

The SAW Next Generation Workflow System



SANDIA ANALYSIS WORKBENCH
Sandia Enterprise Edition

Ernest J. Friedman-Hill
ejfried@sandia.gov



Motivation

Our Users

- Engineers and scientists
- Computer-literate
- Know scripting or programming
 - **Not “programmers”**
- Run lots of complex, in-house research codes
 - Experts with some, novices with others
- Need to follow best practices
 - Verification and validation

Motivation

Environment

- Heterogeneous hardware
- Multiple, secure networks
- Need-to-know
- *Decentralized*
- Compartmentalized

The Sandia Analysis Workbench

An integrated environment for engineering analysis

- Graphical model building
- Job submission
- Distributed file management
- Scientific data management
- Requirements management

The screenshot displays the Sandia Analysis Workbench interface. The main window is titled "SDM - JoeDemo2/Files/3_point_bend_test1.i - DART Workbench". The interface is divided into several panes:

- Project Navigator:** Shows a tree view of the project structure, including folders like "DAKOTA-Milestone", "Ed_March_Training", "JoeDemo2", and "Files".
- Code Editor:** Displays a job script for "3_point_bend_test1.i". The script includes comments about the SIMBA version and build number, and defines parameters for material properties and job conditions.
- Job Status:** A table showing the status of various jobs. The table has columns for Name, Machine, Stage, Queue Status, and Submit Date.
- Model View:** A 3D visualization of a mechanical part, showing a mesh and a color-coded stress distribution.
- Team Members:** A table listing team members and their roles.
- Plot:** A line graph showing a linear relationship between two variables, with the x-axis ranging from 0.0 to 1.3 and the y-axis from -0.0007 to 0.0000.
- File Explorer:** A file browser showing the contents of the "/scratch1/elhoffm/JoeDemo2/Files" directory, listing files like "3_point_bend_test1.i", "3_point_bend_test1a.cfg", "3_point_bend_test1a.g", etc.

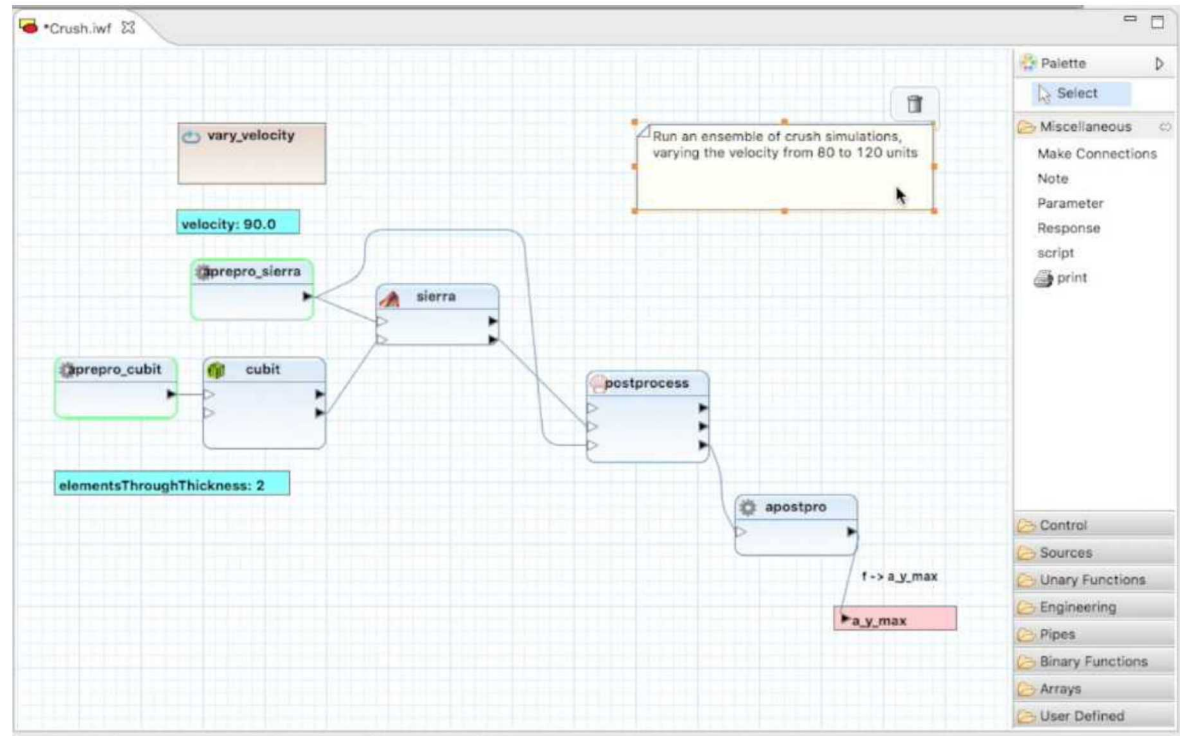
Name	Machine	Stage	Queue Status	Submit Date
Joint_model_SALINAS	shasta	Finished	Completed	Tue Nov 24 15:56:2
Joint_model_SALINAS	shasta	Finished	Completed	Tue Nov 24 16:01:1
Joint_model_SALINAS	shasta	Finished	Completed	Tue Nov 24 16:20:1
dtlb_blivet_060515	thunderbird	Finished	Completed	Tue Nov 24 16:37:1
Joint_model_SALINAS	thunderbird	Finished	Completed	Wed Nov 25 12:09:2
Tail_assy	thunderbird	Finished	Completed	Wed Nov 25 12:25:4

