

Interoperable Application Programming Interfaces for Computer Aided Engineering Applications

William T. Jones
Stephen L. Wood
Kevin E. Jacobson
W. Kyle Anderson

NASA Langley Research Center

Copyright 2021 United States Government as represented by the Administrator of the National Aeronautics and Space Administration. No copyright is claimed in the United States under Title 17, U.S. Code. All Other Rights Reserved. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

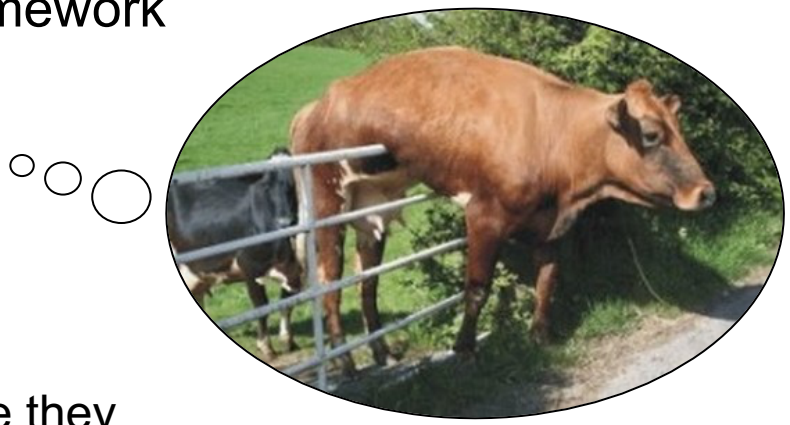


Motivation

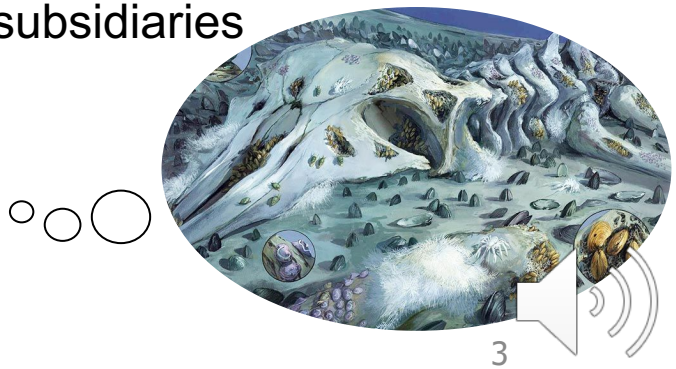
- Computer Aided Engineering integrates components from a number of different disciplines
  - FSI: loads from CFD, displacements from CSM, mesh deformation, loads transfer, etc.
- Traditionally components hardwired together for specific implementations of each discipline
- Multi-discipline, multi-physics coupling capability driven by frameworks
  - Tie together multiple components with defined interaction
- Desire an API to control interaction
  - Interaction via defined abstraction of each component
    - Same regardless of underlying implementation
- Facilitate code reuse and promote collaboration among component developers

Frameworks Woes

- This work focuses on development of a generic API not a framework
- Often frameworks impose on the component developer such that they are required to have knowledge "on both sides of the fence"
  - Programming language, data structures, and workflow dictated by the framework
  - Component developers need knowledge of the framework where they will be used and/or the framework developers need component implementation details

