TIFAS: Topology Indepenent Framework for Aeromechanics Simulation

A novel approach to CFD and Multibody Dynamics software integration

Benjamin Silbaugh[1]

[1]Graduate Research Assistant
silbaugh@umd.edu
Department of Aerospace Engineering
University of Maryland, College Park

October 12, 2012

Introduction

What is "topology independent" modeling?

Why bother with topology independent modeling?

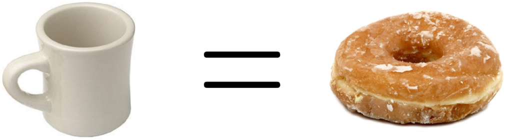
What is this silly TIFAS thing anyhow?

Outline

Outline

What is Topology?

Topology provides a foundation for understanding connectedness

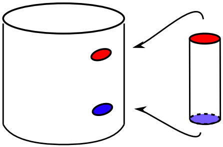


"A topologist is a person who cannot tell the difference between a coffee mug and a donut."

- We're interested in composing arbitrary systems from primitive components
- In this context, "topology" refers to how those components connect

How do we define a topology?

The simplest approach is to define a topology in terms of more primitive topologies.

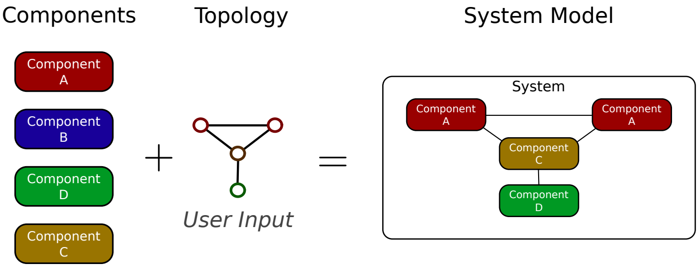


- Identify a collection of primitive objects with well defined topologies
- Identify faces, edges, points that join

Think quotient spaces.

Topology independent modeling?

Tools and algorithms that do not make any a-priori assumptions about the system topology



Outline

Not All Rotorcraft Share the Same Topology

- Multitude of vehicle configurations: conventional helicopter, tilt rotor, tandem, etc
- Rotor hub details
- Aerodynamic/vibration enhancements: e.g. flaps, slats, etc
- Novel control schemes: flaps, IBC, etc

Hard to find a common set of parameters to describe everything

Software Development and Reliability

Topology independent models reduce model development effort

- Facilitates distributed development and maintenance
- Isolated code units are easier to test and debug
- Possible to run basic fluid, structural, and aeroelastic benchmark test cases
- Reuse of tested component models improves system model reliability

Outline

TIFAS Project Description

Goals

- Develop a software framework for topology independent modeling of aeromechanics (CFD/CSD)

Constraints

- Allows solvers to be written in Fortran, C, C++, and use MPI
- Runs on both shared and distributed memory systems
- Compatible with POSIX compliant systems (UNIX/Linux)

Quality metrics

- Code is clearly written and well organized
- Minimizes effort required to add new capabilities (extensible)
- Minimizes effort and skill required to setup simulation
- Minimizes memory and CPU load

TIFAS Architecture Overview

A TIFAS simulation is roughly structured into 3 layers:

- Execution environment
  - Underlying operating system (external)
  - Parallel runtime environment (external, optional)
  - Execution Monitor
- Simulation management
  - SIMMAN: SIMulation MANager
  - Solver Modules
  - Requisition Agents
  - Solver (external)
- Inter-Solver Connectivity
  - Network
  - Surrogates
  - Observers
  - Maps

Fluid and structural solvers are essentially “plug-ins”.

Solver

Definition (Solver)

A solver is an algorithm combined with a data structured that has been designed to model the behavior of a physical body or medium.

