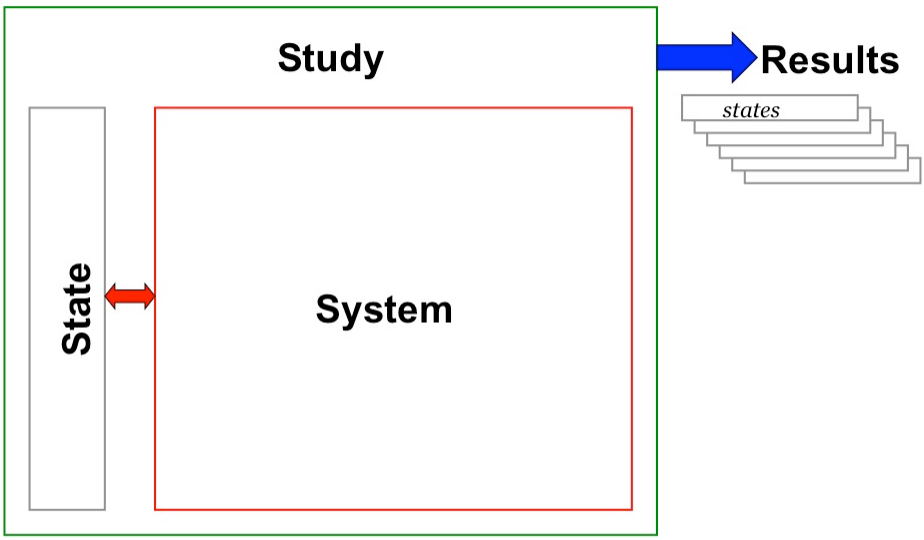
SimTK Simulation Concepts

The figure below shows the primary objects involved in computational simulation of a physical system in SimTK: *System*, *State*, and *Study*. OpenSim creates and manages specific objects of these types that are suitable for the domain of neuromuscular biomechanics. In particular, the OpenSim Model class implements a SimTK System, and the OpenSim Manager represents a Study. OpenSim uses a SimTK::State object directly to represent the state of an OpenSim Model.



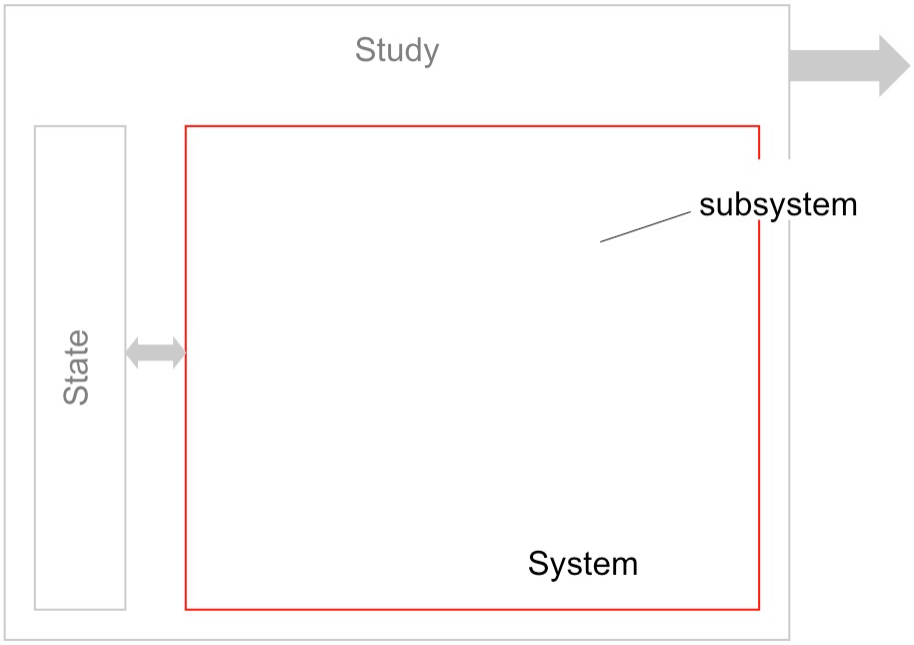
A *System* is the computational embodiment of a mathematical model of the physical world. A System typically comprises several interacting, separately meaningful subsystems. A System contains models for physical objects and the forces that act on them and specifies a set of variables whose values can affect the System's behavior. However, the System itself is an unchangeable, state-free ("const") object. Instead, the values of its variables are stored in a separate object, called a *State* (more details below). Finally, a *Study* couples a System and one or more States, and represents a computational experiment intended to reveal something about the System. By design, the results of *any* Study can be expressed as a State value or set of State values which satisfies some pre-specified criteria, along with results which the System can calculate directly from those State values. Such a set of State values is often called a *trajectory*.

