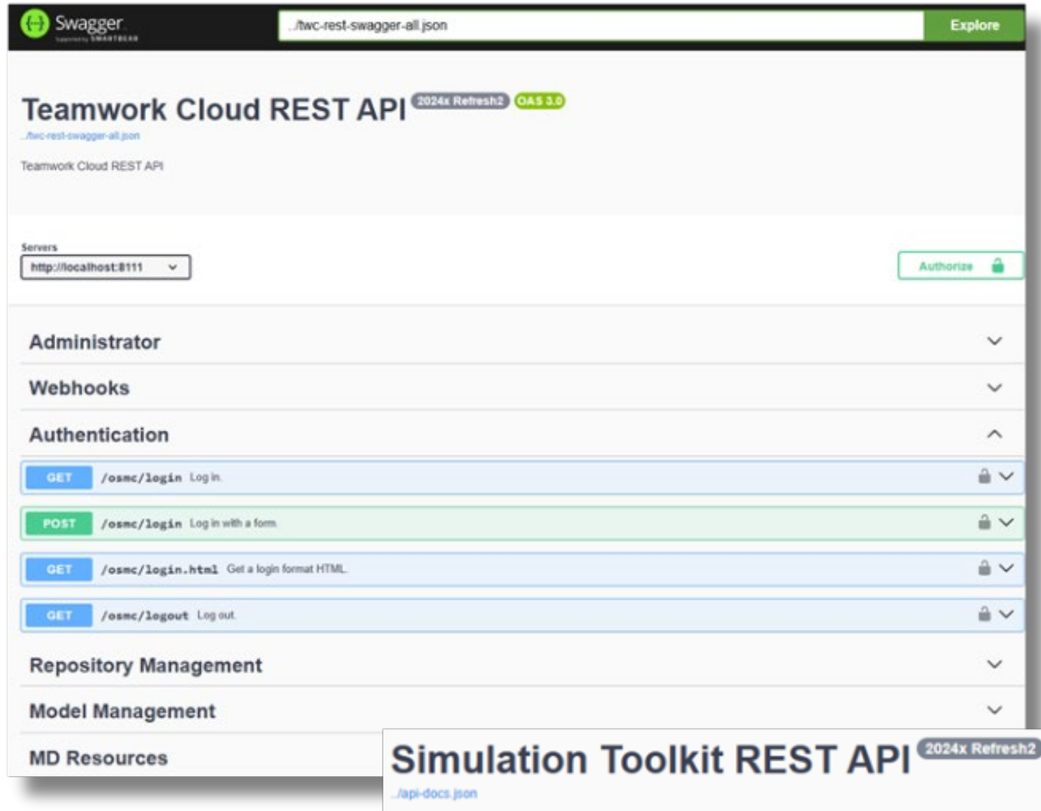


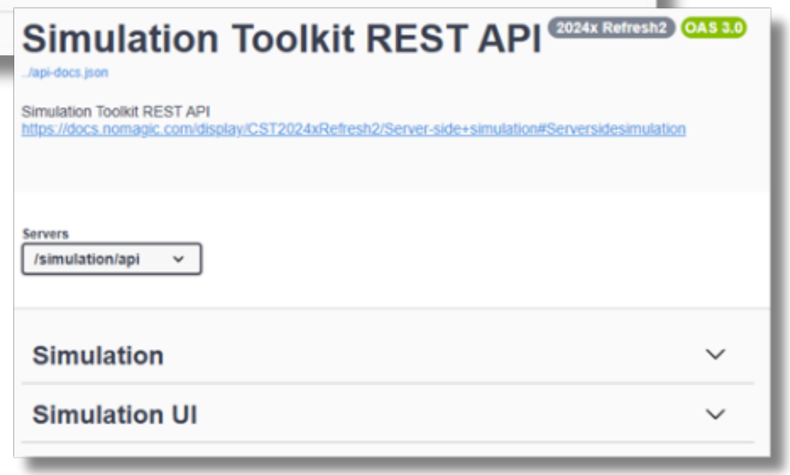

```
14  from requests import Session
15  session = Session()
16  session.verify = verifySSL
17  session.auth = (username, password)
18
19  api_call = f"/osmc/workspaces" # get all workspaces
20  payload = session.get(f"{server}{api_call}").json()
```

The Teamwork Cloud REST API

Teamwork Cloud provides a set of REST APIs for server management

- REST APIs are essential properties in Teamwork Cloud that allow you to query data about user accounts, roles, projects, and that provide access and basic operations for the resources
- REST API also provides an administrator-friendly way to manage the server because it can be run in the text mode and in a shell script.





Five Main Sections

- Administrator
- Webhooks
- Authentication
- Repository Management
- Model Management

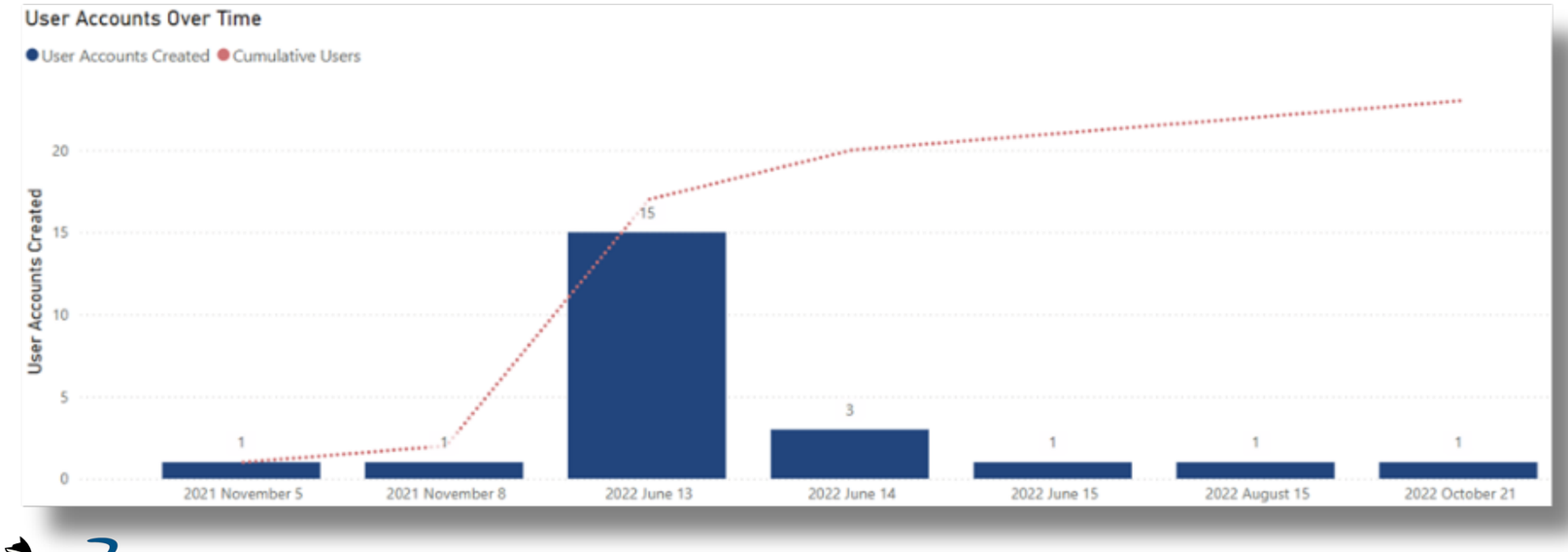
A separate simulation toolkit REST API enables server simulations from scripts - out of scope for this presentation and covered in tutorials at MCSS

ADMINISTRATIVE METRICS

ENOLA

Metrics Overview

Teamwork Cloud provides several basic REST API calls for getting information about all the users, projects, categories, etc. on the server:



The JSON response can be imported into a Business Intelligence (BI) or data analysis tools for visualization and analysis

ENOLA

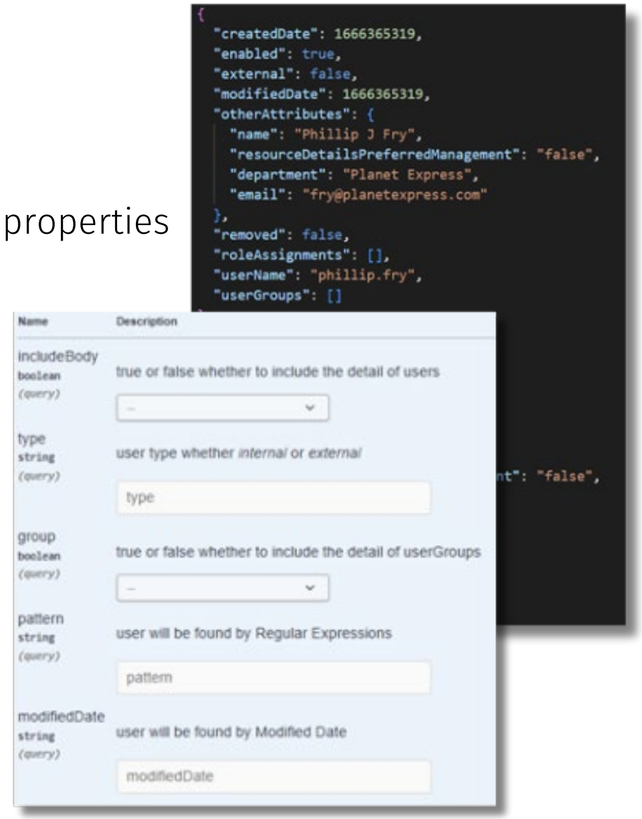
Example:
Get All Users with Details

GET /osmc/admin/users List all users.

