ROSMOD: A Domain Specific Tool-suite (DSTS) for Distributed CPS

William Emfinger, Pranav Srinivas Kumar, and Gabor Karsai

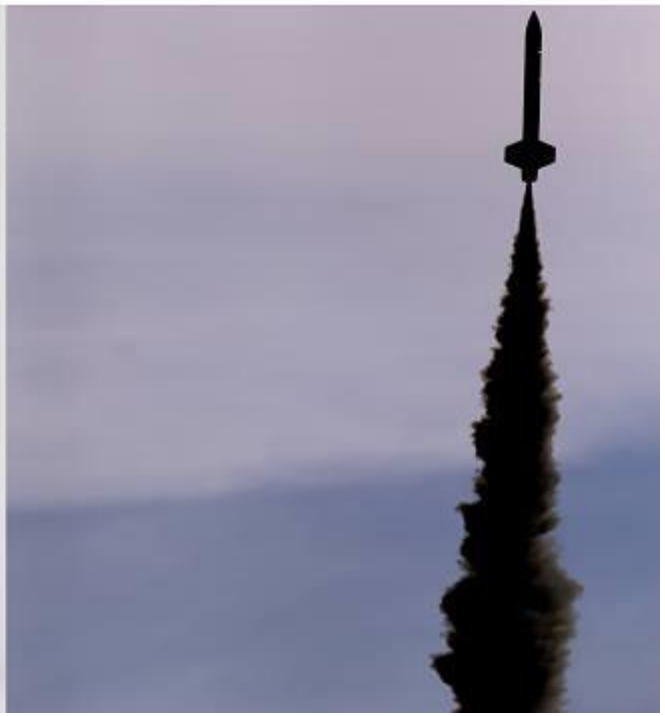
Kumar, P.S.; Emfinger, W.; Karsai, G.; Watkins, D.; Gasser, B.; Anilkumar, A. ROSMOD: A Toolsuite for Modeling, Generating, Deploying, and Managing Distributed Real-time Component-based Software using ROS. *Electronics* **2016**, *5*, 53.

This work was supported by DARPA under contract NNA11AB14C and USAF/AFRL under Cooperative Agreement FA8750-13-2-0050, and by the National Science Foundation (CNS-1035655). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, USAF/AFRL, or the NSF.

Outline

- Motivation
- What is ROSMOD
  - Architecture
  - Capabilities and features
  - Why use ROSMOD
- ROSMOD in Vanderbilt Aerospace Design Lab
  - AGSE
  - High Roller
  - VUSAT
- What does ROSMOD mean for small spacecraft
- What does ROSMOD lack
- Lessons learned

ROSMOD



Motivation

- Distributed CPS are hard to design, develop, analyze, deploy, and manage
- Integration of these key requirements into an IDE would make these processes
  - Easier to train / teach
  - Less error prone
  - Faster / more repeatable
- Some IDEs are resource-heavy and complicated to install / set-up
  - Require training as well
  - Need for maintenance to ensure proper versions and roll-out of updates / bug-fixes
  - Need to be cross-platform
- Not every system (or type of system) can or should conform to the same meta-model
  - Even within the same class of *Distributed CPS*
  - E.g. not everyone wants or needs the context of very explicit / fine-grained network specification

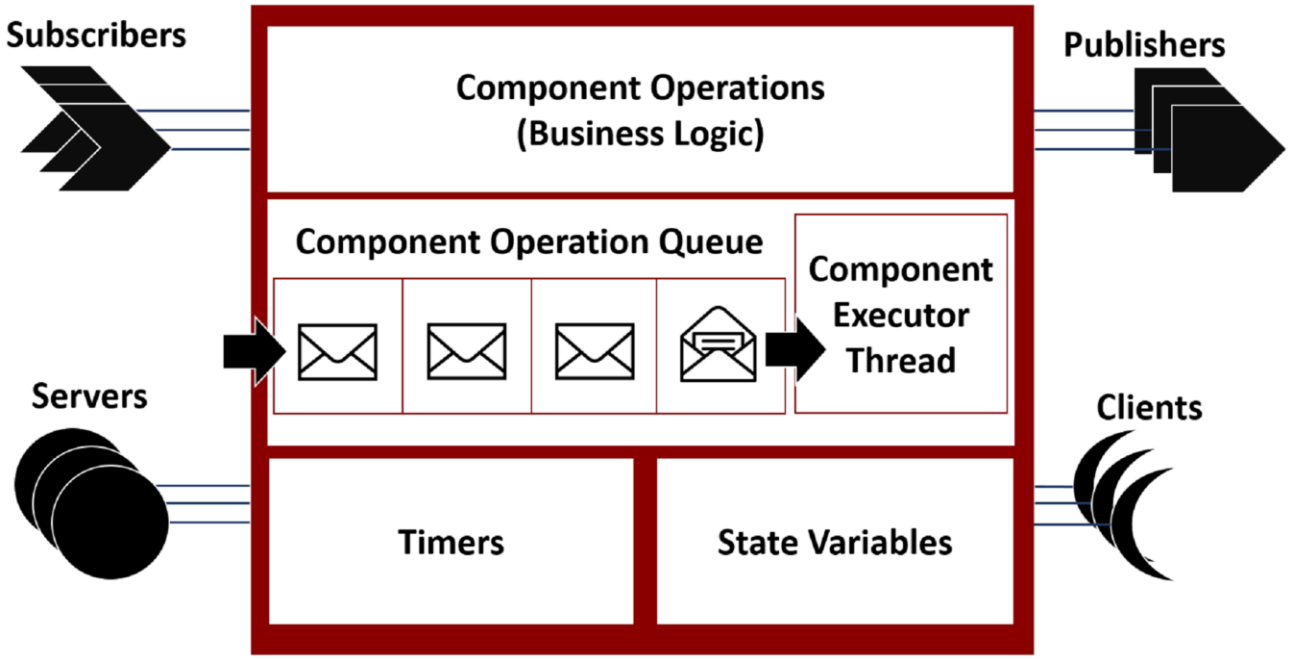
What is ROSMOD IDE?

- Web-based Integrated Development Environment
  - Built on top of the *Web Generic Modeling Environment* ***(WebGME)***
  - Graphical and textual software development for distributed CPS
- Automated
  - Code Generation / Build
  - Deployment
  - Analysis
- Extensible, modular tool-suite that allows users / developers to create or swap feature sets for different deployments of ROSMOD
  - You can integrate static code analysis tools, or online run-time monitoring frameworks