Examples:

- A CFD solver such as OVERTURNS, OVERFLOW, SU^2
- A multi-body dynamics solver such as Rodymol, MBDyn, DYMORE
- A vortex lattice model

Comments:

- A solver is external component of TIFAS (simplifies devel and licensing)
- Solvers are compiled into dynamically loaded libraries (simplifies licensing)
- TIFAS defines an API that each solver must implement
- The API is such that TIFAS can control module initialization, deallocation, and time step synchronization.
- Upon initialization, solvers are given a handle to a Requisition Agent, through which each solver obtains access to TIFAS resources

Solver C API

```c
/*******************************************
 * All functions return an integer return code. *
 * Non-zero value indicates an error.           *
 *******************************************/

// Initializes solver module.
int tifas_module_init(RequistionAgent* agent /* in */);

// Advances the solver by one time step
int tifas_module_step();

// Deallocates heap allocated memory, if any.
int tifas_module_del();
```

Choose C as Solver API language because:

- Most solvers appear to be written in C, Fortran, or C++
- Fortran (2003) and C++ provide intrinsic mechanisms for C interoperability
- C has well defined ABI

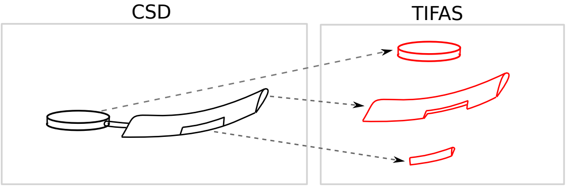
Surrogates

Definition

A surrogate is an object modeled after a primitive type of structural component or load, that responds to requests for information on behalf of a solver.

- Surrogates are a generic extension of a Solver's interface
- Solvers do not share surrogates
- Each solver module is responsible for updating its surrogates when change in state occurs.

Example:



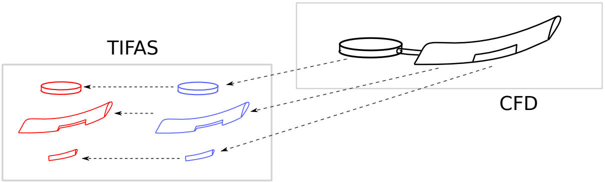
Observers

Definition

An observer is an object providing access to external information required by a solver.

- Each observer is assigned to a specific data source.
- Data sources include surrogates and maps
- Observers can function as a “window” by which a solver can view another solver’s data
- A single data source may be viewed by multiple observers

Example:



Maps

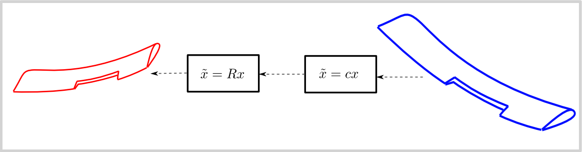
Definition

A Map is an object that applies a static transformation to the output of a data source.

- Is associated with a unique surrogate and observer pair
- Implements surrogate and observer interfaces
- May be composed (chained) to form a composite transformation

Example:

TIFAS

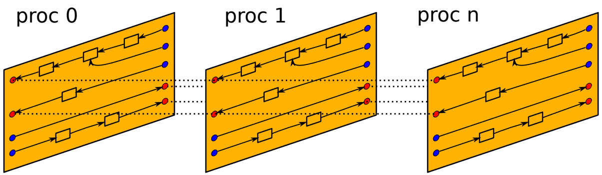


Network

Definition

A Network is a collection of surrogates, maps, and observers linked into the form of directed graphs with a surrogate as its origin.

- Constructs itself from a user description of system topology
- Is mirrored across all processes
- Mirrored surrogates share memory



Requisition Agent

Definition

The object by which a solver requests access to TIFAS resources during initialization is called a Requistion Agent.

- A unique Requisition Agent is created for each solver module
- Each agent may monitor and restrict access to resources on a per solver basis
- Facilitates detection of user configuration errors
- May be used in future to enable automated network configuration

Example:

```
int tifas_module_init(RequistionAgent* agent){
                    ...
    // Get handle to a blade surrogate...
    Object* blade_surrogate = agent.request_surrogate("BLADE_1");
    // Get handle to an airloads observer...
    Object* blade_airloads = agent.request_observer("AIRLOADS_1");
                    ...
    return 0;
}
```

Solver Module

Definition

A Solver Module is an abstract model of a Solver.

- An internal (C++) representation of a solver (external component)
- Contains mechanisms to find and load a solver from disk
- Maps error codes returned from solver functions (C) into TIFAS exceptions (C++)

Modeling solvers as modules instead of classes simplifies the reuse of existing CFD and multi-body dynamics software–which may contain statically allocated data (i.e. global variables).

SIMMAN: SIMulation MANager

Definition

The SIMulation MANager (SIMMAN) is the object that coordinates and monitors all elements of a simulation.