A single REST API call gets every user accounts on the Teamwork Cloud server

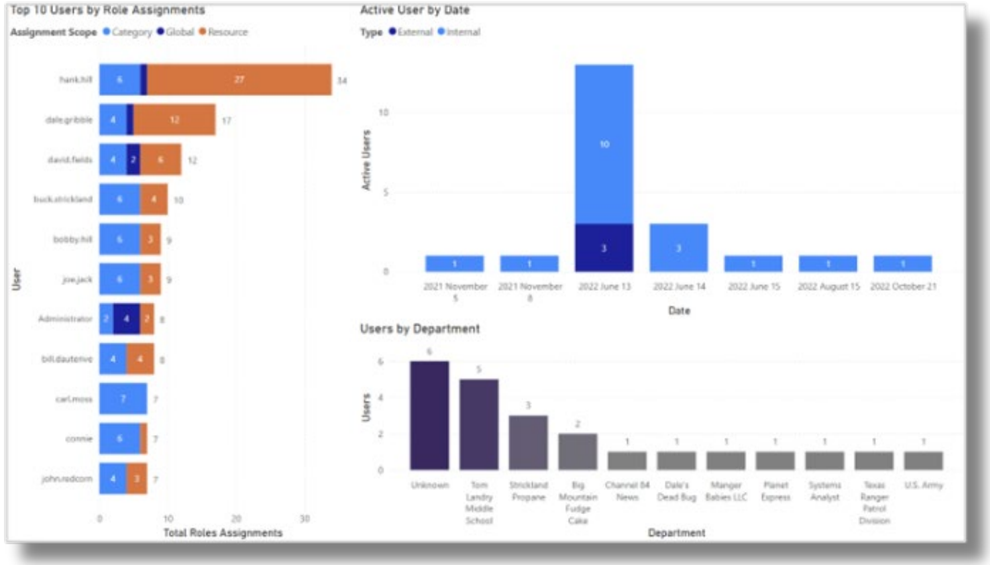
- Has optional parameters for additional properties and filtering
  - **includeBody** : If true, returns additional data about the user including Name, Email, and Department properties
  - **type** : Allows the results to be filtered by type of users (internal or external)
  - **group** : If true, returns information about the User Groups the users a member of
  - **pattern** : Allows for advance queries using Regular Expressions
  - **modifiedDate** : Allows for filtering by Modified Date

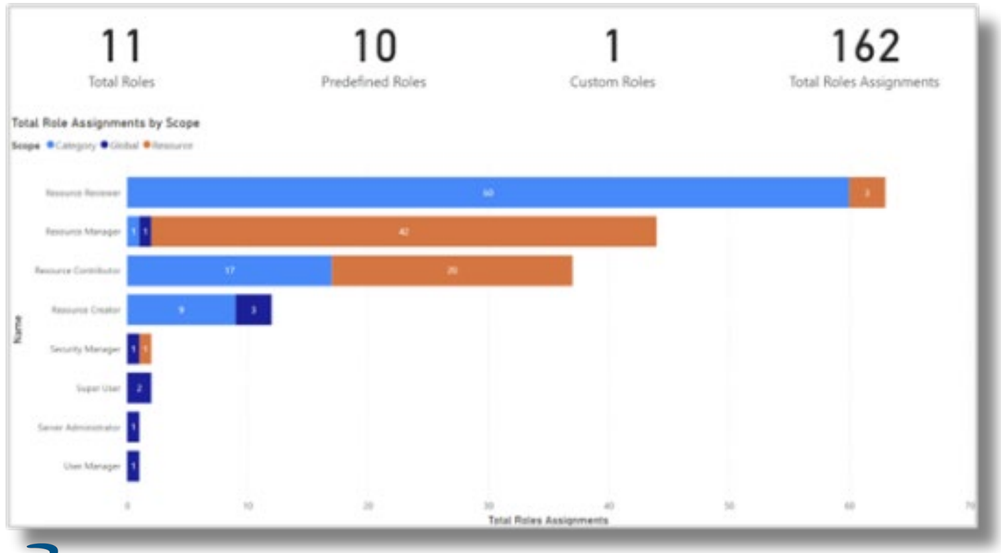
- When combined with data from additional endpoints, a full picture of the user account be created!
  - **User Details**: Name, Department, and other basic account information
  - **User Groups**: All User Groups the users is a member of
  - **Role Assignments**: All role assignments for the user including resource, scope, assignment date, etc.
  - **Resource Assignments**: All the Resources the user has access to and with what roles

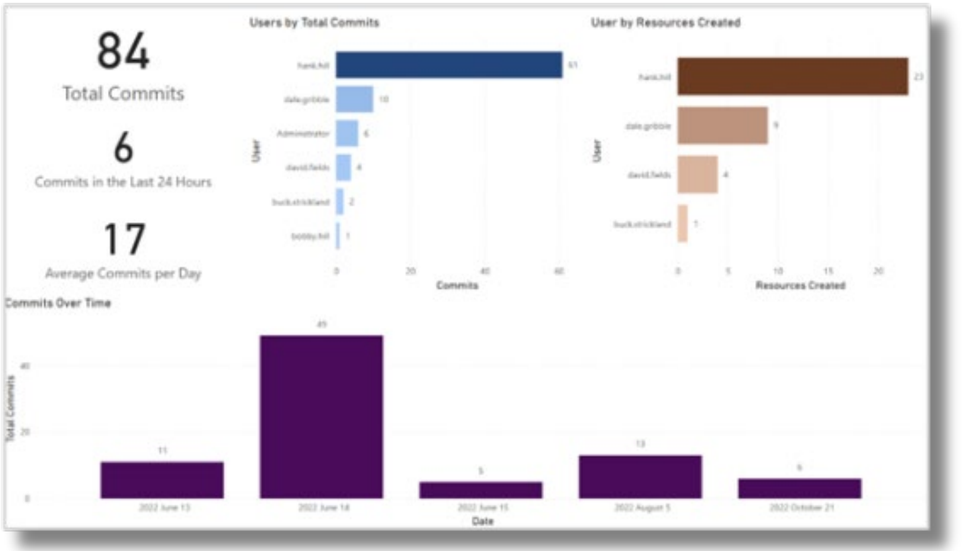


ENOLA

Metrics Dashboards in PowerBI





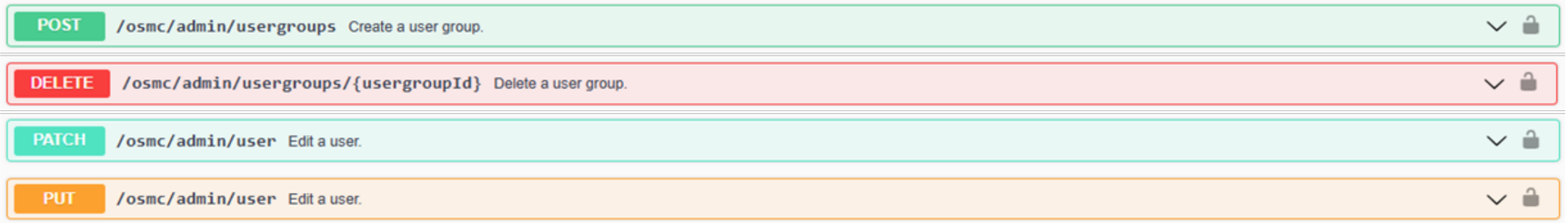


ADMINISTRATIVE AUTOMATION

ENOLA

Automation Overview

- **Allows for automation and management using scripts**
  - Python, PowerShell, Bash, or other languages that can leverage HTTPS request and JSON data
  - One time or scheduled operation
- **Example Use Cases:**
  - Create users in bulk
  - Update user information
  - Disable accounts that haven't logged in a specified period
  - Verify and update User Group membership
  - Check for and remove global scope assignments
  - Manage user access control from a custom interface



Example: Disable Users

```powershell
# Get all users
try {
    Write-Host "Fetching users from $TeamworkCloudUrl/osmc/admin/users?includeBody=true"
    $Response = Invoke-RestMethod -Uri "$TeamworkCloudUrl/osmc/admin/users?includeBody=true" -Headers $Headers -Method Get

    if ($null -eq $Response -or $Response.Count -eq 0) {
        Write-Warning "No users found."
        exit
    }

    Write-Host "Retrieved $($Response.Count) users."

    $UsersToDisable = @()

    # Identify users who haven't logged in for over threshold days
    foreach ($User in $Response) {
        if ($null -ne $User.lastLoginDate -and $User.lastLoginDate -lt $DaysAgoUnixTime -and $User.enabled -eq $true) {
            $LastLoginDate = [DateTimeOffset]::FromUnixTimeSeconds($User.lastLoginDate).DateTime
            $UsersToDisable += [PSCustomObject]@{
                Username = $User.username
                LastLoginDate = $LastLoginDate
                DaysSinceLogin = [math]::Floor(($CurrentUnixTime - $User.lastLoginDate) / 86400)
                Realm = $User.realm
            }
        }
    }
```

```powershell
# Disable users
foreach ($User in $UsersToDisable) {
    $UserUrl = "$TeamworkCloudUrl/osmc/admin/user?username=$($User.Username)"

    # Prepare request body
    if ($MinimalPayload) {
        # Only include the enabled field
        $RequestBody = @{
            enabled = $false
        } | ConvertTo-Json
    } else {
        # Get current user data first to ensure we have all required fields
        $CurrentUserData = Invoke-RestMethod -Uri $UserUrl -Headers $Headers -Method Get

        # Update only the enabled field in the complete user data
        $CurrentUserData.enabled = $false
        $RequestBody = $CurrentUserData | ConvertTo-Json
    }

    try {
        Write-Host "Disabling user $($User.Username) (last login: $($User.LastLoginDate))"
        $UpdateResponse = Invoke-RestMethod -Uri $UserUrl -Headers $Headers -Method Put -Body $RequestBody
        Write-Host "User $($User.Username) disabled successfully." -ForegroundColor Green
    } catch {
        Write-Host "Failed to disable user $($User.Username): $_" -ForegroundColor Red
    }
```

Using Teamwork Cloud's REST API we can disable any user who has not logged in within the last 60 days

General Process:

1. Get all users via GET /osmc/admin/users?includeBody=true
2. Loop through each user
    a. Convert the lastLogin value from Unix Epoch to DateTime
    b. Check if lastLogin is within the 60-day window
    c. If NO, put that user in a list to disable
3. Loop through each user to disable
    a. Get the required user data
    b. Update the enabled field to false
    c. Update the user details via the PUT /osmc/admin/user

Optional Enahancements

- Set to automatically run at specified interval (e.g., weekly)
- Email users letting them know their account has been disabled