Role	Email	Name
Team Member	mjgibso@sandia.gov	Gibson, Marcus J
Project Manager	elhoffm@sandia.gov	Hoffman, Edward
Team Member	jagreen@sandia.gov	Greenfield, John

SAW Next Gen Workflow (NGW)

Automated workflow that

- is integrated with everything else in SAW
- works with with external codes
- runs inside or outside SAW
- is intuitive



NGW System Requirements

Low cognitive load

- Users can think in the domain, not about the workflow system
- You don't *program* workflow, you *build* it

Transparent and “escapable”

- It's always obvious what the system is doing
- It's always possible for user to customize behavior

Flexible system architecture

- Everything can execute anywhere

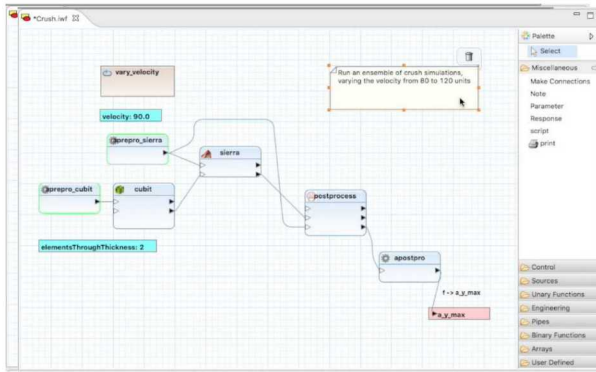
Zero-install

- *Can't assume root access on runtime system*
- *Can't install a persistent server*

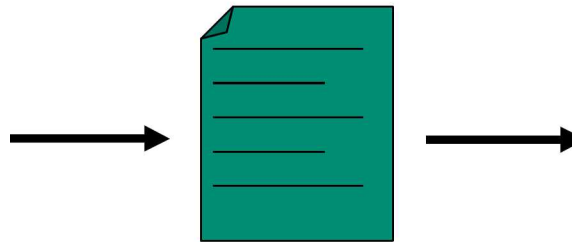
Workflow engine independence

- Can be implemented on multiple products

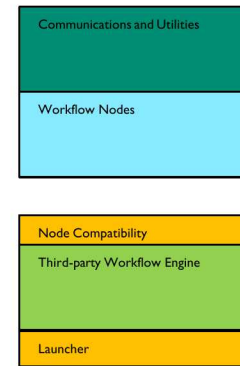
Next-Gen Workflow Architecture



1. Graphical Editor



2. Workflow File



3. Layered Runtime

Graphical editor and runtime are independent and can be used separately in time and space. Editor runs on desktop, while runtime can execute anywhere from desktop to HPC clusters

NGW Definitions

A **workflow** is an executable graph of connected components

A **component** or “**node**” represents an activity

- Executing an external code
- Extracting a column of numbers from a table
- Components are user-configurable
- Some very specific, some very generic

Connections represent data interchange and control flow

- A number or a path to a file
- An identifier for a shared memory segment
- *Connections are themselves configurable*