webgme.org
github.com/webgme

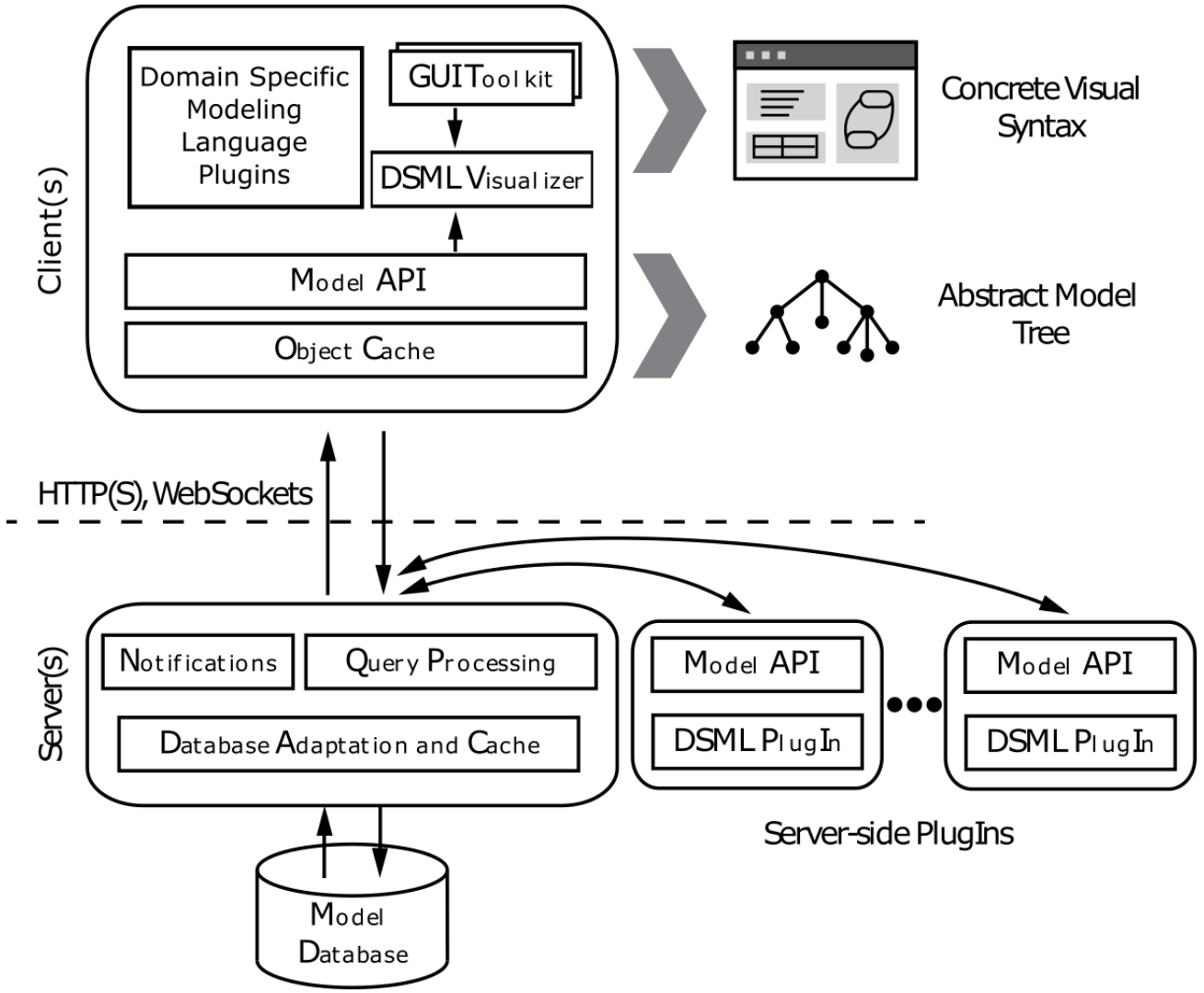
What is ROSMOD Run-Time?

- Extension to ROS run-time for better control over queuing and predictability of response times
- Provides a precise *component execution model* and implementation
  - Don't want to have to deal with mutexes and thread management
- FIFO, PFIFO, EDF queuing with deadline monitoring
  - Integration into design-time analysis
- Event tracing (optional)



ROSMOD : Architecture – WebGME Primer

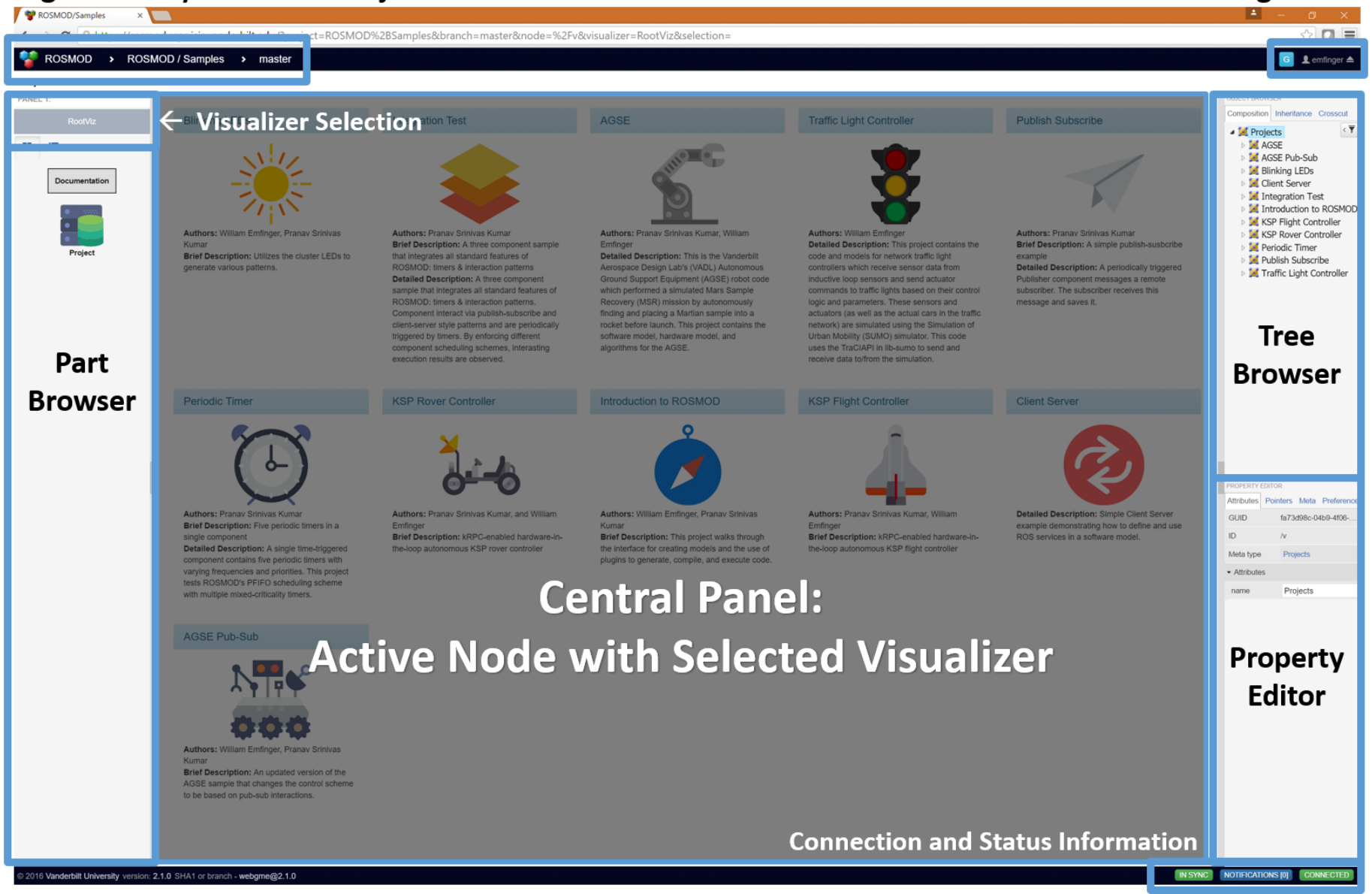
- Browser-based graphical (meta-) modeling
  - JS client with version management and distribution through npm / bower
  - NodeJS server side infrastructure
  - MongoDB database for models etc.
- Model edits are automatically versioned and allow
  - Tagging
  - Branching
  - Merging
- Component based design with customizable:
  - Plugins
  - Decorators
  - Visualizers
  - Layouts
- Organization / Project / User management with RWD control
- Associated tools facilitate generation of WebGME components
  - Good documentation helps!