ENOLA

Example: Bulk User Creation

Populating a Teamwork Cloud server with a list of known users is easy with the REST API

```powershell
# Construct base URL and endpoints
$RESTBASEURL = "https://$Server:8111/osmc/"
$ADDUSERURL  = "$RESTBASEURL/admin/users"
$LOGINURL    = "$RESTBASEURL/login"
$LOGOUTURL   = "$RESTBASEURL/logout"

# Prepare Basic Authentication header using ASCII encoding
$pair       = "$Admin:$Passwd"
$base64Auth = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes($pair))
$headers    = @{ Authorization = "Basic $base64Auth" }

# Login to obtain a session
Try {
    $Session = New-Object Microsoft.PowerShell.Commands.WebRequestSession
    $loginResponse = Invoke-RestMethod -Uri $LOGINURL -Method Post -Headers $headers -Web
    Write-Host "Successfully logged in to Teamwork Cloud." -ForegroundColor Green
} Catch {
    Write-Error "Login failed: $_"
    exit
}

# Import the CSV file
# The CSV is expected to have headers: login, password, username, department, email
try {
    $users = Import-Csv -Path $UserFile
} catch {
    Write-Error "Failed to import CSV file: $_"
    exit
}

# Loop through each user record and create the user via REST API
foreach ($user in $users) {
    # Build JSON payload for user creation
    $payload = @{
        userName        = $user.login
        password        = $user.password
        otherAttributes = @{
            mobile     = ""
            name       = $user.username
            department = $user.department
            email      = $user.email
        }
        enabled         = $true
    }

    $jsonPayload = $payload | ConvertTo-Json -Depth 3

    Write-Host "Creating user: $($user.login)" -ForegroundColor Cyan

    Try {
        $response = Invoke-RestMethod -Uri $ADDUSERURL -Method Post -Body $jsonPayload -C
        Write-Host "User '$($user.login)' created successfully." -ForegroundColor Green
    } Catch {
        Write-Error "Error creating user '$($user.login)': $_"
    }
```

- Great for quickly standing up a test or demo servers!
- General Process:
  1. Get the server and input details from the user (hardcoded or via prompt)
     a. Username/Password (if not using other method)
     b. Server URL
     c. CSV file location
  2. Loop through the CSV file for each user
     a. Build required JSON payload for the user
        i. Username, Name, Department, Email
        ii. Password
        iii. Enabled (true/false)
     b. Create the user via the /osmc/admin/user
        i. Included JSON payload with user details

	A	B	C	D	E
1	hank.hill	password	Hank Hill	Strickland Propane	hank.hill@stricklandprop.com
2	peggy.hill	password	Peggy Hill	Tom Landry Middle School	margaret.hill@tlms.edu
3	bobby.hill	password	Bobby Hill	Tom Landry Middle School	robert.hill@tlms.edu
4	luanne.hill	password	Luanne Platter Kleinschmidt	Manger Babies LLC	manger_babies@yahoo.com
5	dale.gribble	password	Dale Gribble	Dale's Dead Bug	dale@deadbug.com
6	jeff.boomhauer	password	Jeff Boomhauer III	Texas Ranger Patrol Division	jeffrey.boomhauer@texasrang.gov
7	bill.dauterive	password	William Fontaine de La Tour Dauterive	U.S. Army	william.dauterive@us.army.gov
8	lucky	password	Elroy Kleinschmidt	Big Montain Fudge Cake	lucky@bmfc.band
9	kahn	password	Kahn Souphanousinphone	Systems Analyst	kahn.soup@sallc.com
10	minh	password	Minh Souphanousinphone		minsoup3233@gmail.com
11	connie	password	Connie Souphanousinphone	Tom Landry Middle School	connie.soup@tlms.edu
12	john.redcorn	password	John Redcorn	Big Montain Fudge Cake	john@bmfc.band

- Optional Improvements:
  - User Group creation and membership management
  - Common category creation

Example PowerShell script based on original import script written by Benjamin Krajmalnik

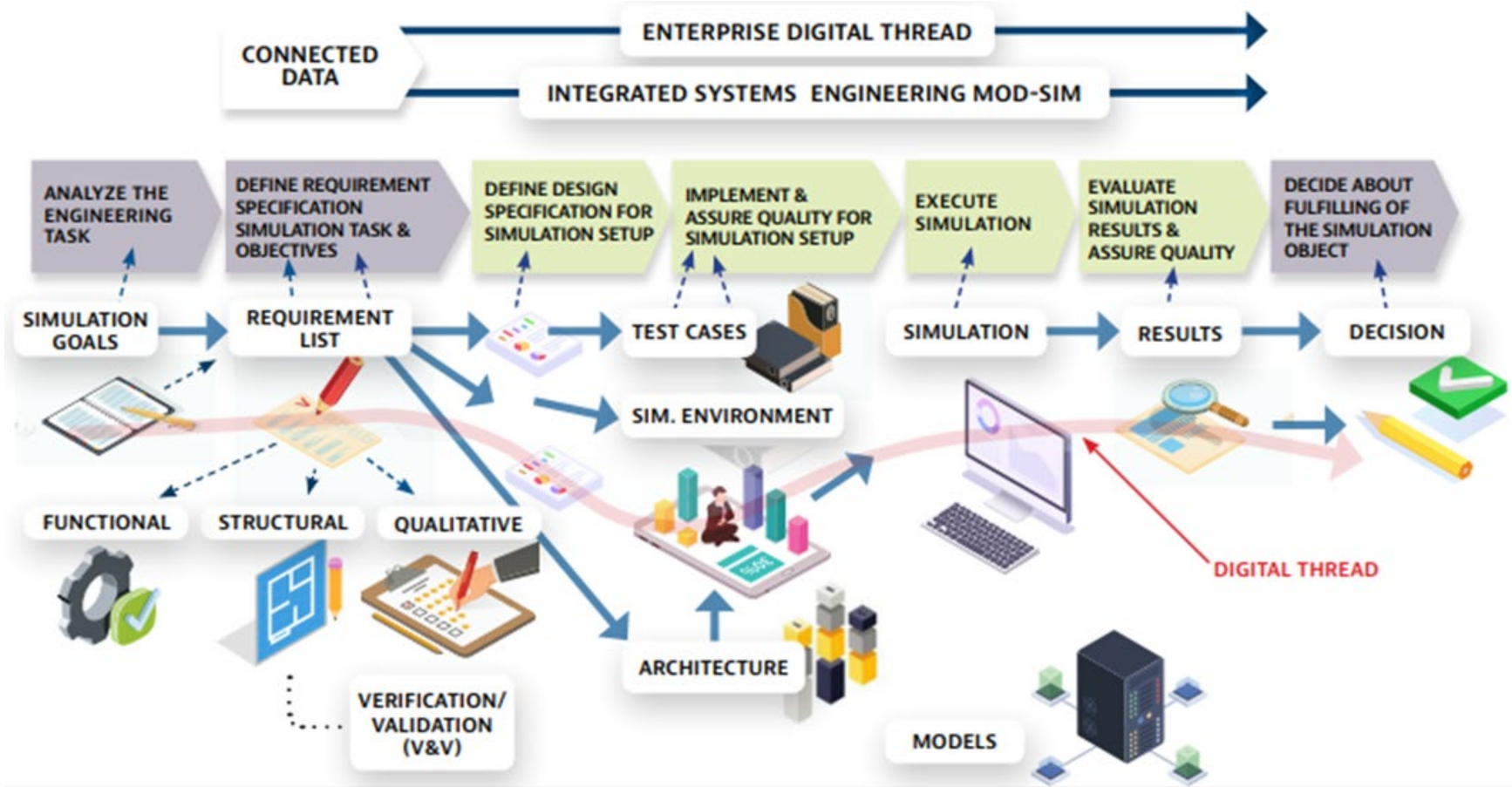
INTEROPERABILITY

ENOLA

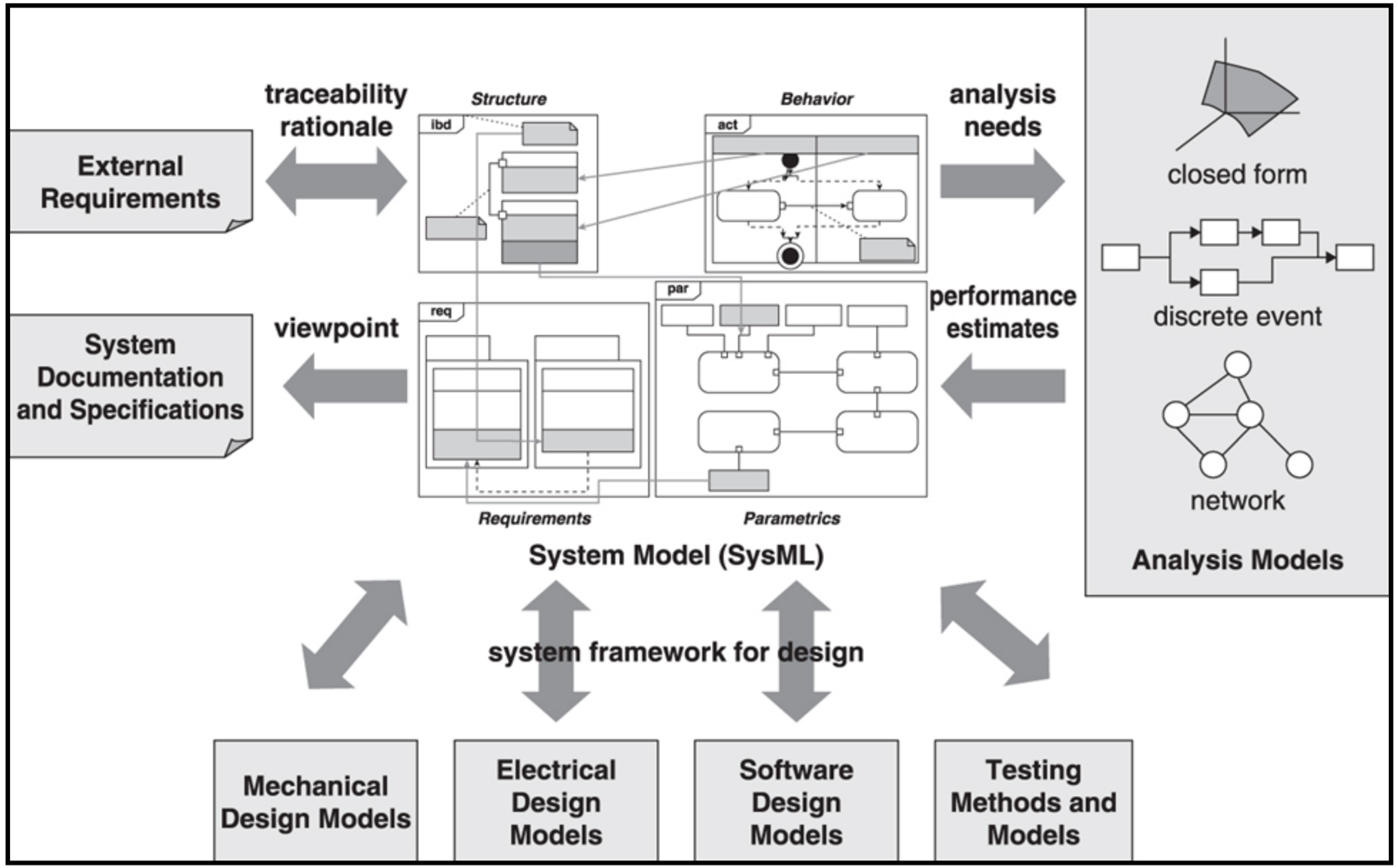
Interoperability and the Digital Thread

In the System Engineering Vision 2035, INCOSE states that

"*The Future of Systems Engineering is Predominantly Model Based*."