NGW Graphical Editor

Based on Eclipse IDE framework

- Graphiti / EMF
- Borrows some ideas from Triquetrum/ESWG

Data-driven

- XML file to describe available component types

Easy to extend

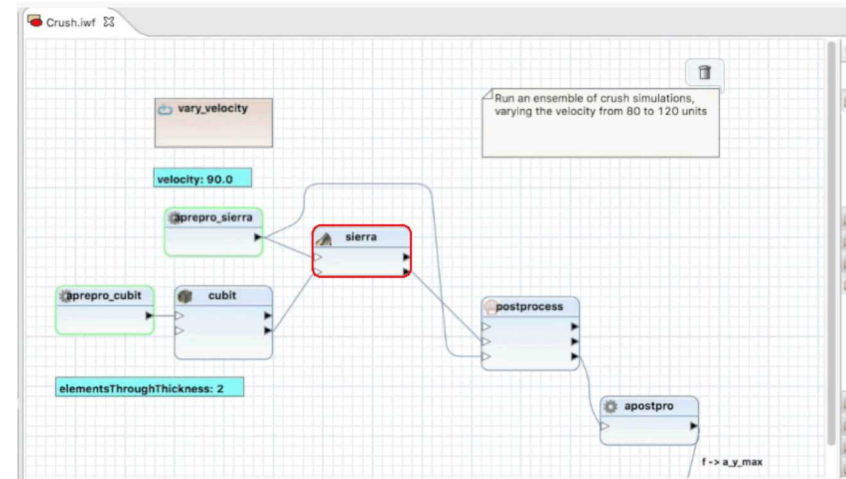
- Can add component types
- Can add component-specific editing
- Can add special behaviors for embedded execution
- Programmers work with editor's EMF model

NGW Graphical Editor

Eclipse integration

Ease of use

- Drag file from project, creates appropriate component to use that file
- File-type-specific editors automatically invoked



Integrates with other tools in SAW

NGW Graphical Editor

Edit model components naturally using familiar tools

The screenshot displays the NGW Graphical Editor interface, which is divided into several panels:

- Workflow Diagram (Top Left):** A visual representation of a simulation workflow. It includes components like 'vary_velocity' (with a value of 90.0), 'aprepro_Sierra', 'aprepro_Cubit' (with a value of 'elementsThroughThickness: 2'), 'cubit', 'sierra', 'bashScript', 'apostpro', and 'a_y_max'. Arrows indicate the flow of data between these components.
- Code Editor (Top Right):** A text editor showing the input file 'crush.template.inp'. The code defines functions for 'pmdi20_constant', 'pmdi20_modulus', and 'pmdi20_rate'. The 'pmdi20_modulus' function is currently selected and highlighted.
- Settings Panel (Bottom Left):** A panel for editing the 'definition for function' 'pmdi20_modulus'. It shows the function type as 'piecewise linear' and provides a table for the function definition.
- Plot View (Bottom Right):** A graph showing the 'pmdi20_modulus' function. The x-axis is 'time' (ranging from -53.9 to 81.66) and the y-axis is 'Temperature' (ranging from 0.71 to 1.28). The plot shows a piecewise linear function that decreases over time.

Function Definition Data:

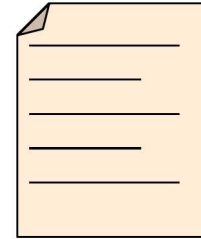
x value	y value
-53.90	1.23
21.10	1.0
73.90	0.80
82.20	0.75

Workflow File Format

Based on EMF native format (XML)

Simple file format contains

- ... component configuration
- ... connectivity
- ... optional graphical (layout) information



Does *not* contain component implementations

- Different runtime implementations can execute the same workflow definition

Does *not* contain rendering information

- Appearance of components easy to change

Can be generated or consumed by any code

- i.e., workflows can be defined by code or scripts
- Eclipse parser library not required
- No Eclipse code (EMF) in server