ROSMOD IDE Specifics

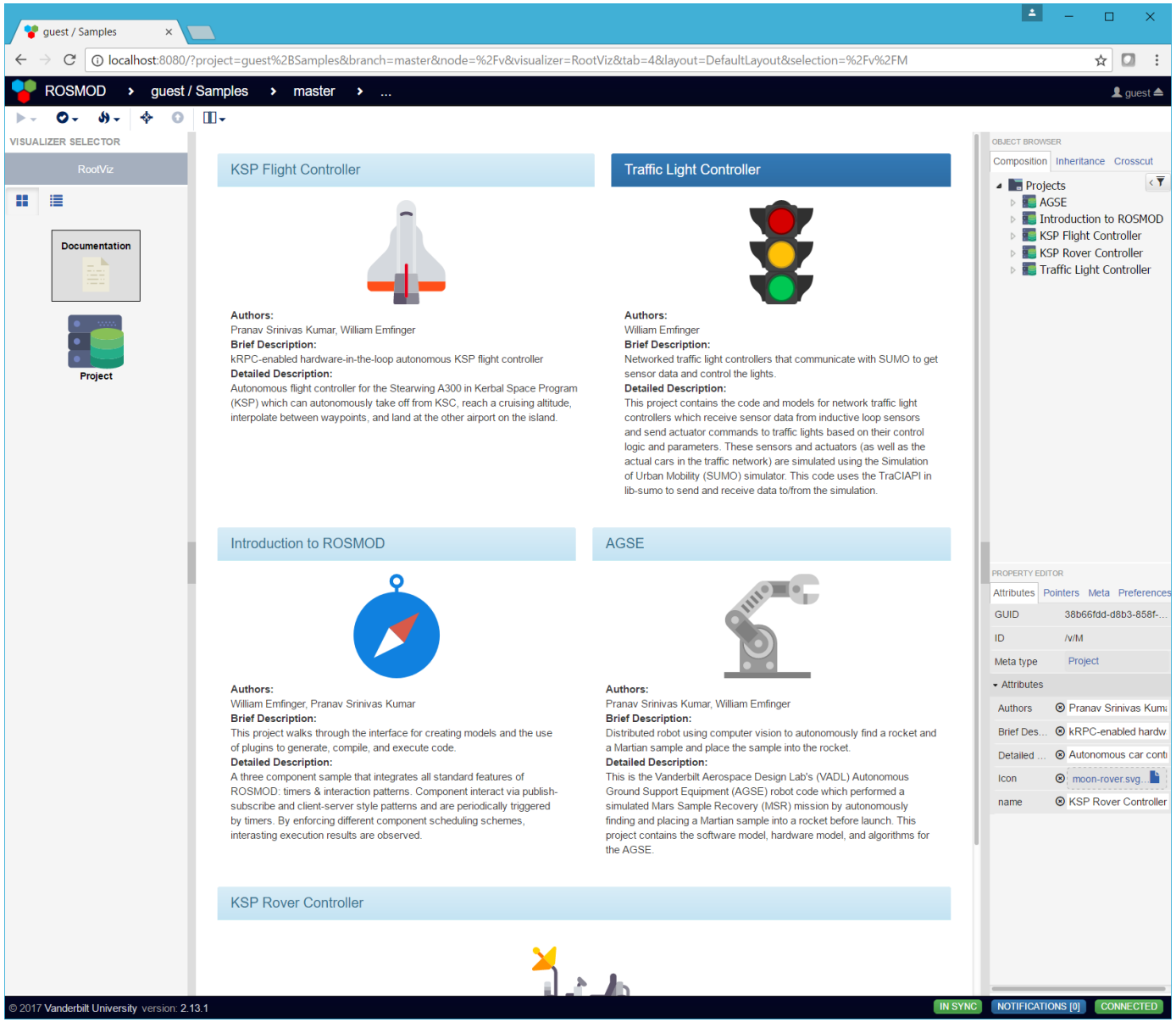
- Meta contains everything (we) needed to specify
  - Distributed, component-based software
  - Networked embedded systems
  - Deployments of software onto hardware
  - Experiment executions and their results
- Plugins enable
  - Code generation/compilation
  - Functional (timing / network) model analysis
  - Documentation generation
  - Experiment deployment / execution
  - Experiment teardown and results aggregation
- Visualizers enable
  - Project browser with relevant descriptions and identification
  - Deployment visual inspection
  - Execution trace and user log visualization

ROSMOD / WebGME Interface



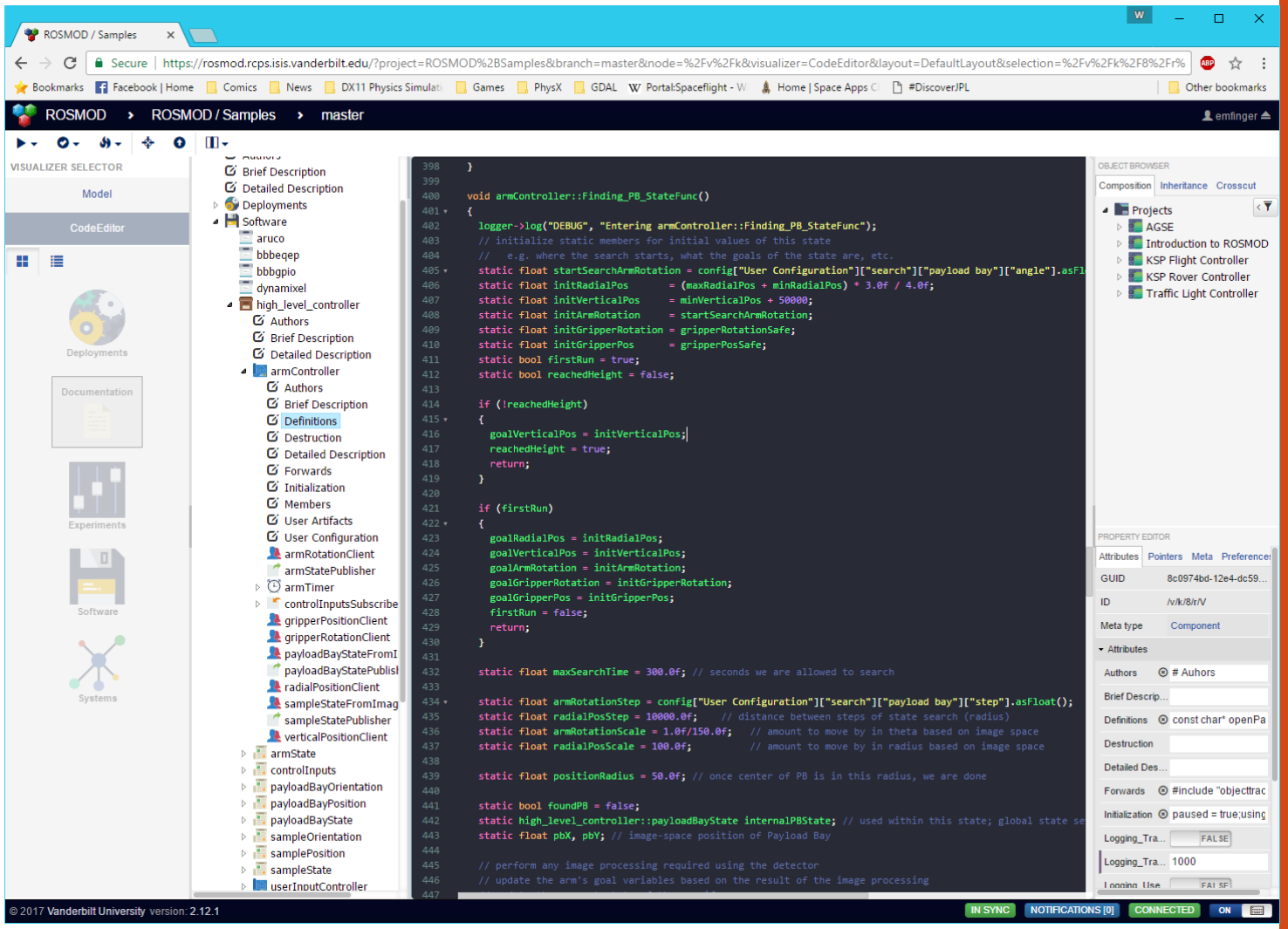
ROSMOD Project Browser

- Allows users to easily navigate and select on which project they want to work
- Acts as a landing page for users
  - Provides better interface than just a selection of nodes in a tree browser with no other context
  - Easy visual search / identification of projects