SysML as the Architecture

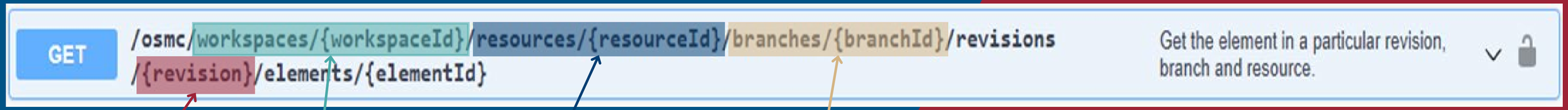


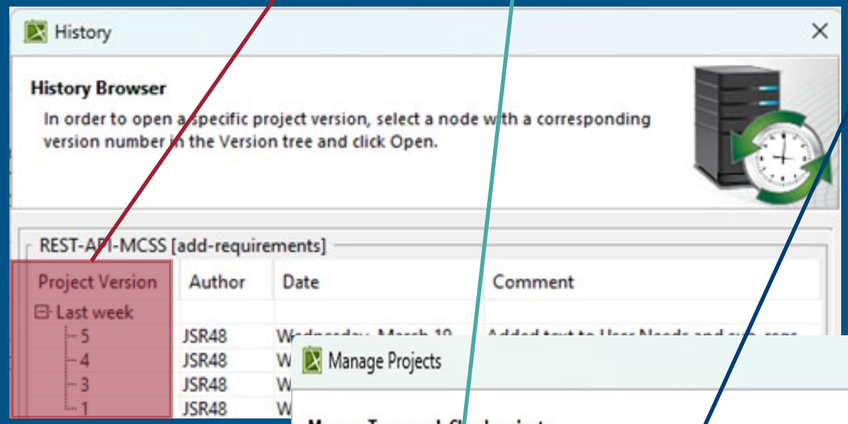
Background on REST API Payloads

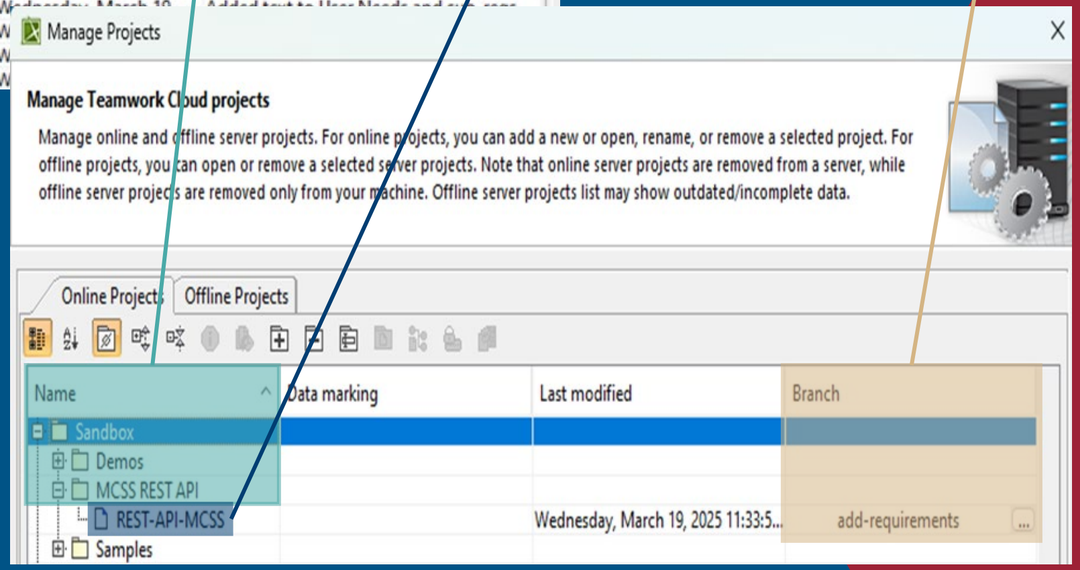
Data is returned from the server using a few different protocols as shown on the screenshot of a reduced payload to get information about a TWC Category

- *Linked Data Protocol - w3 consortium*
  -Defining structure for many of the project-level organization features
- *KerML model elements - OMG*
  -Element and relationship definitions
- *Dublin Core Terms - Dublin Core Metadata Initiative*
  -Specific metadata fields for defining objects
- *Eclipse Modeling Framework*

Navigating the Project Repository



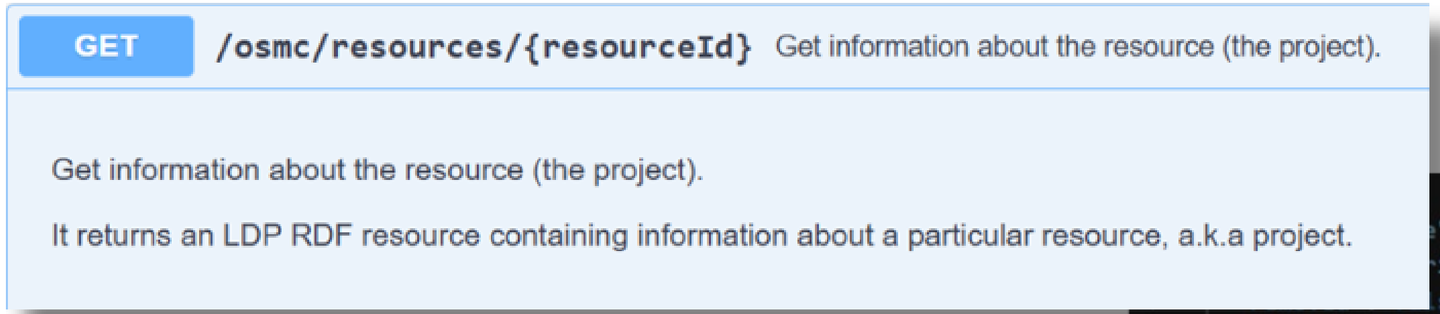




Using a “Drill Down” type of approach, analysts can find any data starting with a Category and/or a Project

The branch name and/or revisions can also be specified to find the exact snapshot from any time throughout the project’s history

Collecting Project Information

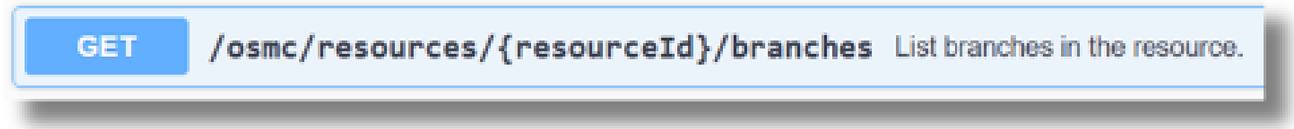



```
": "REST-API-MCSS.MASTER",
iption": "",
se,
"kerml:branches": "branches",
"metadata": {
  "HIDDEN_PACKAGES_AVAILABLE_3ddc84ee-953a-4b46-868a-5852a7b9f524_5": "false",
  "HIDDEN_PACKAGES_AVAILABLE_3ddc84ee-953a-4b46-868a-5852a7b9f524_4": "false",
  "name": "REST-API-MCSS.MASTER",
  "description": "",
  "local.cache.blob.id1-4+8": "ec52032c-0f69-453f-b2fd-f10a3a4868da",
  "isStandardProfile": "false",
  "PROJECT_ID": "PROJECT-b07b9d8a-d0dc-4b95-923a-f2720ce3bd92"
},
"categoryID": "09db8086-b27a-467e-b6bc-a8cfeea5092c",
"trunkID": "4c8bfd73-9d42-4860-b589-413e414a1a26",
"modifiedDate": 1742445234,
"createdDate": 1742440712,
"@type": "kerml:Resource",
"ID": "."
}
```


- Key information about the resource is clearly displayed
- Many convenient connection up and down the hierarchy are incldued by default
  - categoryID
  - trunkID
- *Note that projects include a ".MASTER" signifier in the title, where Collaborator documents include a ".CC" in the title*

ENOLA

The Need for Repetitive Calls



BE SURE TO LOGIN AND RUN ALL CALLS IN A SESSION TO AVOID USER LIMITS IN TEAMWORK CLOUD


```
{
  "@context": "https://ag-mxg-twc2024x.dsone.3ds.com:8111/osmc/schemas/branches",
  "@id": "",
  "@type": [
    "ldp:BasicContainer"
  ],
  "ldp:contains": [
    {
      "@id": "4c8bfd73-9d42-4860-b589-413e414a1a26"
    },
    {
      "@id": "fe879744-5fbd-48d0-a29a-0a608160754f"
    },
    {
      "@id": "3ddc84ee-953a-4b46-868a-5852a7b9f524"
    }
  ]
}
```


- The example payload to the left demonstrates the structure returned when querying for all branches in a project
- Note that this list of branches is returned as an LDP container and a JSON list of IDs are returned.
  - Since no additional data is known at this point, queries will need to be looped in order to collect further information
- Secondary calls to query each of these IDs will expose further information such as:
  - List of branch numbers
  - ID
  - Title
  - Parent branch ID
  - Author
  - latestRevision