- Creates a network on each processor
- Instantiates all Solvers (passing each a Requisition Agent)
- Executes solvers in sequential fashion (staggered coupling)

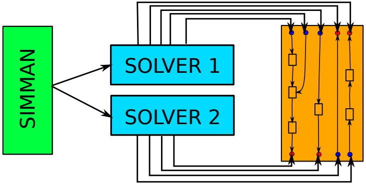


Figure : Only 2 solvers shown for clarity.

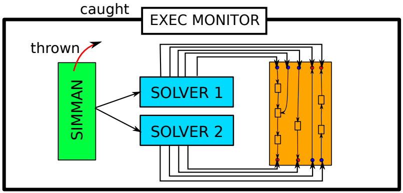
(Recall the game Simon Says)

Execution Monitor

Definition

The Execution Monitor is a driver program that loads inputs, instantiates SIMMAN, and catches runtime errors.

- A small program that “boots” SIMMAN, then waits for an exception
- Could be easily replaced by a different driver program



TIFAS DSL (user input)

A declarative, Domain Specific Language (DSL), used as user input.

Declaration of surrogates, maps, observers

```
<object name>
{
    type_name = <type name>;
    <property name> = <property value> ;
    [ <property name> = <property value>;]
    [ ... ]
}
```

Network topology syntax:

```
<surrogate name> -> [ <map name> -> [...]]] <observer name> ;
```

DSL could be easily mapped to a graphical language/interface (e.g. LabView, Simulink).

TIFAS Scripting Example

```
# Declare surrogates, maps, observers...
csd:beam{ type_name=FlexibleBeam; host_rank=0;  }
cfd:blade_mesh{ type_name=FlexibleBeamObserver; }
unit_conv{
    type_name=FlexibleBeamUnitConversion;
    config_file=unit_conv.in;
}
frame_conv{
    type_name=FlexibleBeamCoorRotation;
    config_file=frame_config.in;
}

# Define network topology...
csd:beam -> unit_conv -> frame_conv -> cfd:blade_mesh;
                         frame_conv -> freewake:blade_motion;
```

Outline

Language Choice

TIFAS is implemented in C++

- Desire OOP language: modular, extensible, verifiable, maintainable (DRY-principle)
- Mature compilers available for virtually all platforms
- ISO regulated - maintains backwards compatibility
- Can directly use libraries supporting C ABI

Connectivity Library provides C++, C and Fortran API's

Connectivity Library

Contract Programming

- A design scheme in which formal interface definitions are central to a program's structure
- The interfaces need not model a well defined object type
- Preconditions, post-conditions, and invariants may be used to ensure that a particular object meets spec
- Is found to work well for frameworks modeling servant-client relationships

In TIFAS:

- Combine C++ pure abstract classes with RTTI to emulate Contract Programming
- All surrogates, maps, and observers derive from a generic "object" type, and implement a specific set of predefined interfaces.
- Use of contracts simplifies implementation of Network as well as the test suite

Parallelization of the TIFAS Connectivity Library

MPI-2 Remote Memory Access (RMA)

- An alternative to traditional message passing
- Uses separate mechanisms for data transfer and synchronization
- Communication is by way of "memory windows"
- Synchronization schemes: active target (e.g. fences), passive target (locks)

- An identical Network is created on all MPI processes
- Each surrogate creates a memory window that is shared with its clones on other processes
- Surrogates share information with their respective clones by reading/writing to their memory window

Outline

Summary

Topology

- Connectedness: a geometric property more fundamental than shape or size

Topology Independent Modeling

- Regards topology definition as a user input
- Mitigates the need to find a common parameter space

TIFAS

- A framework for topology independent CFD/CSD analysis
- Solvers communicate by way of surrogates and observers
- Time synchronization is managed separately by SIMMAN
- System topology is described using a high-level, declarative, DSL
- TIFAS is implemented in C++

Future Work

Items on the TODO list:

- Look into releasing TIFAS under open source license
- Implement virtual work preserving load transfer algorithms
- Profile framework to better understand “bottlenecks”

Possible directions for future research:

- Revisit the choice of “pull” versus “push” model
- Investigate alternative concurrency models
- Hybrid CPU/GPU extensions
- Algorithms for automatic configuration of unit and frame conversions
- Alternative DSL’s for topology independent modeling:
  - Facilitates formal correctness proofs
  - More robust grammar (i.e. stronger syntax checking)

Questions?