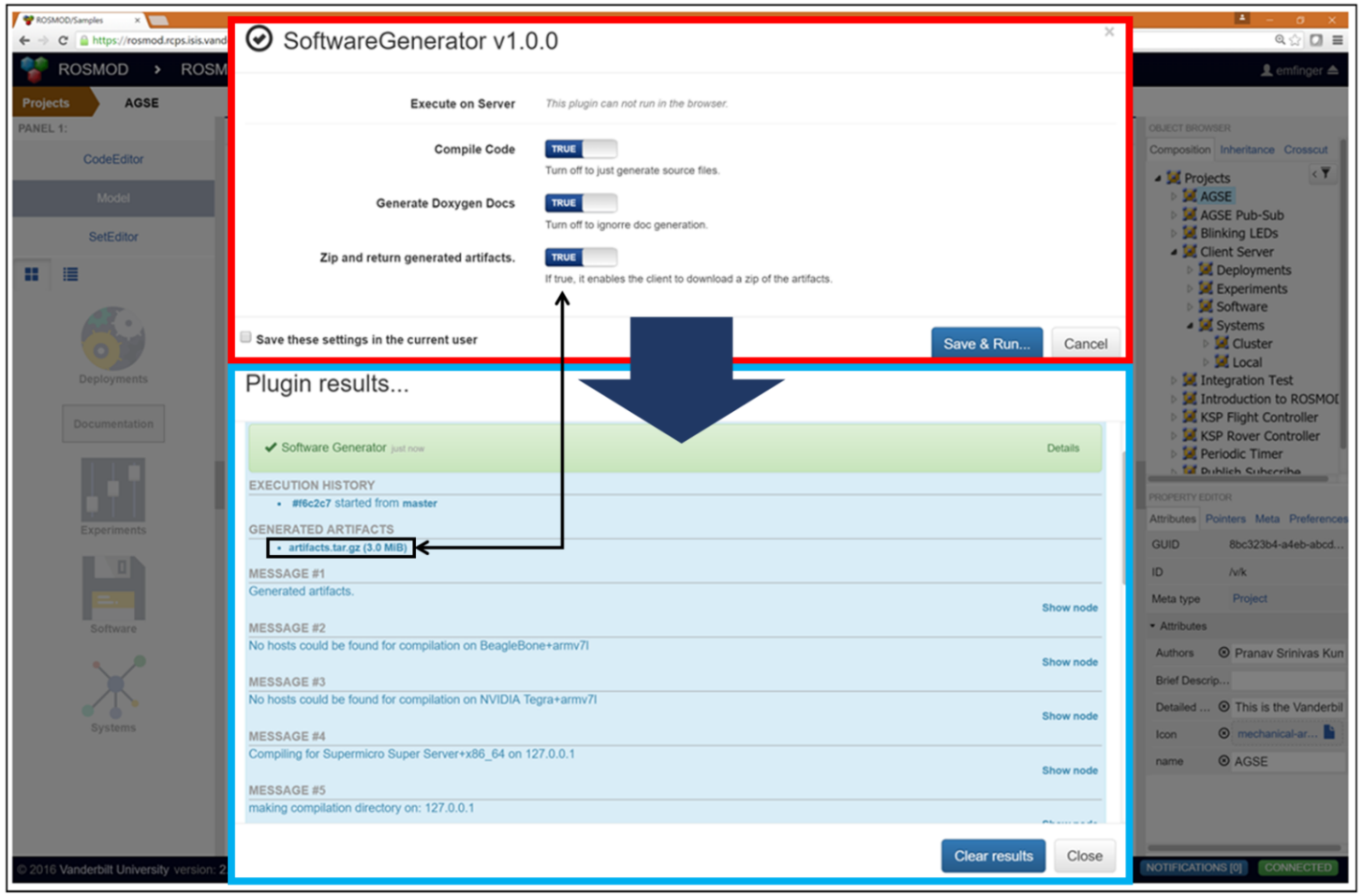
ROSMOD CodeEditor

- Code Editing in the browser; need features of an IDE:
  - Syntax highlighting
    - For multiple (including custom) languages
  - Code folding
  - Theming
  - Keybindings
  - Code Browser
  - (Some) code-completion
  - (Some) linting



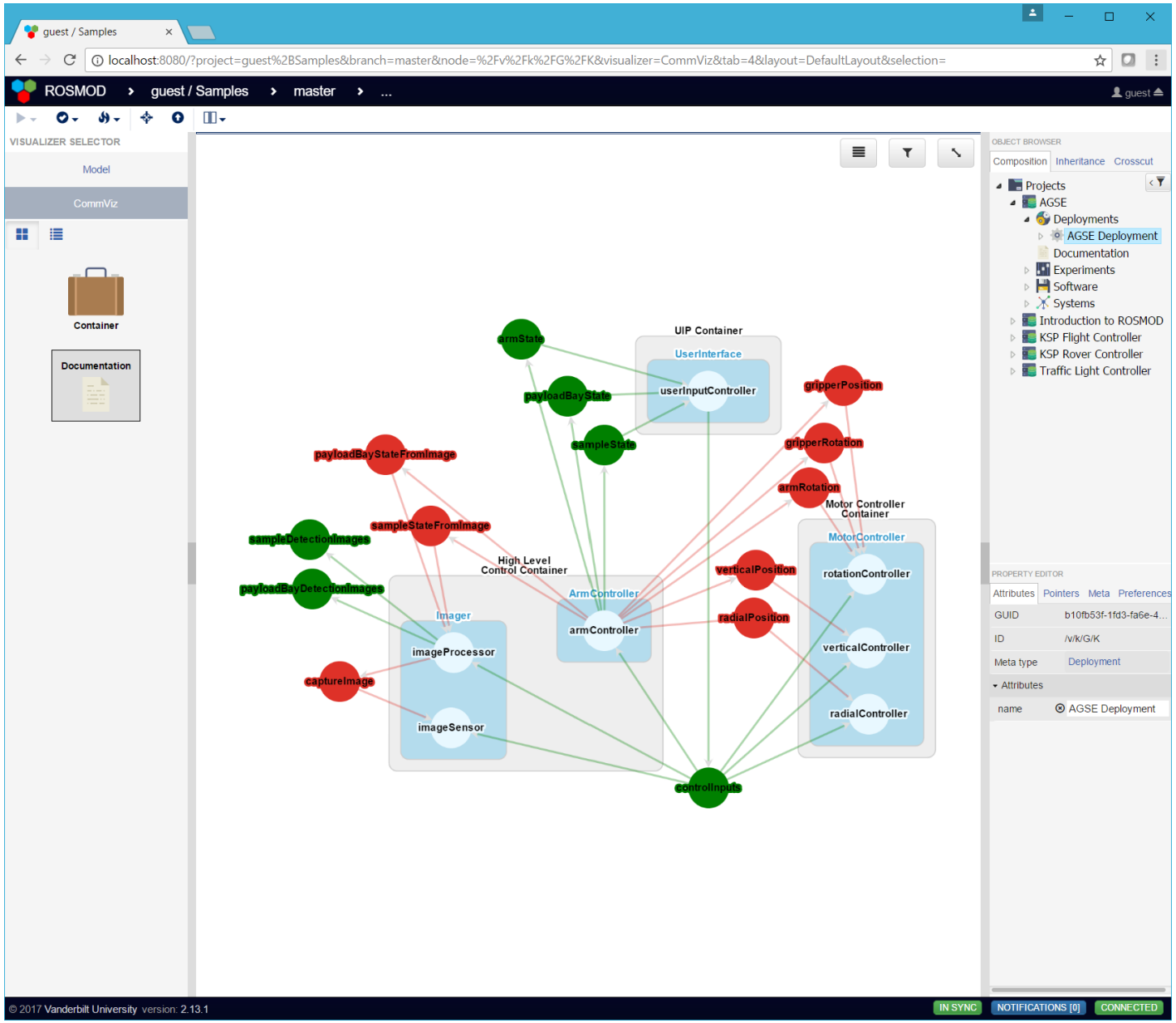
ROSMOD Code Generation / Compilation

- All the code is either
  - Contained within the model,
  - Generatable from the model, or
  - Located in a repository as part of a library which is referenced in the model
- This means that users don't have to touch the generated code
  - Large quality of life improvement over previous systems we've used in the past
- The compilation runs on the server; means
  - Users don't have to install or manage the compilers or their dependencies
  - Updates to the compilers can be managed in a centralized fashion by sys-admins