Element Level Information

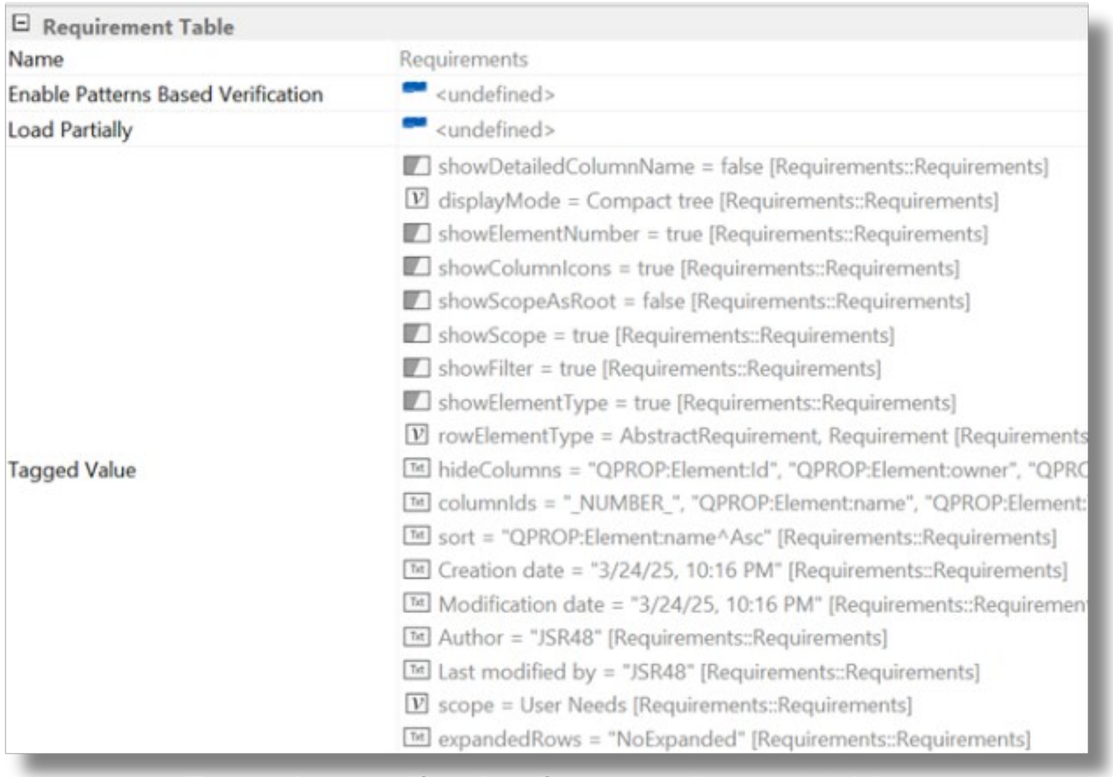
All elements exist with various types based on their database and model designations

- Packages - LDP Container + UML Package
- Part Properties = LDP Container + UML Property
- Blocks = LDP Container + UML Class
- IBD = LDP Container + UML Diagram
- Many, many more . . .

Programmers interacting with the API will need to have some level of UML/SysML knowledge to appropriately travers relationships and metadata fields built from these models

Every item in the model will have a server element ID which is reachable from the Specification window

Looking through the Specification window can offer great insight into the composition of different diagrams



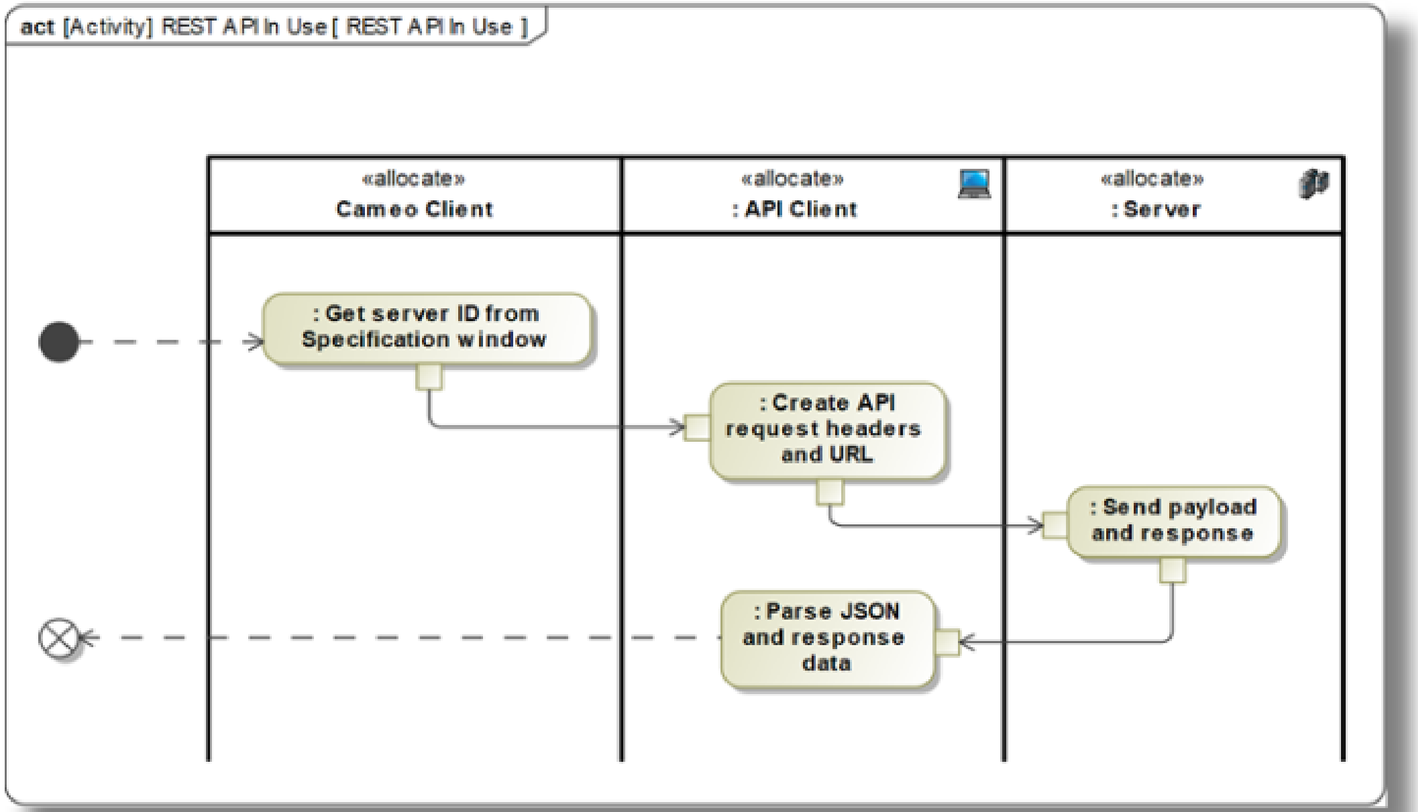
Example specification for a requirement table

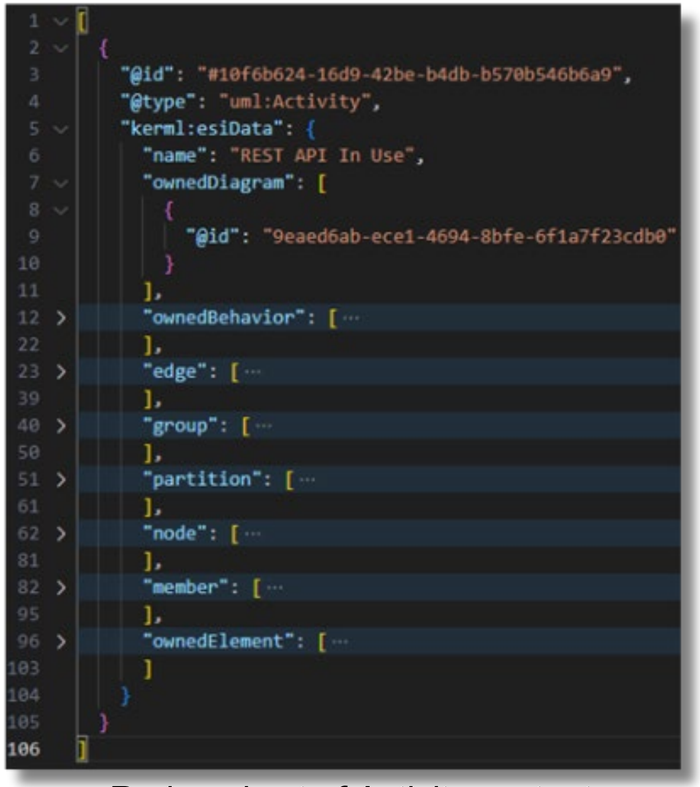
There are no direct relations to table entries within the specification and therefore the API.

Tables store query input information and would take significant effort to extract all information from the API.

Example: Activity from Cameo

GET /osmc/resources/{resourceId}/elements/{elementId} Get the element of the latest revision in the master branch of the resource.





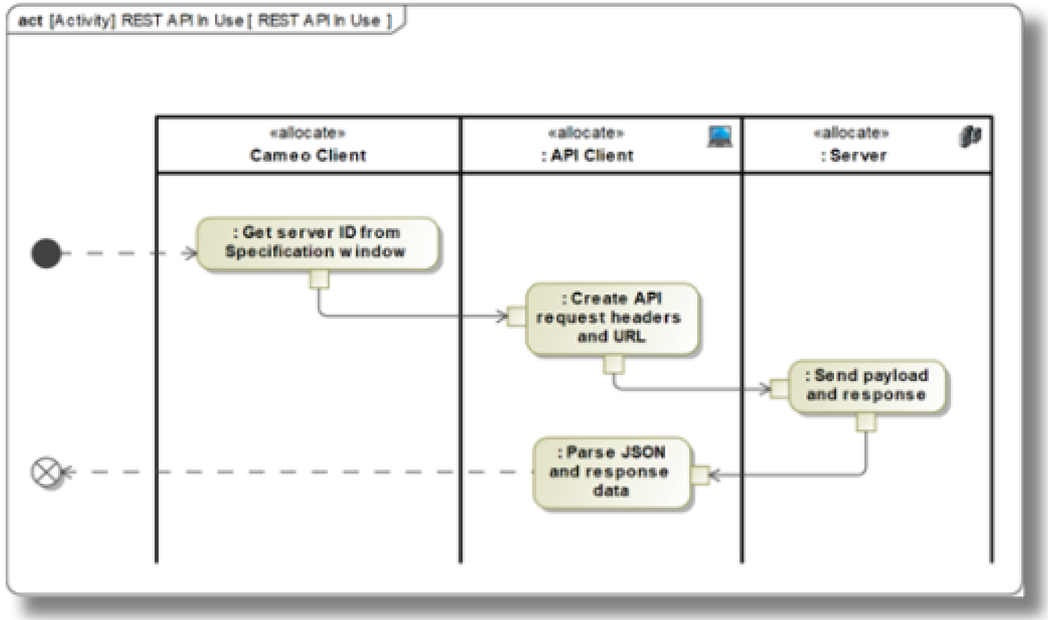
Reduced set of Activity contents

NOTE: THIS RESULT COMES FROM THE ACTIVITY ELEMENT, NOT THE ACTIVITY DIAGRAM!