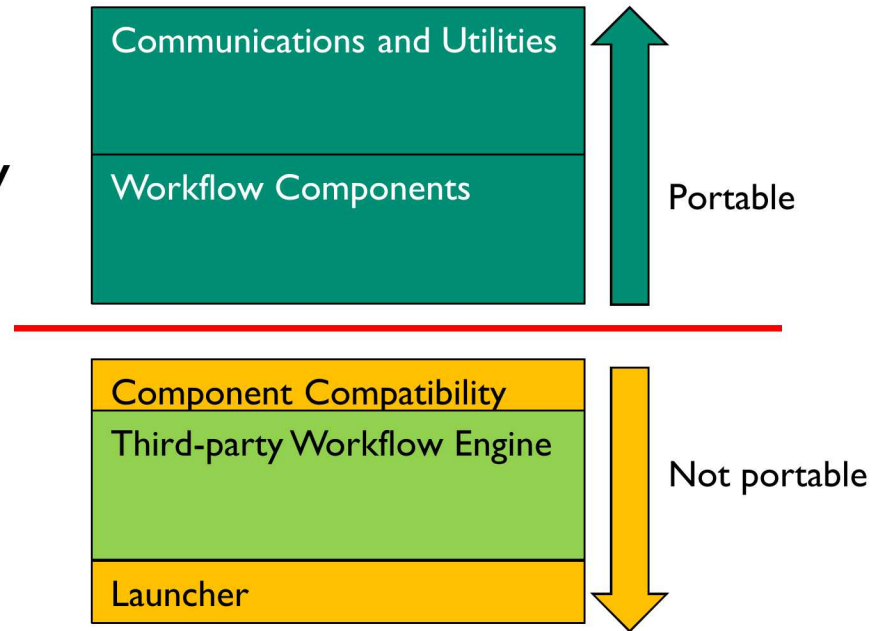
Workflow File Format

```
<gov.sandia.dart.workflow.domain:WFNode name="print" properties="/4 /5 /6 /7"
  inputPorts="/2" outputPorts="/3" start="true" type="print" label="print"/>
<gov.sandia.dart.workflow.domain:InputPort name="x" type="default" node="/1"/>
<gov.sandia.dart.workflow.domain:OutputPort name="f" type="default" node="/1"/>
<gov.sandia.dart.workflow.domain:Property name="formatString" type="default" value="Hello, world!" node="/1"/>
<gov.sandia.dart.workflow.domain:Property name="async" type="boolean" value="false" node="/1"/>
<gov.sandia.dart.workflow.domain:Property name="privateWorkDir" type="boolean" value="false" node="/1"/>
<gov.sandia.dart.workflow.domain:Property name="clear private work directory"
  type="boolean" value="false" node="/1"/>
```

NGW Engine

Layered Runtime Architecture

- NGW Components are written in Java to a vendor-neutral API. The compatibility layer adapts to a third-party rule engine
- Variant component implementations can be provided for different host environments

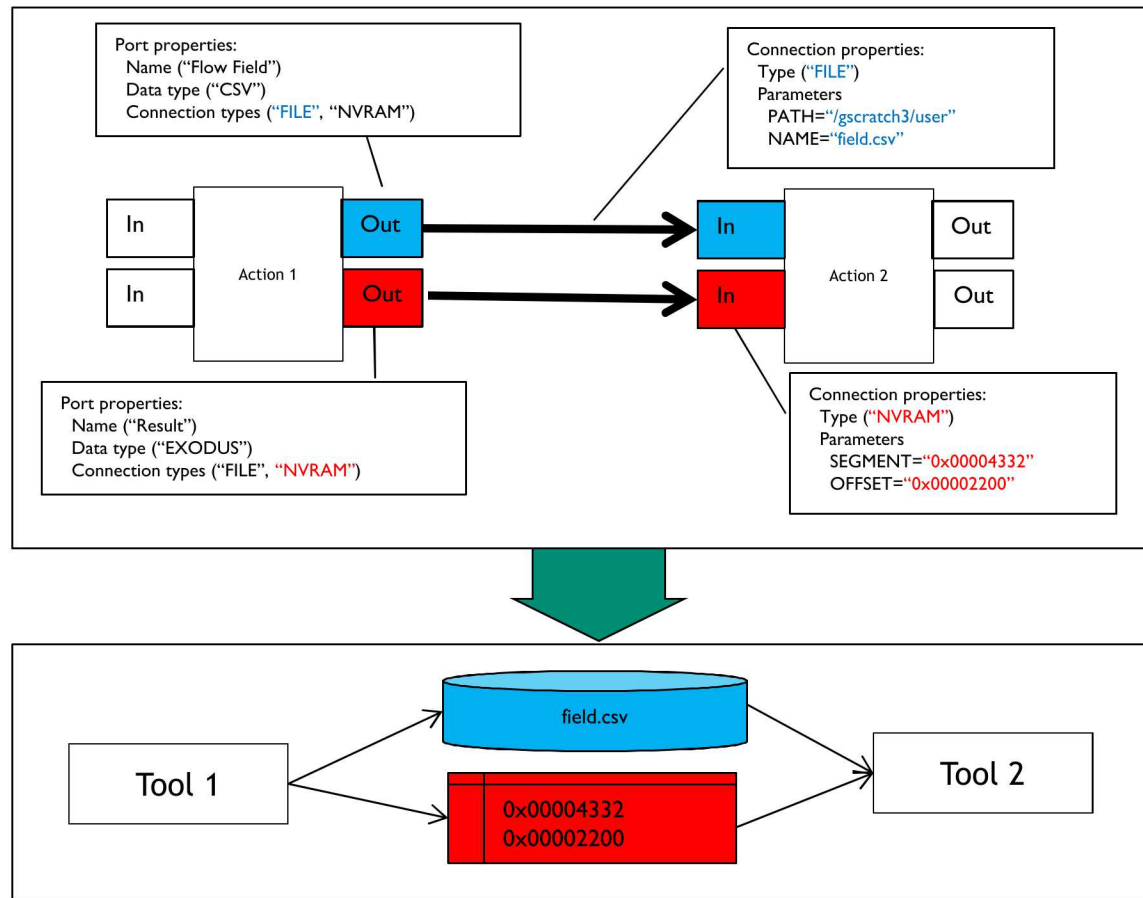


By using a thin compatibility layer, we can deploy on arbitrary third-party Java workflow engines with little effort