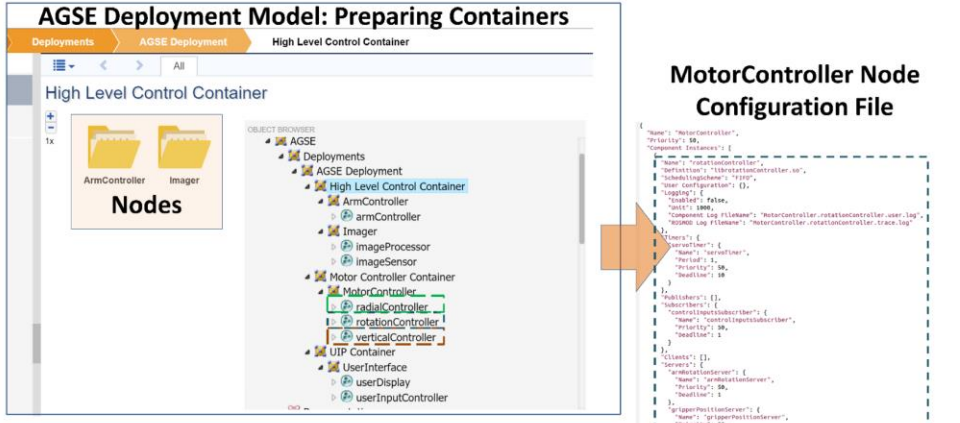
ROSMOD Deployment Visualization

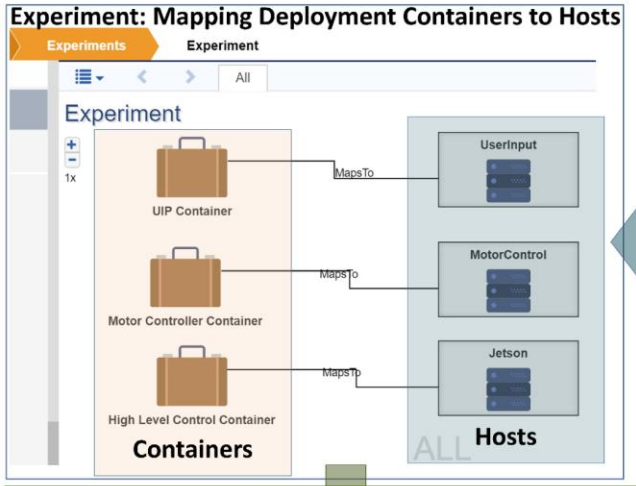
- Users want to know what configuration is actually running in a deployment
  - Since not all software components may be used, and the components in use may not be correctly configured
- The actual deployment may be large and difficult to visualize with many connections
  - So again, interactivity is key
- Being able to let the user select their current *context/focus* is important as the scale of the models/systems increases
  - Clicking on an object focuses its 1-hop neighbors and organizes them

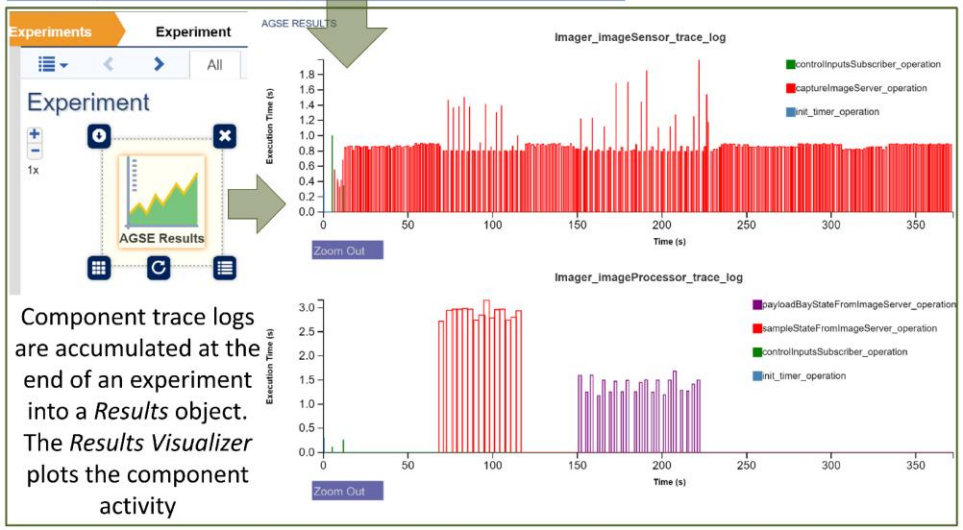


ROSMOD Experiment Execution

- Like the software generation/compilation, must run on the server, since it actually moves the binaries and configuration files over to the distributed systems
- Automatically queries the systems described in the model to determine which have available resources for running the experiment
- Updates the model to create a map that the user can see (and that the other components can interact with) which specifies the exact mapping that the plugin calculated
  - User who starts the experiment may leave, and another user may need to stop it
- When Experiment is stopped, the map is removed from the model and the results of the experiment are returned to the user and saved in the model

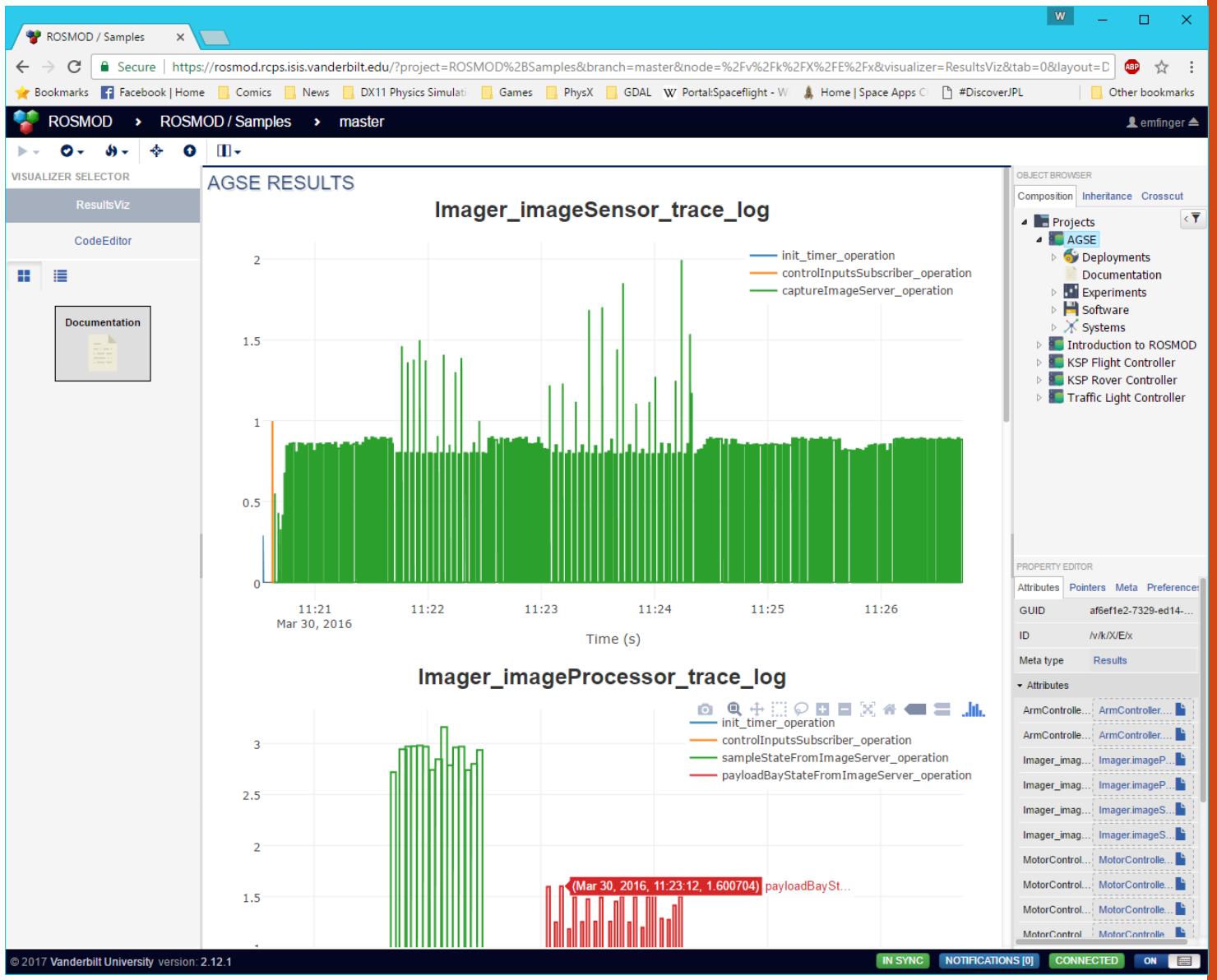






ROSMOD Experiment Results

- Results Visualization is important for distributed systems
  - If users have to look through tons of text logs from different processes on different nodes, they will not use your system.
  - Visualization lowers the difficulty and time it takes to find execution errors in your code/system
- Interactivity is key
  - Static plots look nice, but have limited utility when actually analyzing or debugging the system.
  - Need easy methods for users to massage the plots/data into something more meaningful for their current context.
    - Remove extra plots / data
    - Zoom x/y/x&y
    - Pan