Example: Activity from Cameo

When peeking into the element information from the lists of Nodes contained in the activity, Incoming and Outgoing flows can be collected

Node			
Type	ID	Incoming	Outgoing
uml:InitialNode	f95c4ad0-169b-4ba2-b643-bf6cdad76b24		94051f27-3386-43d2-af40-b221353d6ea1
uml:CallBehaviorAction	0400fe5c-42e7-47a4-83cb-8f271ba6bd5a	94051f27-3386-43d2-af40-b221353d6ea1	
uml:CallBehaviorAction	9333d585-b6bc-4ca4-ac32-13f071e97b5b		
uml:CallBehaviorAction	01401243-536f-4c69-bd75-d36de7319867		
uml:CallBehaviorAction	07f1ff51-5592-4b73-832b-d302a571584a		65092d2d-0e20-4047-ae43-2f7467df8a8a
uml:FlowFinalNode	a05b52c4-3883-4ebb-812c-d0ecd8248a54	65092d2d-0e20-4047-ae43-2f7467df8a8a	

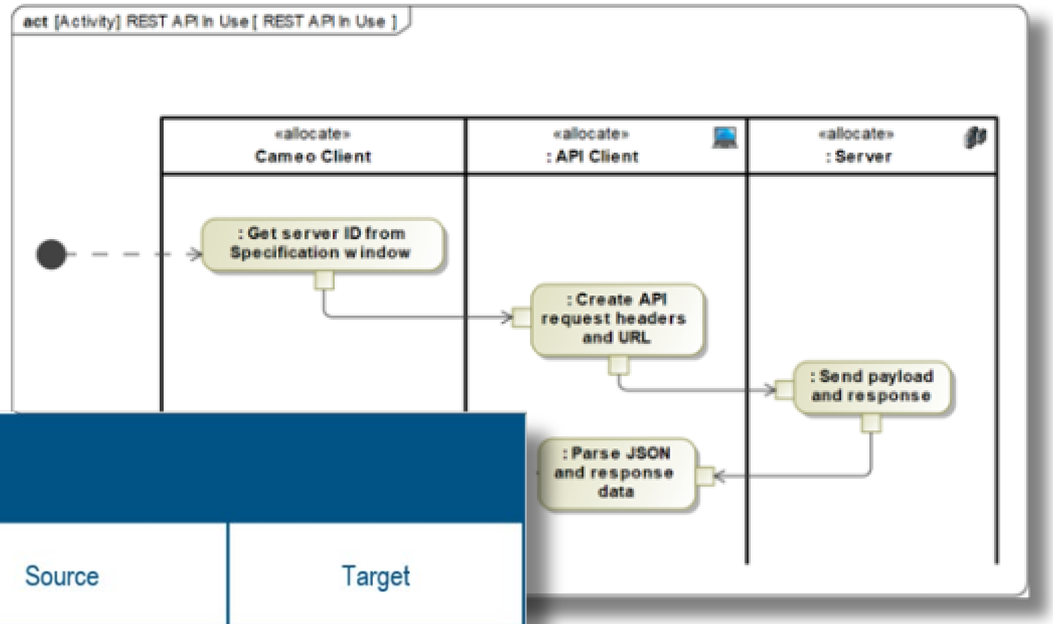



```
def getNodeInfo(payload):
    type = payload[1]["@type"]
    outgoing=''
    incoming=''
    try:
        outgoing = payload[1]["kerml:esiData"]["outgoing"][0]["@id"]
    except:
        pass
    try:
        incoming = payload[1]["kerml:esiData"]["incoming"][0]["@id"]
    except:
        pass
    return [type, incoming, outgoing]
```

WHERE ARE THE REST OF THE INCOMING AND OUTGOING FLOWS?

Example: Activity from Cameo

Next, a similar list of Edges in queried for more information. For Object and Control flows, Sources and Targets can be identified.



Node	
Type	ID
uml:InitialNode	f95c4ad0-169b-4ba2-b643-bf6cdad76b24
uml:CallBehaviorAction	0400fe5c-42e7-47a4-83cb-8f271ba6bd5a
uml:CallBehaviorAction	9333d585-b6bc-4ca4-ac32-13f071e97b5b
uml:CallBehaviorAction	01401243-536f-4c69-bd75-d36de7319867
uml:CallBehaviorAction	07f1ff51-5592-4b73-832b-d302a571584a
uml:FlowFinalNode	a05b52c4-3883-4ebb-812c-d0ecd8248a54

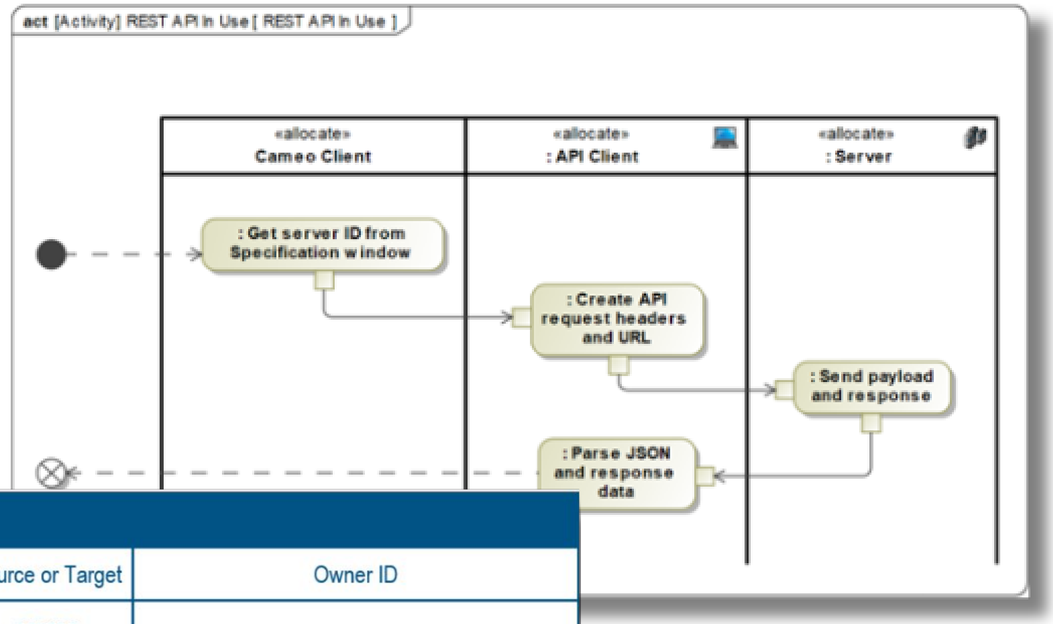
Edge			
Type	ID	Source	Target
uml:ControlFlow	94051f27-3386-43d2-af40-b221353d6ea1	f95c4ad0-169b-4ba2-b643-bf6cdad76b24	0400fe5c-42e7-47a4-83cb-8f271ba6bd5a
uml:ObjectFlow	d6920e51-b348-43cc-b6c1-b37516ef2cea	94f26243-c7f0-4748-aee6-cb8fb3716c0c	6db705a7-28ef-45e0-8103-ff0bebf2b8b0
uml:ObjectFlow	cf754ef4-6495-4f01-b9af-92669e4194a2	70b7f27f-eb28-4745-b888-f30da4f91622	0c68a137-b509-4087-878f-d466791f46b6
uml:ObjectFlow	413fb7c4-1bae-47ba-a05a-97980dd791ce	f3d30ba5-5db4-4846-a1ec-bba42da98bd9	2702b2fc-617c-4576-85ce-e9907311b405
uml:ControlFlow	65092d2d-0e20-4047-ae43-2f7467df8a8a	07f1ff51-5592-4b73-832b-d302a571584a	a05b52c4-3883-4ebb-812c-d0ecd8248a54

NODE TO NODE DIRECT FLOWS ARE CAPTURED

ENOLA

Example: Activity from Cameo

By then querying information from all of the edges in the previous slides (sources and targets), we can compile a true list of all of the control flows which travel through ports



Node			
Type	ID	Incoming	Outgoing
uml:InitialNode	f95c4ad0-169b-4ba2-b643-bf6cdad76b24		
uml:CallBehaviorAction	0400fe5c-42e7-47a4-83cb-8f271ba6bd5a		
uml:CallBehaviorAction	9333d585-b6bc-4ca4-ac32-13f071e97b5b		
uml:CallBehaviorAction	01401243-536f-4c69-bd75-d36de7319867		
uml:CallBehaviorAction	07f1ff51-5592-4b73-832b-d302a571584a		
uml:FlowFinalNode	a05b52c4-3883-4ebb-812c-d0ecd8248a54		

Pins			
ID	Type	Source or Target	Owner ID
f95c4ad0-169b-4ba2-b643-bf6cdad76b24	uml:InitialNode	source	
0400fe5c-42e7-47a4-83cb-8f271ba6bd5a	uml:CallBehaviorAction	target	
94f26243-c7f0-4748-aee6-cb8fb3716c0c	uml:OutputPin	source	0400fe5c-42e7-47a4-83cb-8f271ba6bd5a
6db705a7-28ef-45e0-8103-ff0bebf2b8b0	uml:InputPin	target	9333d585-b6bc-4ca4-ac32-13f071e97b5b
70b7f27f-eb28-4745-b888-f30da4f91622	uml:OutputPin	source	9333d585-b6bc-4ca4-ac32-13f071e97b5b
0c68a137-b509-4087-878f-d466791f46b6	uml:InputPin	target	01401243-536f-4c69-bd75-d36de7319867
f3d30ba5-5db4-4846-a1ec-bba42da98bd9	uml:OutputPin	source	01401243-536f-4c69-bd75-d36de7319867
2702b2fc-617c-4576-85ce-e9907311b405	uml:InputPin	target	07f1ff51-5592-4b73-832b-d302a571584a
07f1ff51-5592-4b73-832b-d302a571584a	uml:CallBehaviorAction	source	
a05b52c4-3883-4ebb-812c-d0ecd8248a54	uml:FlowFinalNode	target	