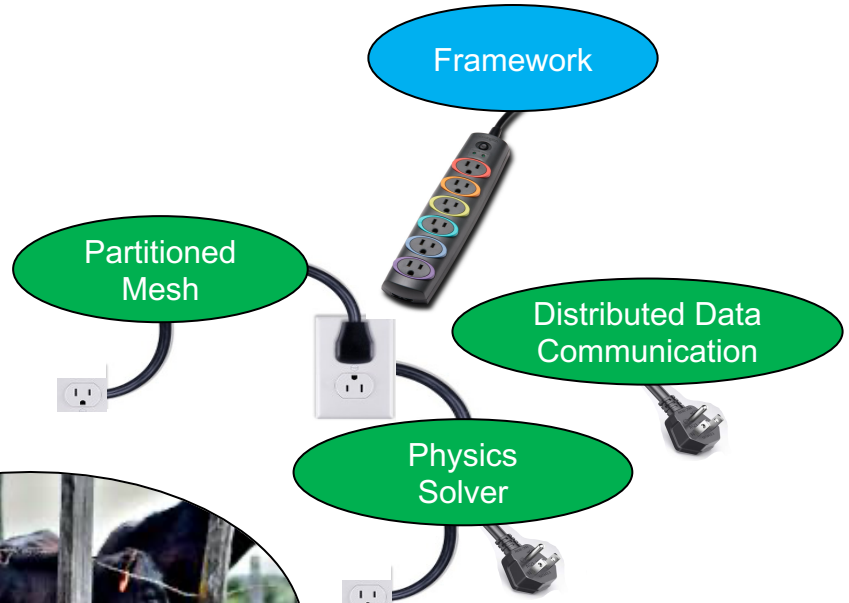


- Difficult to seamlessly utilize a new/external component implementation
  - Rigid set of *usable* components
  - Limited code reuse/collaboration
  - Framework fully successful only *within* the developing organization or its subsidiaries
- Often leads to limited use of the framework and/or the component
  - As a result, most frameworks have become "Whalefall" – J. Alonso, 2018
    - Ocean floor is littered with the carcasses of behemoths



Interface Design

- Treat components as appliances (with standardized plugs, receptacles, voltage, etc.)
- *Abstract* software interfaces for CAE component interoperability
- With standardized interfaces, a single component can be used by many frameworks and other components without modification
  - No dependency of component on framework and visa-versa
    - Component developer focuses on component
    - Framework workflow independent of component implementation details
  - Language independent
    - ”Flat” software interface using only interoperable data types
    - Interface *exposed* as C since it is callable from all major languages
      - Interface implemented in developer’s language of choice
    - Component *state* communicated via opaque *Pointer* argument





Interface Example in C

```
extern "C" int64_t tinf_mesh_node_count(void* mesh, int32_t* err)
```

Description

Retrieve the total number of nodes in the mesh partition (resident and neighboring/ghost nodes).

Parameters

mesh Opaque mesh pointer

error Error code

Return

Total number of nodes in the partition

Interface Example in Fortran

```
interface
 function tinf_mesh_node_count(mesh, error) result(cnt) bind(c)
  use, intrinsic :: iso_c_binding, only : c_ptr, c_int64_t, c_int32_t
  implicit none
  type(c_ptr),    value,     intent(in)    :: mesh
  integer(c_int32_t),        intent(out)   :: error
  integer(c_int64_t)                       :: cnt
 end function tinf_mesh_node_count
end interface
```

Description

Retrieve the total number of nodes in the mesh partition (resident and neighboring/ghost nodes).

Parameters

mesh Opaque mesh pointer

error Error code

Return

Total number of nodes in the partition

Interface Definition

- Goal is to develop interfaces for CAE
- Initial thrust focused on CFD
- As of publication, interfaces categorized as follows:
  - Parallel communication
  - Problem definition
  - Mesh access
  - Linear solver execution
  - Nonlinear iterative solver execution
  - Solution access
- Full definitions included as an Appendix in paper
  - Follow Open-Closed design principle
- Multiple implementations developed

Creation of State

```
int32_t tinf_mesh_file_create(void** mesh, const char* name,
                              size_t nlen, const char* style,
                              size_t slen, const char* form,
                              size_t flen, void* problem, void* comm)
```

Description

Create a mesh object from a file (implements mesh interface)

Parameters

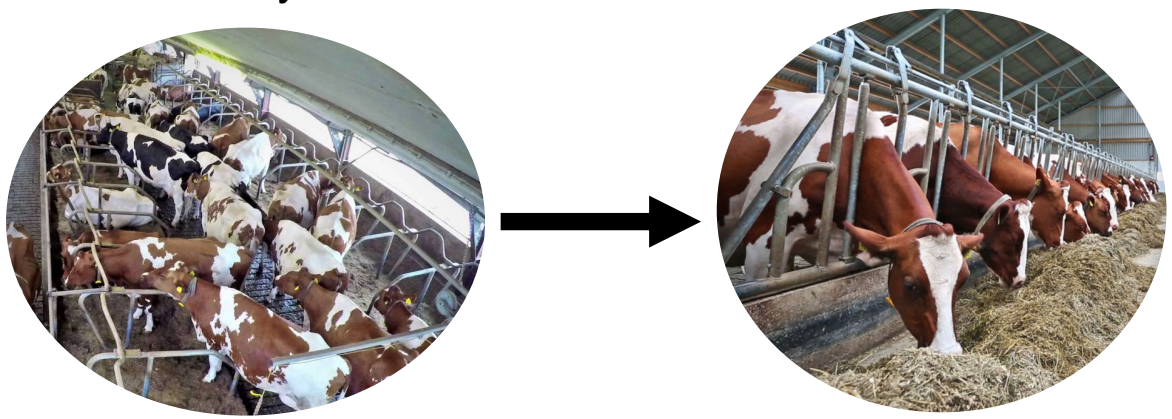
mesh	Opaque mesh pointer
name	The name of the file containing the mesh
nlen	Length of name
style	The style of the data layout in the file
slen	Length of style
form	The data format (i.e., 'binary')
flen	Length of form
problem	A problem description object
comm	A communications object