ROSMOD : Capabilities & Features

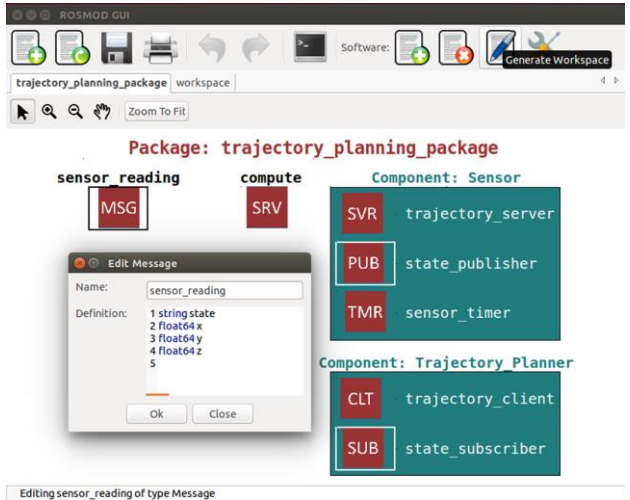
- Instant client setup, client computers only need
  - Network access to the ROSMOD server
  - Supported web browser
- Easy to migrate context between computers
  - Develop on computer at your desk, then log in and pull up the same model view in the lab where your experiment hardware is
- Integrated automatic version control (Thank you WebGME!)
  - Branching with integrated merge tool
  - Tagging
  - User-tagged commits (with optional squashing and commit messages)

Vanderbilt Aerospace Design Lab (VADL)

- Small group of undergraduate seniors in engineering (mainly Mechanical Engineering) with a few graduate students
- Competing since 2007 in the NASA Student Launch Competition
  - Design, build, and fly low-altitude rockets with novel scientific or technological payloads
  - Short design cycle (6-8 months), with PDR, CDR, FRR, LRR by NASA
  - 300 Page design review reports
- Honors
  - 4 National Championships
  - 7 Payload Design Awards
  - 3 Educational Engagement Awards
  - 2 Project Review Awards

ROSMOD in VADL (2014-2015)

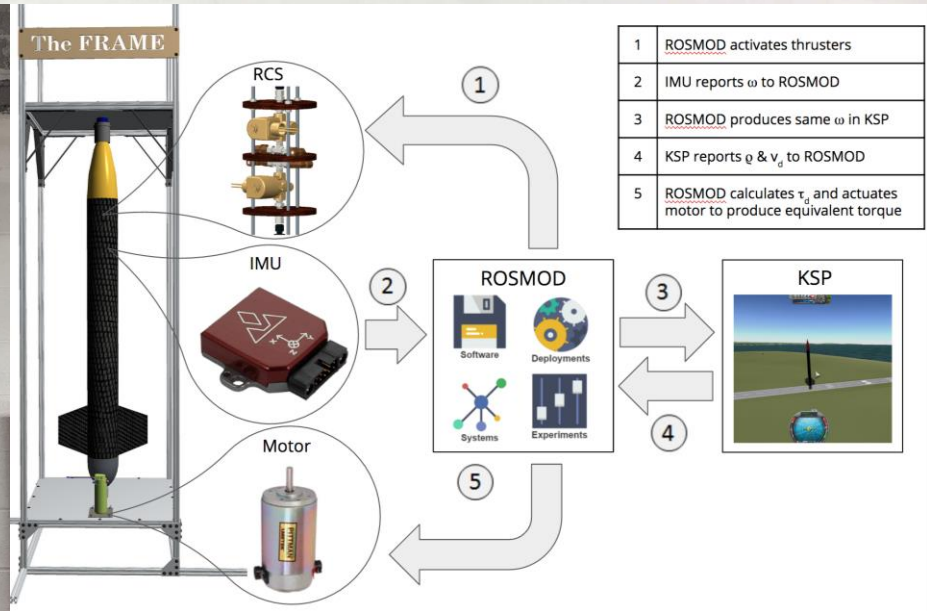
- Mars Autonomous Ground Support Equipment (AGSE)
  - Low Level Feedback Control
  - High Level Planning
  - Computer Vision
- Earliest implementation of ROSMOD
  - Originally published in *IEEE Symposium on Rapid Systems Prototyping, 2015*
- Rocket, Robot, and ROSMOD all developed concurrently over 6 months by 7 undergraduates and 4 graduate students



Original ROSMOD GUI (written in Python with wxPython)
… we don't talk about the tkinter version.

ROSMOD in VADL (2016-2017)

- High Roller
  - Cold-gas thruster roll control system in a low-altitude rocket
- Frictionless Roll Actuation and Modeling Environment (FRAME)
  - Ground-based testing environment for control systems and thruster actuation
- Hotbox
  - Temperature controlled environment for curing carbon-fiber
- Designed and built by undergraduates



All built with the (mostly) latest ROSMOD ☺

VADL Satellite Initiative (VUSAT)

- University funded student immersion project to develop a small satellite for earth-based sensing projects
  - Building upon the framework we developed with the Student Launch program
  - 1 hour course for 20 students (across disciplines)
  - Starting this fall (2017)
- Studying changes in the contemporary world combining perspectives from
  - Engineering
  - Earth Science
  - Anthropology
  - Physics
  - Economics
  - Political Science
- Partnering with NASA JPL (hopefully ☺), NASA Goddard, and ISRO
  - ISRO will develop sister satellite and be involved with design process
- Continuing ROSMOD's development and extending its use in both education and satellite development

What does ROSMOD mean for small spacecraft? (1/2)

- Standardization has been great for nano-sats
  - Cubesat platform provides a component-based design for the hardware which enables
    - Lower barrier to entry (need more student involvement!)
    - Shorter design cycles
    - Easier integration into testing facilities and launch vehicles
- But software development is still fairly custom
  - Especially as you get to larger spacecraft
  - Management software / tools are custom and their interfaces vary wildly
  - Adds overhead to development, testing, review, and management
- How to start adoption of more standardization?
  - Lower barrier to entry for such standardization
  - Teach and train early ☺

What does ROSMOD mean for small spacecraft? (2/2)

- ROSMOD's infrastructure / meta-models can be adapted for different run-time frameworks
  - Not dependent on ROS, Linux targets, etc.
- Centralization of infrastructure means more complex tools can be integrated *without burdening the users*
  - E.g. design-time V&V
  - Doesn't affect client configuration / setup ***at all***
  - Provides easier migration to (pseudo) cloud-based tooling for legacy tools
- Not all space systems are the same; can provide some standardization of interface / environment without locking down to a standardized run-time infrastructure
  - Enables migration towards more reusable software and hardware components for spacecraft systems
- Can improve student involvement at the undergraduate and advanced high-school level

What does ROSMOD lack?

- Better context for code completion
  - Including some measure of code linting or analysis before compilation
- Run-time debugging support
  - Can't rely on just trace logs and exception handlers when debugging CPS
  - GDB can be integrated, just requires development
- Some interfaces / visualization for developing the models
  - Component-based design requires some level of abstraction over the actual problem being solved
  - Need a good process for developing abstract / generic components from a specific problem
- Complex components
  - Dynamic timer periods, complex state-machines (coming soon!)
- Better support for sharing these reusable components
  - Currently possible, but inelegant
  - Improving thanks to WebGME development

Lessons Learned

- Need *appropriate* levels of abstraction
  - Especially when teaching to undergraduates
  - Process of converting hard-code into component interfaces
    - E.g. messages / services / user config
- ROSMOD / WebGME platform easy to develop extensions for
  - No prior JS / NodeJS experience required!
  - Build on large library of existing html / css / js / nodejs projects
- ROSMOD can be taught successfully and then used to develop robust ground-based and flight systems
- Collaborative, versioned development lowers the barrier to entry and expedites adoption
  - Enables on-line teaching / oversight
  - Eases the fear of messing up without adding the learning curve of versioning tools

Thank you!

Questions?