The owner of these ports can be mapped back onto the activity nodes

ENOLA

Summary: Activity from Cameo

1. Get all information from an Activity through an API query for the Activity Element ID

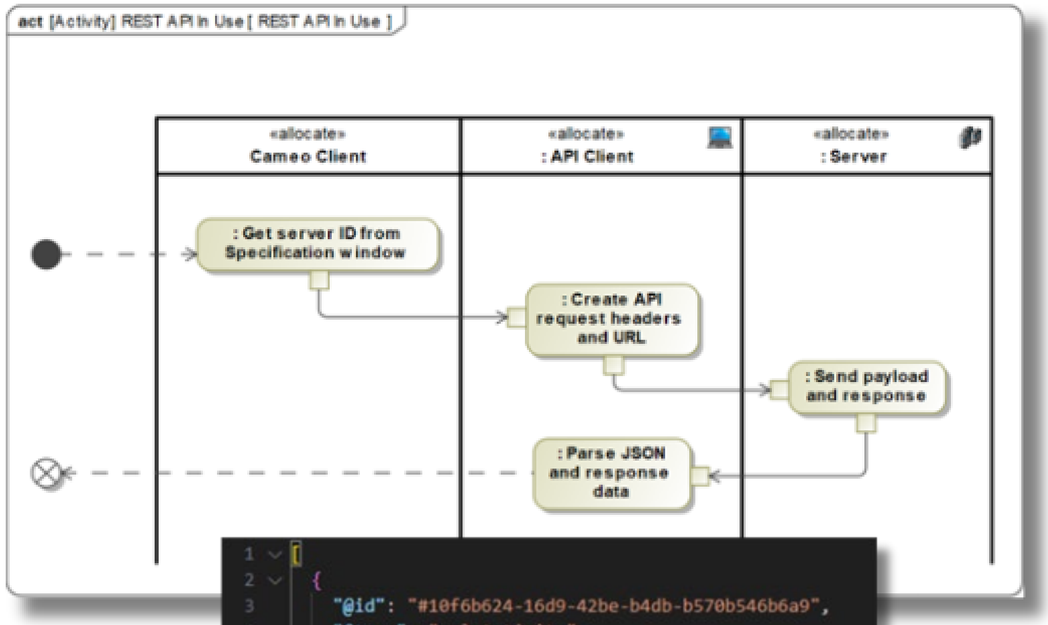
2. Query the API for information about all Nodes
   -Find all nodes and any flows which connect node-node

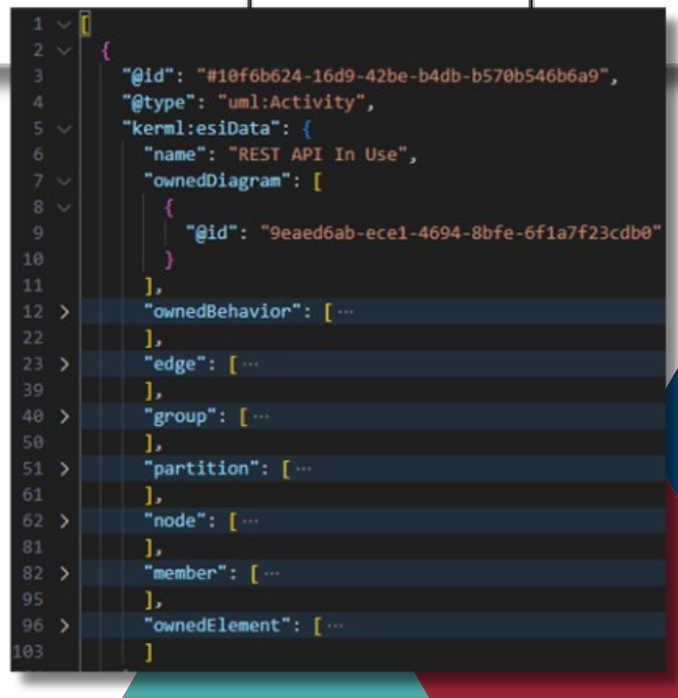
3. Query the API for information about all Edges
   -Find all flows which exist in the activity

4. Query the API for information about all pins
   -Developers must understand that object flows move through pins which are not included as items in the Activity
   -Build mappings between each pin and its owning node
   -This list of pins comes from the mappings built in step 3

5. Compile a final hierarchy of:
   -Nodes
   -Control flows between nodes
   -Pins owned by each node
   -Object flows between pins

Further queries can be completed to capture all swimlane allocation relationships, types of control flows, recursive expansion for sub -activity information, applied stereotypes, etc.

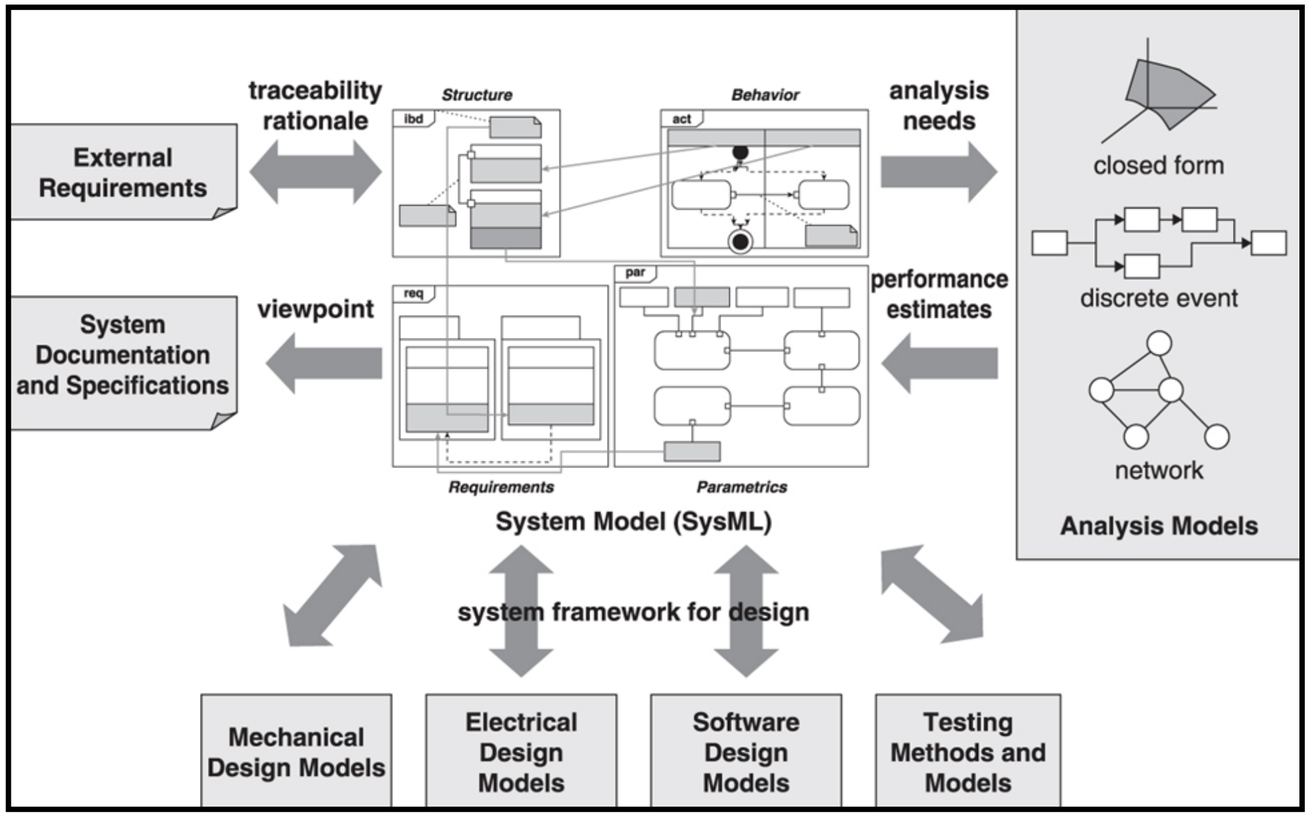




Extending this Approach

This case example only discussed the possibility of exporting Behavior as an activity diagram

Similar approaches can be employed to extract information about requirements, structure, and other architecture for a system



A Practical Guide to SysML 3rd Ed

This empowers developers to:

-Create skeletons for test cases and simulation workflows to be used in external tools

-Create documentation as code for detailed design teams

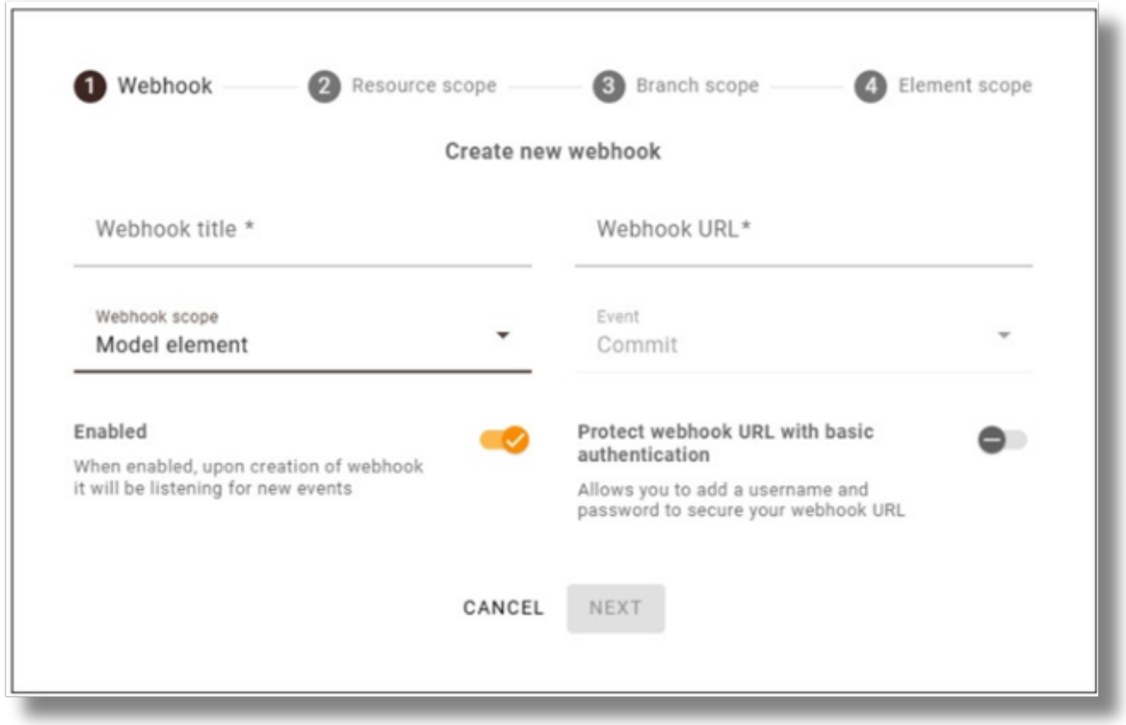
-Update models programmatically via model element updates