It is important to note that our notion of "state" is somewhat more general than the common use of the term. By state, we mean *everything* variable about a System. That includes not only the traditional continuous time, position and velocity variables, but also discrete variables, memory of past events, modeling choices, and a wide variety of parameters that we call *instance variables*. The System's State has entries for the values of all of these variables.

In an internal coordinate representation, our position coordinates are generalized coordinates q; our velocity coordinates are generalized speeds u. There are also auxiliary continuous state variables we denote z; these are used as state variables for force models, controllers, etc.

Structure of a System

A System is composed of a set of interlocking pieces, which we call *subsystems*.



In this jigsaw puzzle analogy, you can think of the System as providing the "edge pieces" which frame the subsystems into a complete whole.

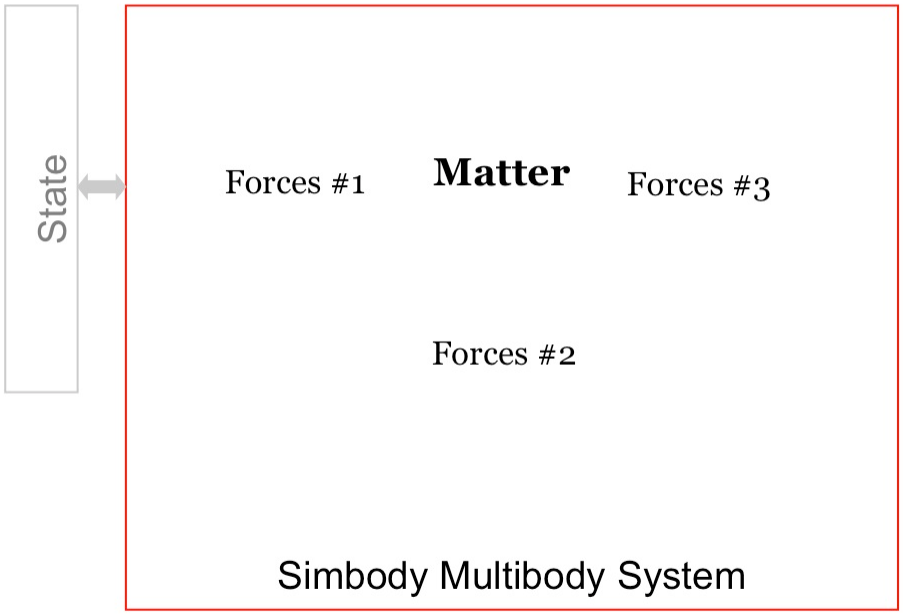
In general, any subsystem of a System may have its own state variables, as can the System itself. The System ensures that its subsystems' state needs are provided for within the overall System's State. The calculations performed by subsystems are interdependent in the sense of having interlocking computational dependencies.

Note that by design this is *not* a hierarchical structure. It is a flat partitioning of a System into a small number of Subsystems. In a higher-level modeling layer like OpenSim, you will find hierarchical models, which are a powerful way to represent the physical world. However, computational resources are flat, not hierarchical, and the SimTK System/Subsystem scheme is a computational device, not a modeling system. The intent is that a modeling layer (or user program) assembles a System from a small library of Subsystems just at the point when it is ready to perform resource-intense computations.