ROSMOD Org: *github.com/rosmod*

ROSMOD Run-Time: *github.com/rosmod/rosmod-comm*

ROSMOD IDE: *github.com/rosmod/webgme-rosmod*

VADL Org: *github.com/vadl*

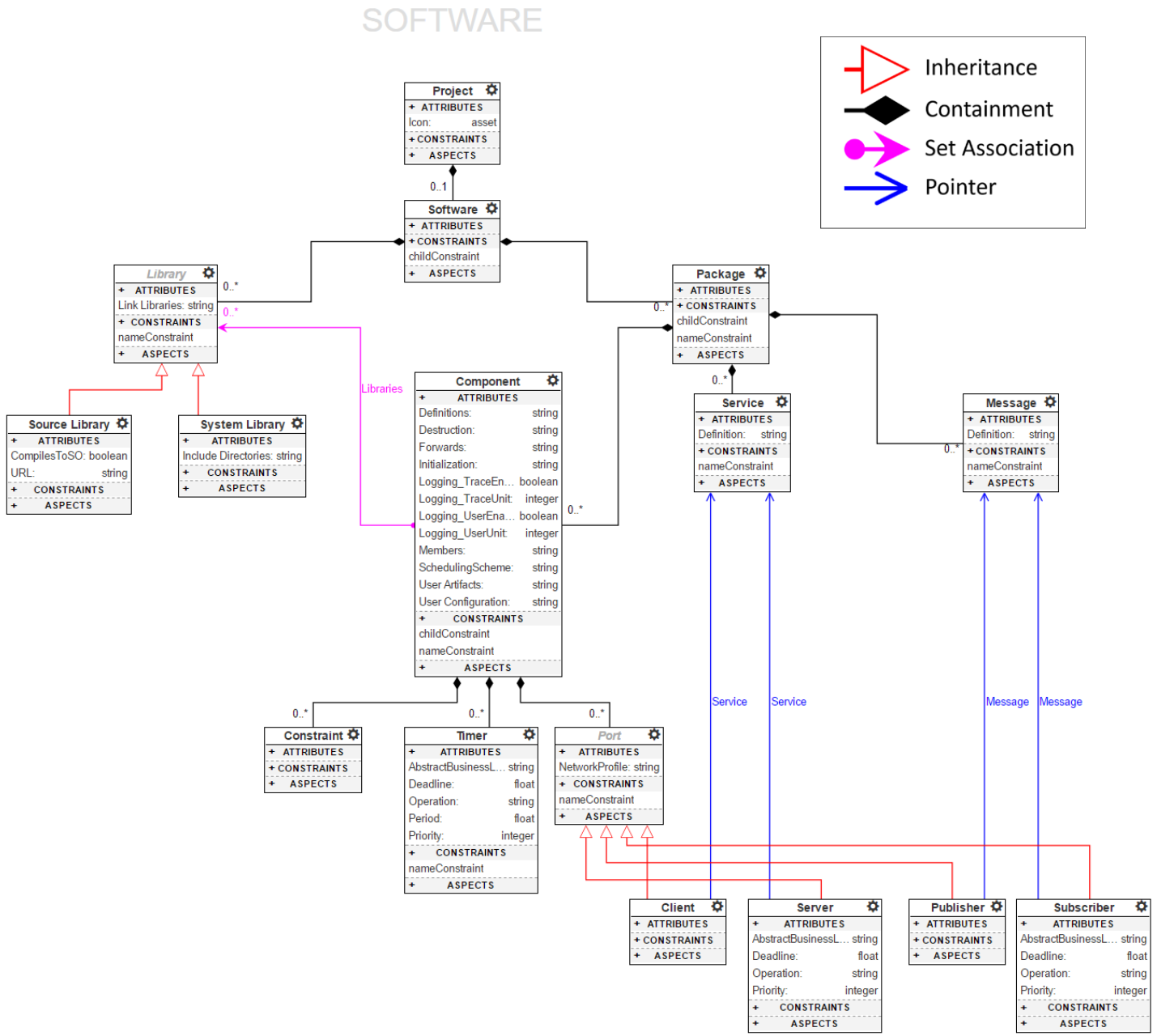
VADL Webpage: *vanderbilt.edu/usli*

Live ROSMOD Server: *rosmod.rcps.isis.vanderbilt.edu*

ROSMOD Demo

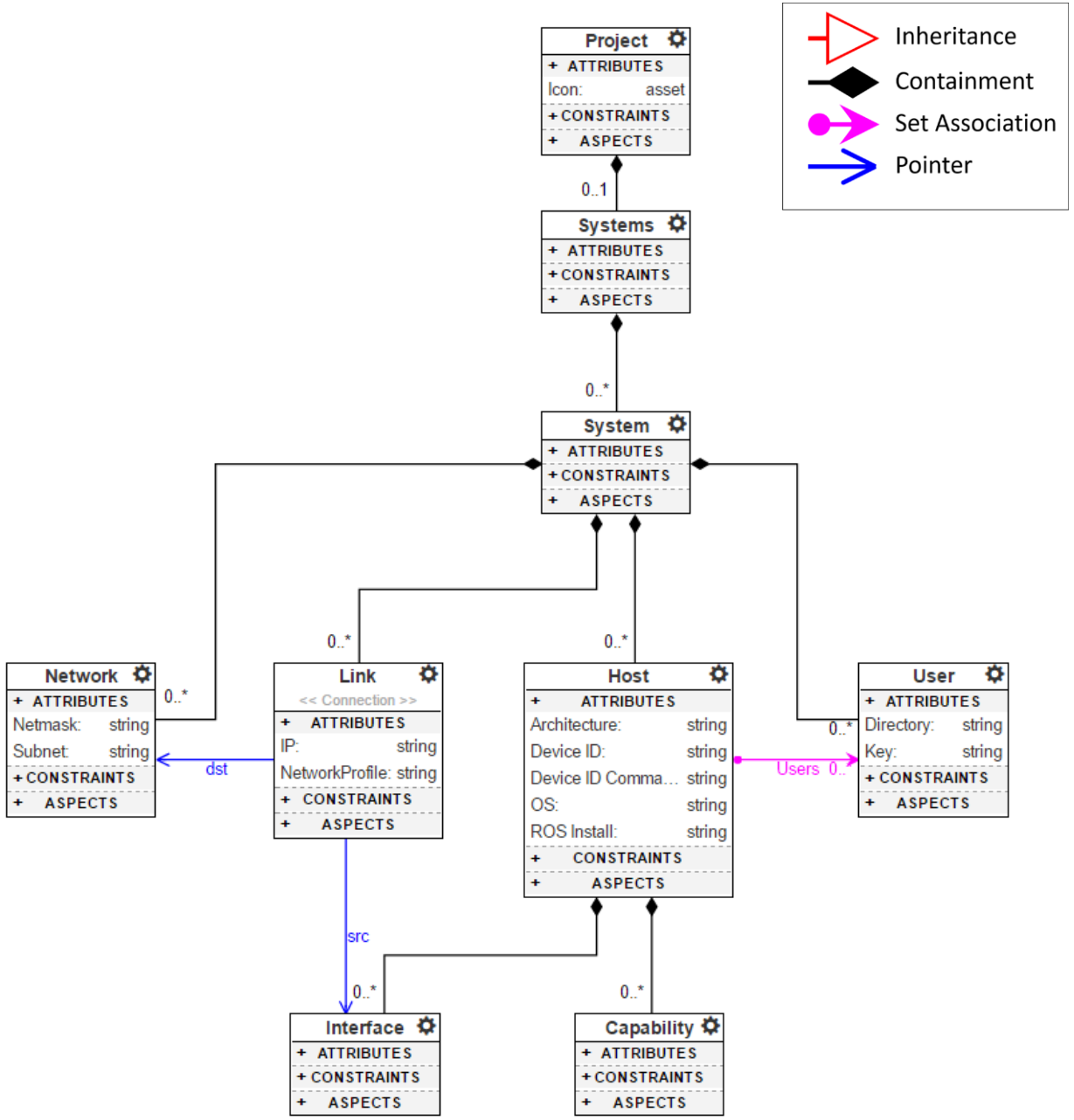
Software Meta-model

- Software contains generic concepts like libraries, operations, definitions, etc.
  - Mostly language agnostic; can be C/C++, python, or any other language which supports these concepts
- ROS specifics only small part of the meta
  - Some naming and structure (easily modifiable)
- Coupled with the software generation / compilation infrastructure