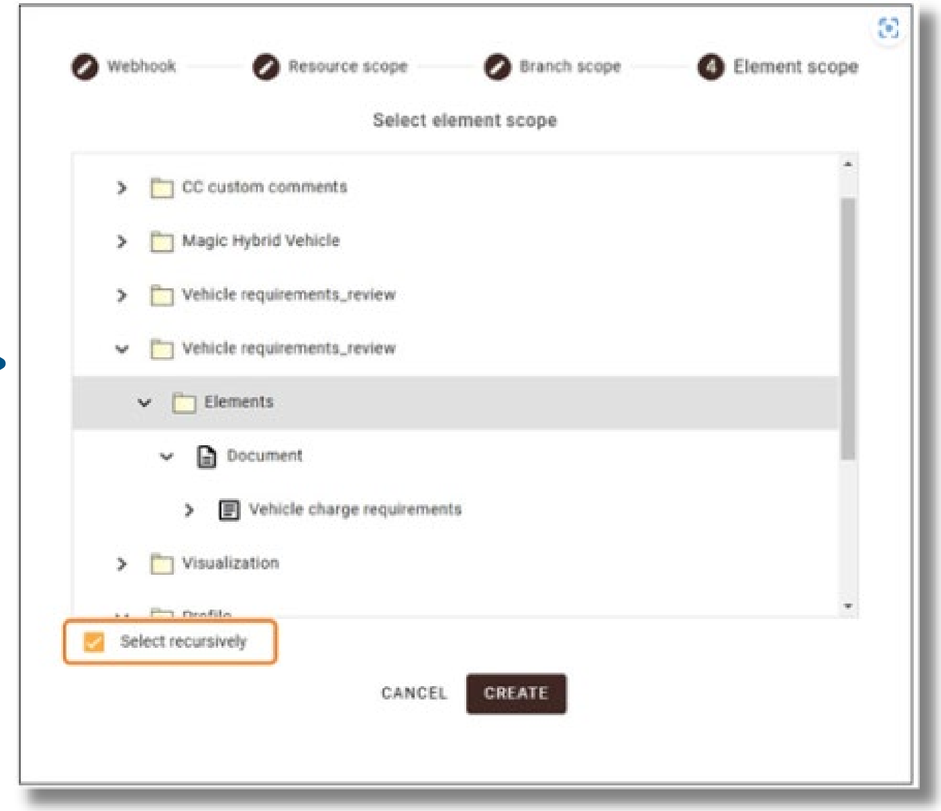
OTHER FEATURES & FUTURE WORK

ENOLA

Webhooks

“Webhooks allow you to get notifications pushed to a specified endpoint on predefined events in Magic Collaboration Studio instead of constantly polling for new data through REST APIs. You can listen to commit-type events in chosen resources or model elements and element-level events, such as commit or tagged commit.”





SysML v2 and REST API

With the release of SysML v2, a new specification for the Systems Modeling API and Services is also delivered.

Many capabilities are well aligned to the REST API shown in this presentation and through the TeamWork Cloud Swagger documentation.

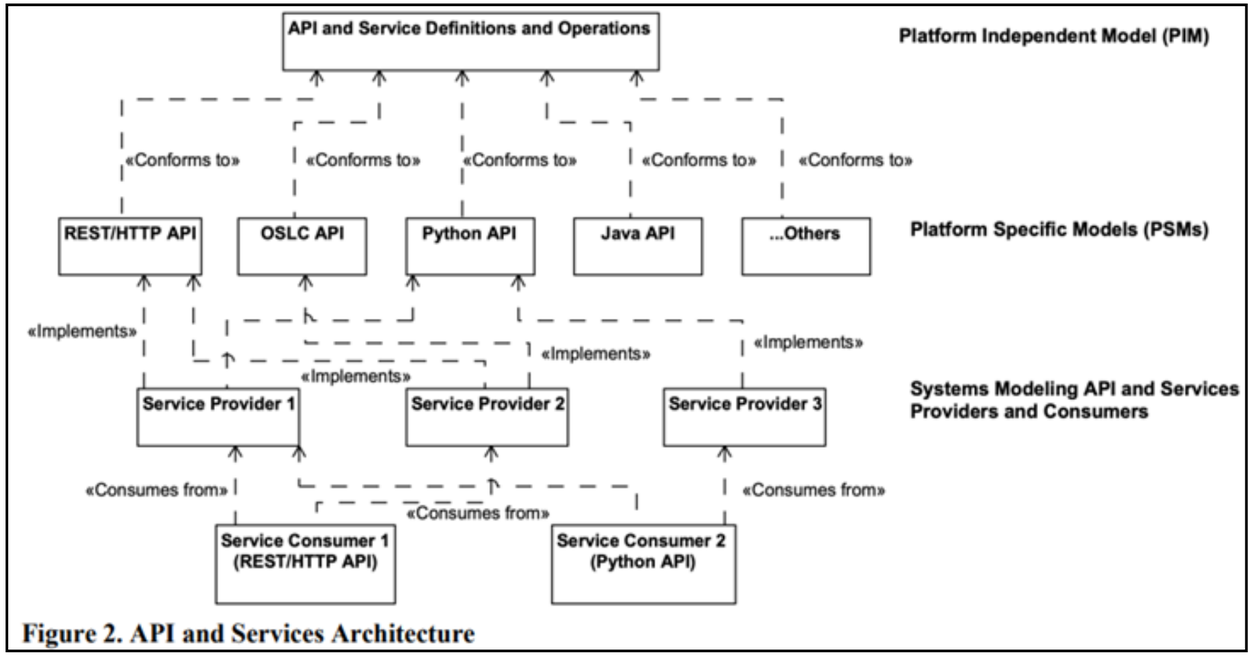


Figure 2. API and Services Architecture

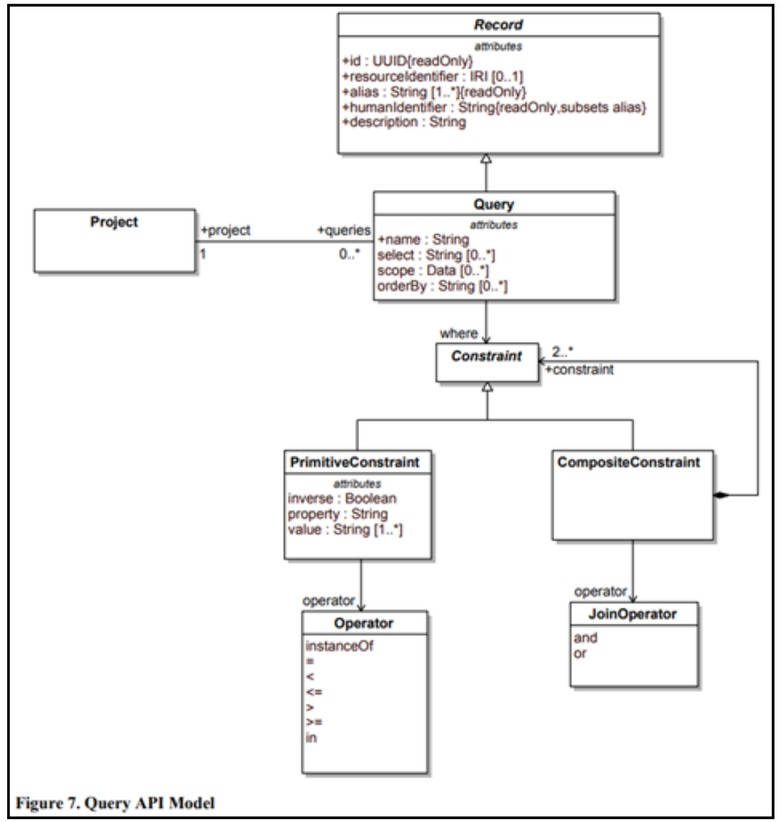


Figure 7. Query API Model

Addition of Query capability has potential to simplify many of the traversal issues for interoperability.

Model experts can specify the queries required, and the interface developers will be relieved of knowing detailed model structure.

Pageintation is also introduced for API efficiency.

SysML v2 and REST API

- For testing within the EEP, you can find the implemented Swagger at:https://<yourTeamworkCloudURL>:8443/sysmlv2-api/swagger-ui/index.html

- The SysML v2 release team is maintaining a cookbook as an Annex in the specification and a separate implementation via [GitHub]

Query to find all part usages

```
In [10]:
# Query vehicle1 (PartUsage) in the
query_input = {
  '@type':'Query',
  'select': ['name','@id','@type','owner'],
  'where': {
    '@type': 'CompositeConstraint',
    'operator': 'and',
    'constraint': [
        {
            '@type': 'PrimitiveConstraint',
            'inverse': False,
            'operator': '=',
            'property': '@type',
            'value': 'PartUsage'
        }
    ]
  }
}

payload = json.dumps(query_input)

vehicle1_id = ''
query_url = f"{host}/projects/{parts_tree_project_id}/query-results"

query_response = requests.post(query_url, json=query_input)
```

	Part Usage Name	Part Usage ID
0	rearAxleAssembly_c1	02fecf3d-69f7-4f16-a2c9-90c0df53dae5
1	frontAxle_c1	6184d9c5-57ae-4e29-906e-d042f7b7e2bb
2	frontWheel	005c0064-c3c2-4ecd-aec8-59483179e7e5
3	vehicle1_c1	dd0cba88-59dd-4b2e-9664-7608d22186ce
4	rearAxle	8ed3d0f9-1bb4-4207-80db-e39c5cfb1e5f
5	frontAxle	b53912c9-213d-4d61-b737-270969d20d65
6	vehicle1	149effb8-9cfb-4f55-80db-a984830eec55
7	rearAxle_c1	b4d0d0e0-8b79-4bea-a071-2af8315c492c
8	frontAxleAssembly	2b723540-a1cc-44e3-8f54-8b29cbb2962a


QUESTIONS?
ENOLA