Structure of a multibody system

Let's look at Simbody in this context. Simbody primarily provides *one* computational subsystem (one puzzle piece) of a complete multibody mechanics System. This piece, called the SimbodyMatterSubsystem, manages the representation of interconnected massive objects (that is, bodies interconnected by joints). Simbody can use this representation to perform computations which permit a wide variety of useful studies to be performed. For example, given a set of applied forces, Simbody can very efficiently solve a generalized form of Newton's 2^{nd} law F=ma. On the other hand, Simbody is agnostic about the forces F, which come from domain-specific models. That is, Simbody fully understands the concept of *forces*, and knows exactly what to do with them, but hasn't any idea from where they might have come. OpenSim provides the remaining pieces, such as muscle force subsystems.

A complete System thus consists of both the matter subsystem implemented by Simbody, and user-written or OpenSim-provided force subsystems. So for a multibody system, the general SimTK System described above is specialized to look something like this:



Although both the SimbodyMatterSubsystem and the forces from subsystems require state variables, as discussed above, any SimTK System (including an OpenSim Model) is a stateless object once constructed. Its subsystems collectively define the System's parameterization, but the parameter values themselves are stored externally in a separate SimTK::State object.

For more information, see https://simtk.org/home/simbody, Documents tab. The SimTK Advanced Programming Guide, Simbody Theory Manual, and detailed Doxygen documents available there describe the individual classes and methods.

State realization

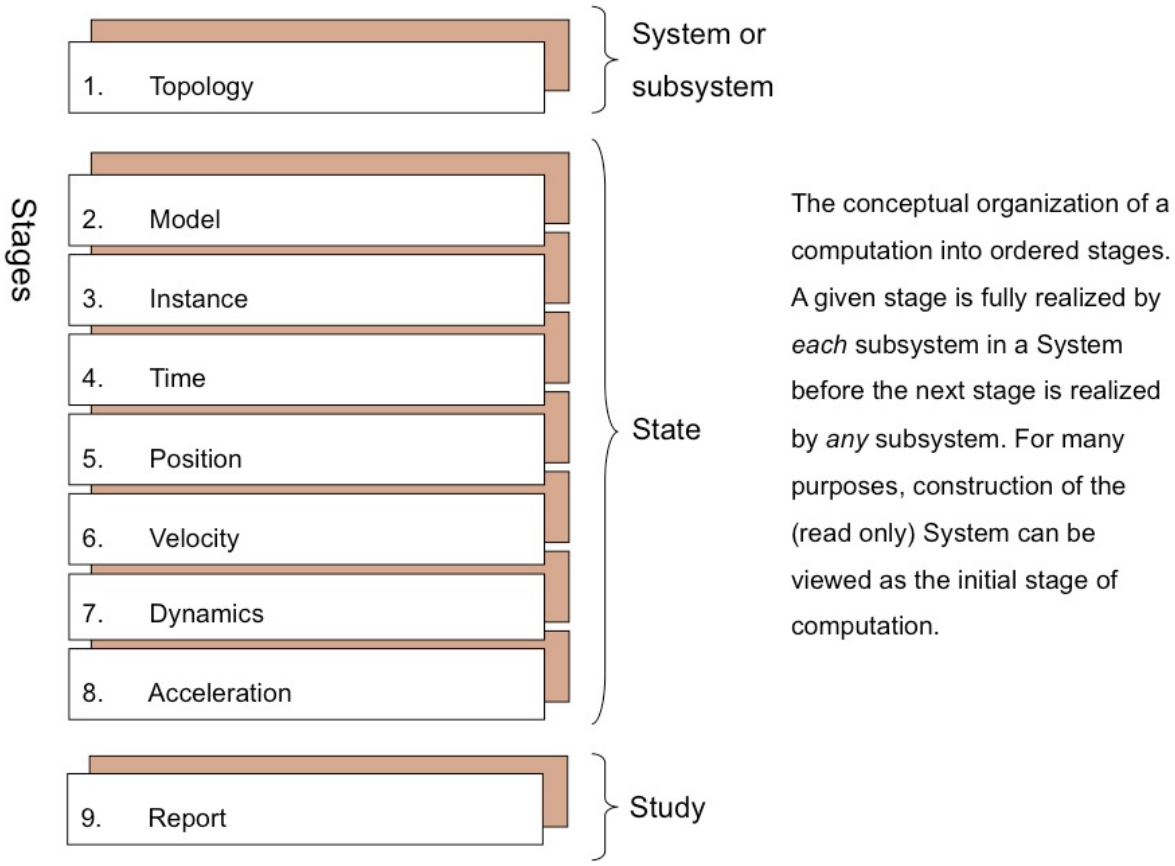
The state variables collectively represent a complete description of the state of a system at a given time. On the other hand, there are lots of other numbers you might want to know. Some examples include:

- The position of each body in Cartesian coordinates
- The force acting on each body
- The generalized acceleration of each internal coordinate

These are not independent pieces of information. Given the state variables, you can calculate them whenever you want. On the other hand, some of them may be expensive to calculate, so you want to avoid recalculating them more often than necessary. The State object therefore provides space for storing these derived values. This space is called the *realization cache*, and the process of calculating the values stored in it is known as *realizing the state*.

If you look at the list of examples above, you will see that they need to be calculated in a particular order. The Cartesian coordinates of each body generally need to be known before the forces can be calculated, and the forces need to be known before the internal coordinate accelerations can be calculated. It also is clear that not all of these pieces of information will be needed in every situation. If you only care about the positions of bodies, you don't want to waste time on an expensive force calculation.

The realization cache is therefore divided into a series of *stages*. Each piece of information in the cache belongs to a particular stage. When you want to realize part of the cache, you specify what stage to realize it up to. This causes the information belonging to that stage and all previous stages to be calculated. In other words, whenever you want to get some information from the cache, you must first make sure the state has been realized up to the stage to which that information belongs. The figure below shows all the stages.



The "Topology" stage is not part of the State; it represents the fixed contents of the System. "Model" stage is used for setting modeling choices, such as whether to use quaternions or Euler angles for joint orientation. "Instance" stage sets instance variables, such as masses, spring constants, attachment points, etc. Those stages are fixed during a simulation. The remaining stages change dynamically:

Time: At this stage, time has advanced and state variables have their new values, but no derived information has yet been calculated. You can query the State for time and any of the state variables, but nothing else.

Position: At this stage, the spatial positions of all bodies are known, along with related quantities such as separation distances.

Velocity: At this stage, the spatial velocities of all bodies are known, along with related quantities.

Dynamics: At this stage, the force acting on each body is known, along with the total kinetic and potential energy of the system.

Acceleration: At this stage, the time derivatives of all continuous state variables are known.

Report: A State is not normally realized to this stage during a simulation. It is available in case a System can calculate values that are not required for time integration, but might be needed for data output. That way, these values will only be calculated when they are actually needed.

The State makes sure that all values in the realization cache are consistent with the current state variables. If you modify any state variable, it will automatically "back itself up" to an earlier stage, invalidating cache entries from later stages so they can no longer be accessed. In particular:

- Changing an instance variable, such as a mass or spring constant, brings the State back to Model stage.
- Modifying time *t* will bring the State back to Instance stage.
- Modifying a generalized coordinate *q* will bring the State back to Time stage.
- Modifying a generalized speed *u* will bring the State back to Position stage.
- Modifying an auxiliary variable *z* will bring the State back to Velocity stage.
- When a System defines a discrete state variable, it specifies what stage the State should be reverted to when that variable is modified. This should be chosen to ensure that modifying the variable will invalidate any cache entry that may depend on it.

Next: Developer Pages

Previous: __40__Multibody Dynamics Concepts (Simbody)

Home: Scripting and Development | Developer's Guide | SimTK Basics