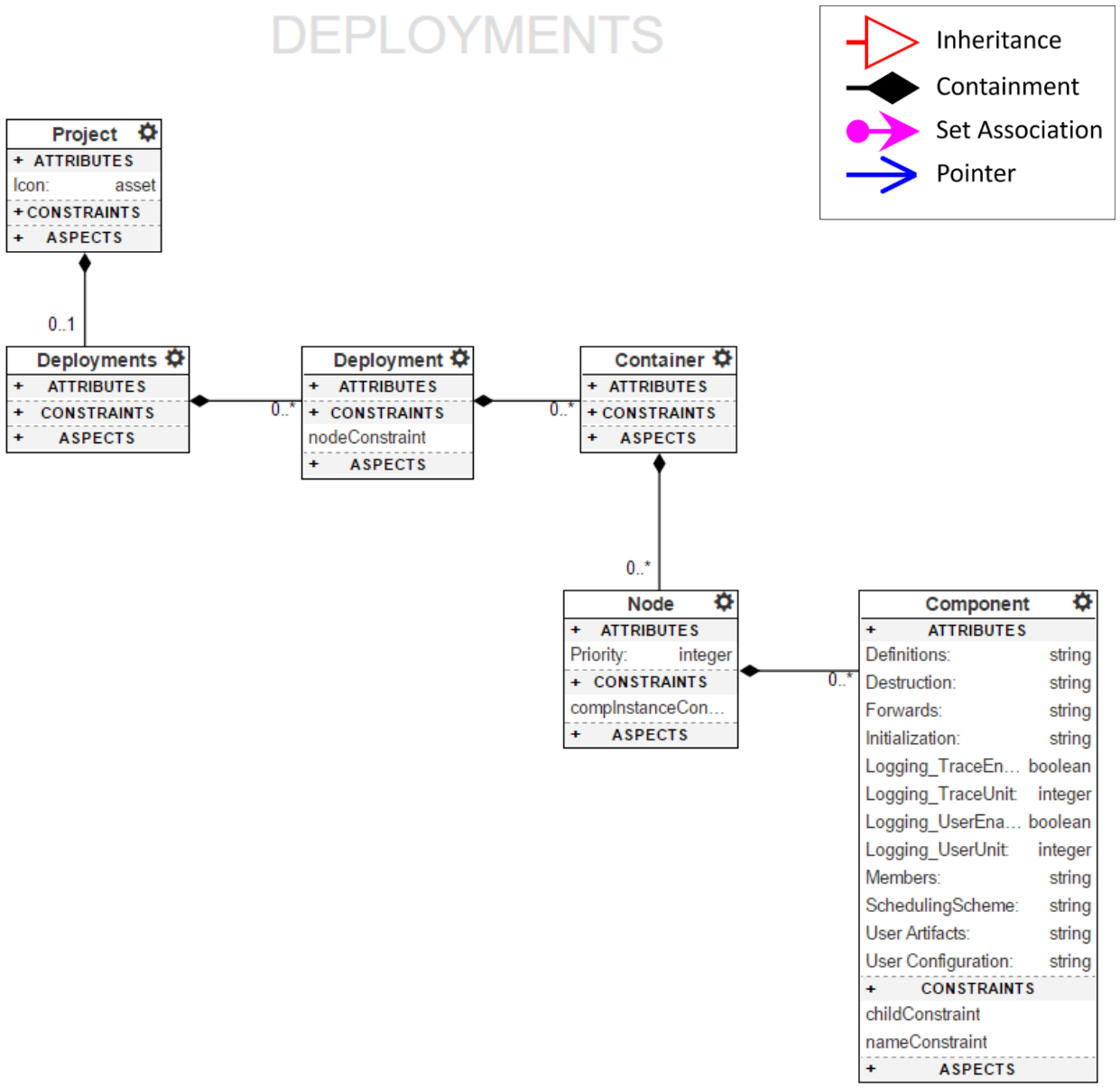
System Meta-model

- Models networked computing nodes
- Implicit / Indirect dependencies
  - Bash / SSH
  - *NIX
  - Network connectivity between *ROSMOD Server* and each *Host*
- Implementation specifics:
  - ROS install
  - Network / IP
  - User authentication
- Coupled to compilation / deployment infrastructure right now



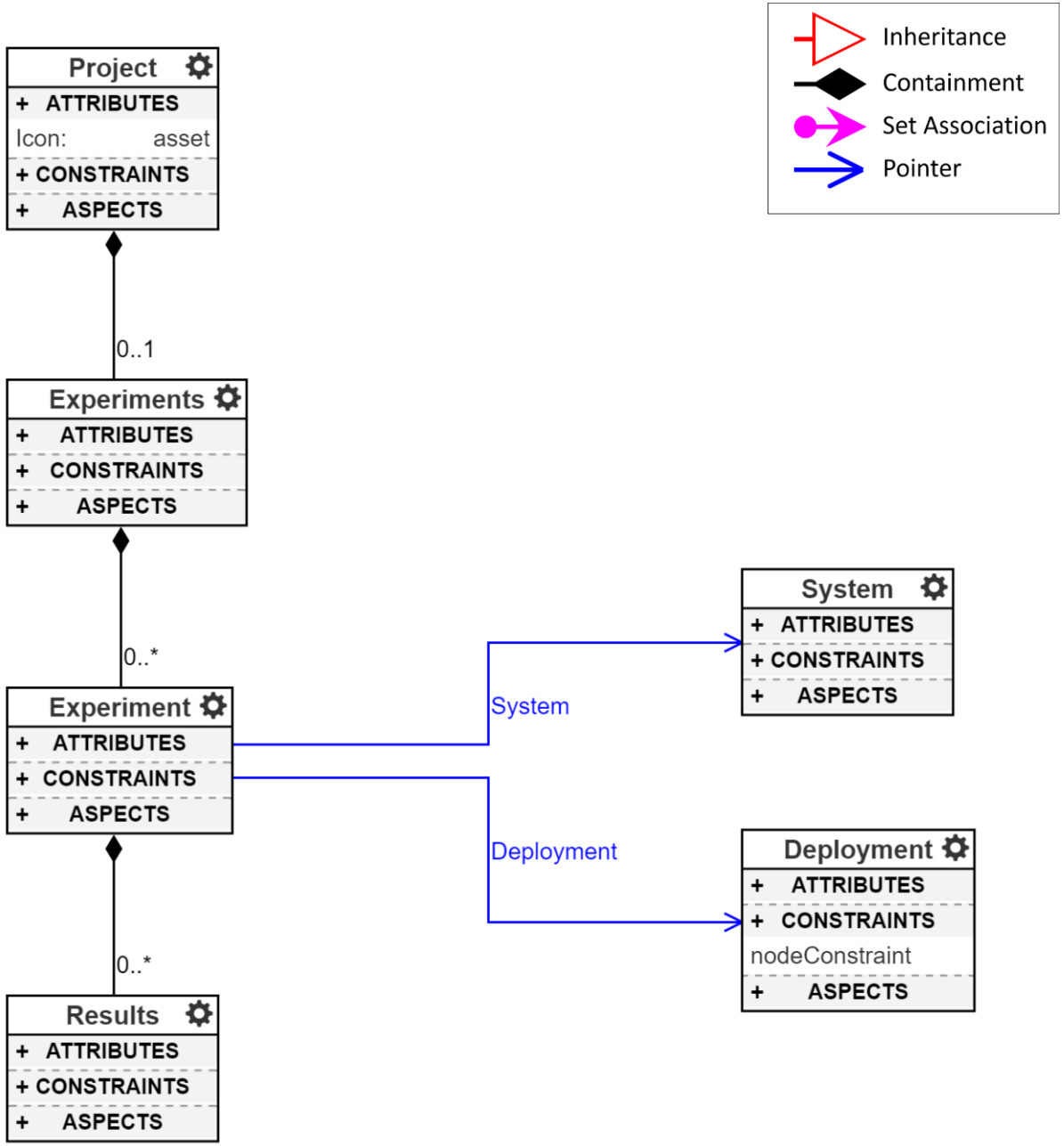
Deployment Meta-model

- Group *Component Instances* into *Nodes* (ROS term for processes)
  - Remember, each component has a single executor thread
- Group processes into *Containers*, the abstract representation of *Hosts*
  - Abstract so that they're not directly tied to specific architectures / IPs
- So how do you link the *Containers* to real *Hosts*?
  - Let ROSMOD do that for you when you run an *Experiment*



Experiment Meta-model

- An *Experiment* allows the user to say onto which S*ystem* they want to run a specific *Deployment*
- ROSMOD infrastructure automatically:
  - Ensures all user-specified component ***constraints*** are satisfied by the system's ***capabilities***
  - Checks the system's **availability**
    - Network connectivity
    - Credentials
    - CPU utilization



Documentation Meta-model

- Documentation is probably the **most important** part of the model!
- User-created **doxygen** documentation for code artifacts
  - In addition to automatically generated docs
  - Automatically compiled into HTML / PDF (optionally)
- User-created **markdown** documentation for every element of the model
  - Rendered in WebGME
  - Generated into MD / RST source files and HTML / PDF (optionally)