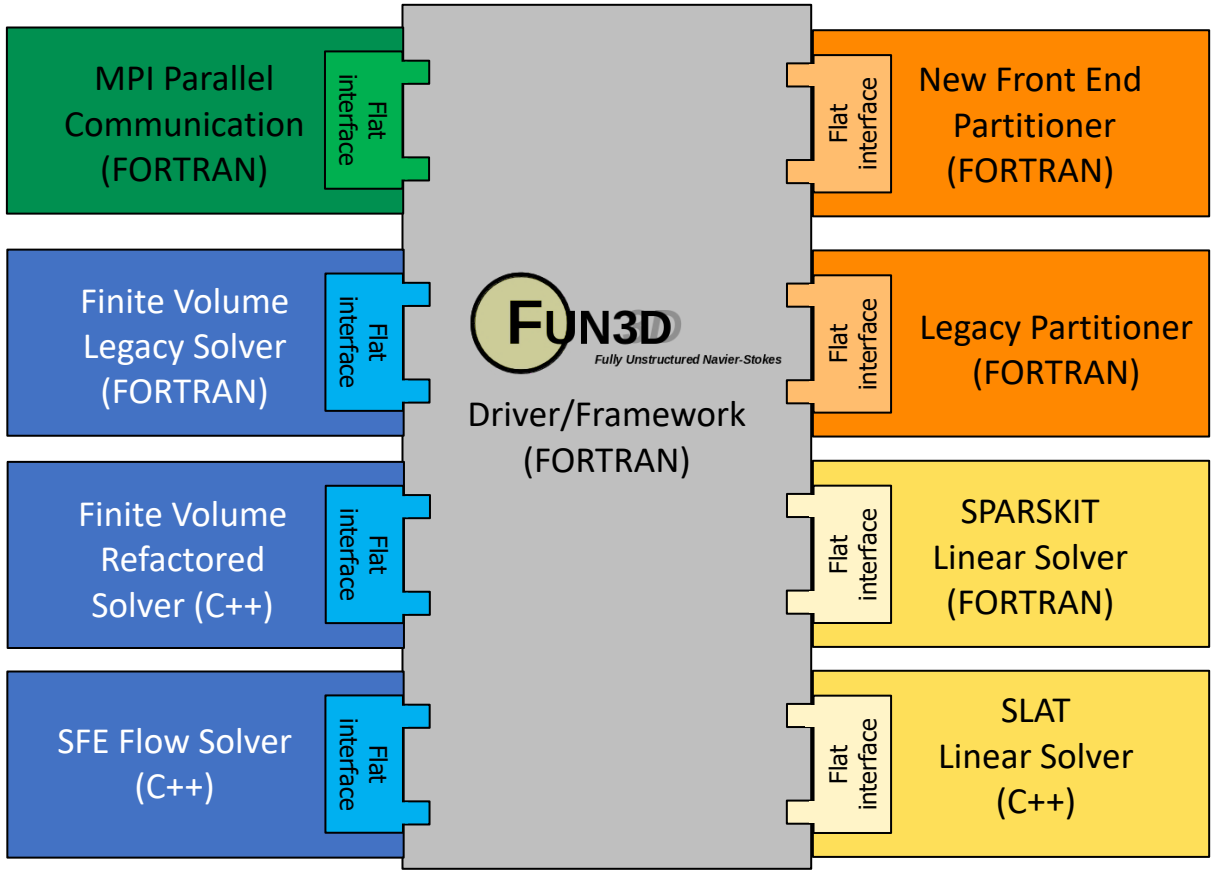
Return

Error code

Application in FUN3D

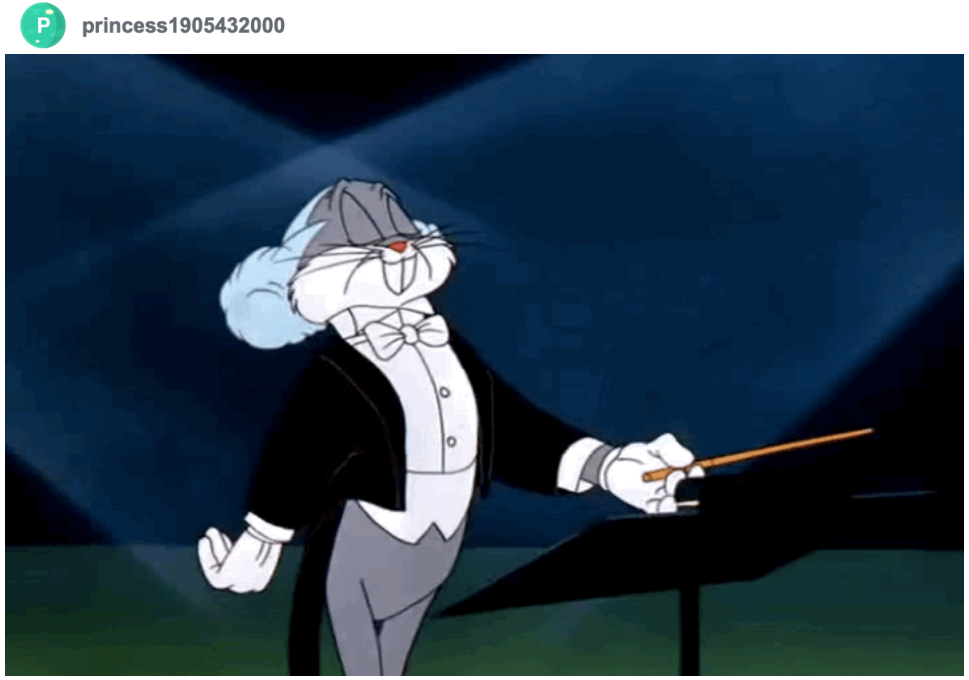
- Major parts of FUN3D have been extracted and put behind the framework-independent flat interfaces
- Significance:
  - Reduced entanglement of the code resulting in improved code maintenance
  - Extracted FUN3D components can be utilized in other frameworks and by external stakeholders
  - Changes are opaque to the FUN3D user
  - SFE and NCFV solvers
  - Legacy partitioning and NFE
  - SPARSKIT and SLAT linear solvers
  - Statically linked





Dynamic Plug-in Components

- Development of new *Maestro* framework
- Components are DSOs
  - Runtime augmentation of capability
- Uses defined interfaces to *orchestrate* interaction of components
  - Unlike traditional frameworks, Maestro does not dictate the means of the interaction
  - Any compliant component, from any source can be simply plugged in
  - No source code modification required to add/change capability
- No special coding of components required
  - DSO created during linking
- Runtime configuration of mesh partitioner and flow solver
  - Produces results consistent with statically linked FUN3D



Buggs Bunny Maestro

Conclusion

- Abstract, standardized interfaces initiated for CAE
  - Initial focus is CFD
- CAE components are tied to the standard interfaces not to a particular framework
- Likewise, the framework is tied to the interfaces and not a specific CAE component
- Components can be seamlessly replaced in framework to change behavior
- Algorithmic details, programming language, and component state are hidden behind the interfaces
  - Programming language independent
- Facilitates plug-and-play architecture

CAE components are treated as *appliances* that can be selected and exchanged as needed within a framework

✓ **You don't rewire the house because you bought a new toaster**

Acknowledgments

- Members of the FUN3D Development Team
  - In particular, Dr. Li Wang for her help in the development of NCFV
- This research is sponsored by the Revolutionary Computational Aerosciences (RCA) subproject of the NASA Transformational Tools and Technologies (TTT) Project of the Transformative Aeronautics Concepts Program (TACP) under the Aeronautics Research Mission Directorate.