NGW Engine

Abstract Dataflow

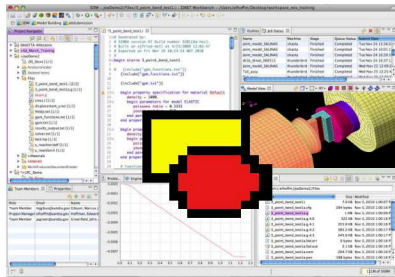
- Connections between components are configurable. Configuration options determine how data is exchanged at runtime
- Future work will involve adding support for more data exchange mechanisms
- In this notional diagram, the blue connection is configured to exchange data via a file, while the red is using a shared memory segment



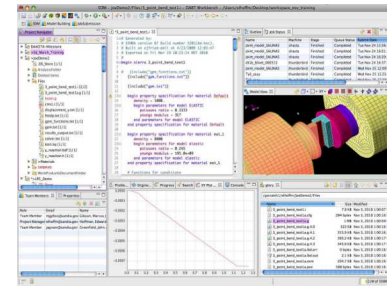
NGW Engine

Flexible Topology

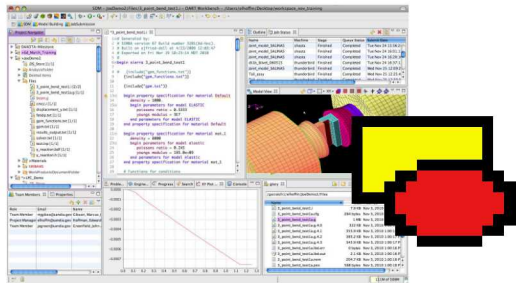
The same workflow can run in multiple environments. We can leverage our existing job submission and monitoring tools to implement this



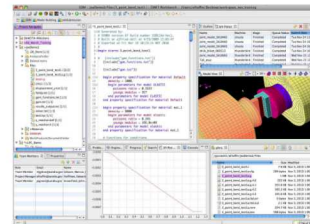
Interactive



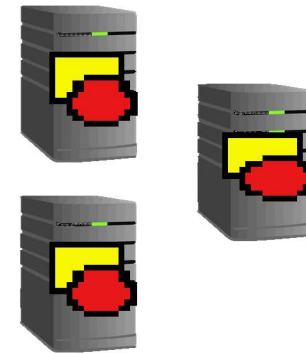
Server



Batch



Distributed



Extending NGW

”Wrapping” an external code for runtime use

- *No programming required*
- Can be done in the workflow editor
- Configure a script node or external process node
- Save to palette for personal use
- Export to JAR file for sharing

Creating a more complex runtime node

- Requires Java programming
- Extend SAWCustomNode
- Implement (at minimum) one method
- Package as JAR file, in Eclipse plugin, or both

Extending NGW

Defining node types for editor

- Requires Java programming
- Implement one method in `IWorkflowEditorNodeTypeContributor`

Custom rendering for nodes in editor

- Requires Java programming
- Extend `AbstractGARenderer` and implement two methods
- Register with `nodeRenderer` extension point
- Package as Eclipse plugin

Extending NGW

Custom editors for specific node types

- Requires Java programming
- Implement `ISettingsEditor` and implement several methods
- Register with `nodeTypeEditor` extension point
- Package as Eclipse plugin

Custom images for nodes in editor

- Register icon with `nodeIcon` extension point
- Package as Eclipse plugin

Providing files and resources to workflow creator

- Register file with `resourceContributor` extension point
- Package as Eclipse plugin

Impact

- Best Practices and New Capabilities
 - Workflows capture the best way to complete a task
- Reproducibility
 - It's easy to make sure you're performing the same steps
- Reliability
 - Errors handled robustly
- Reusability
 - No reinventing the wheel
- Team communication
 - Intrinsically documented processes
- Archival Information
 - Automatically preserve what was done and why

Future work

Façades

Web client

Alternate servers

- Python
- C++
- Other Java?

In situ workflow