Simulate Your World

A deeper look at extending Gazebo

Goals

Give developers the tools to build additional Gazebo functionality

- Discuss the Gazebo simulation architecture
- Discuss the components of a Gazebo system
- Explain how Gazebo locates, loads, and runs custom plugins

Preview improvements to the Gazebo API for developer ergonomics

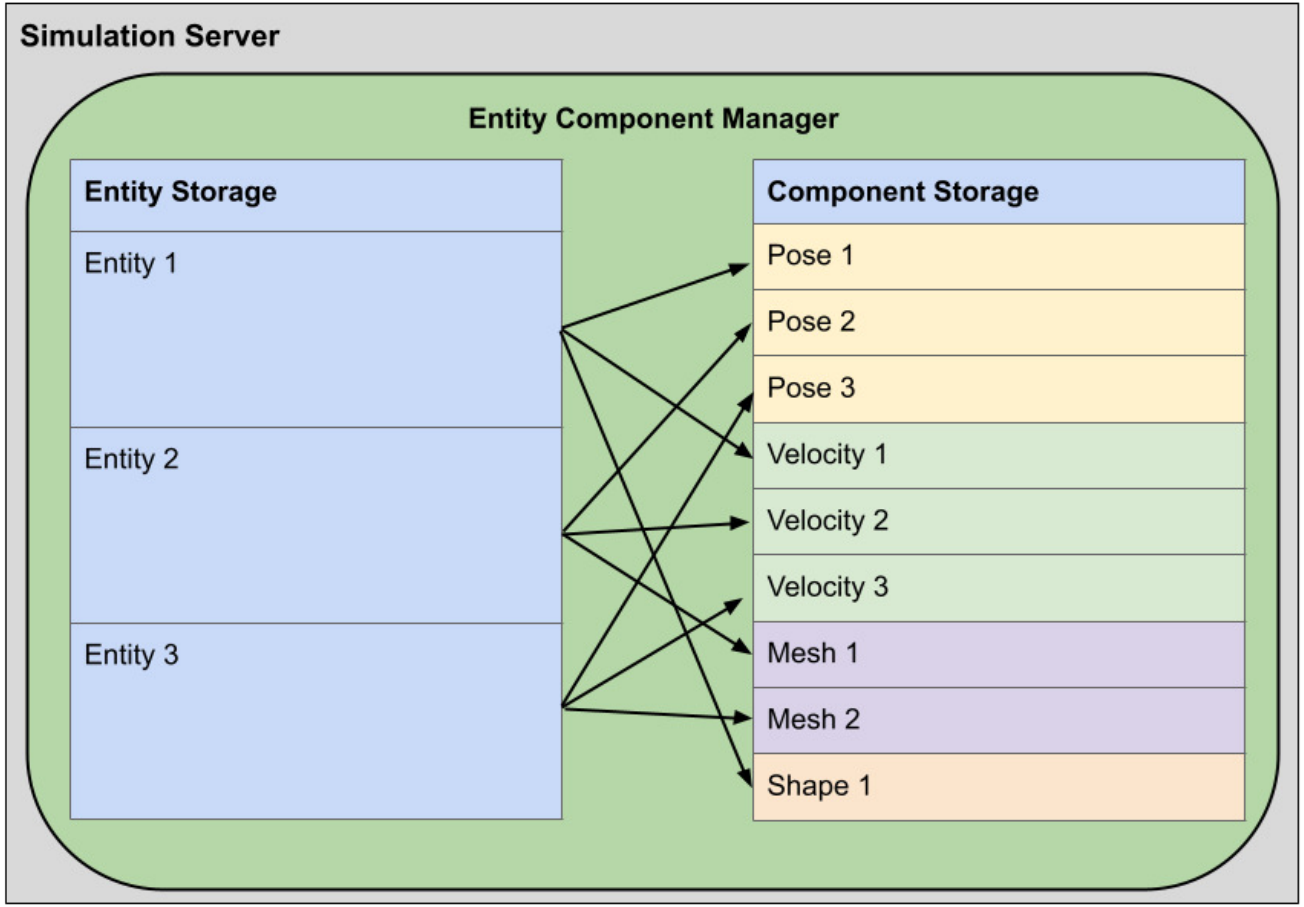
Gazebo Extension Points

Simulator Plugin Types

- **System Plugin -** Run in the gz-sim server, interacts with the EntityComponentManager to provide additional simulation functionality
- **GUI Plugin -** Run in the gz-sim client, provides additional GUI functionality

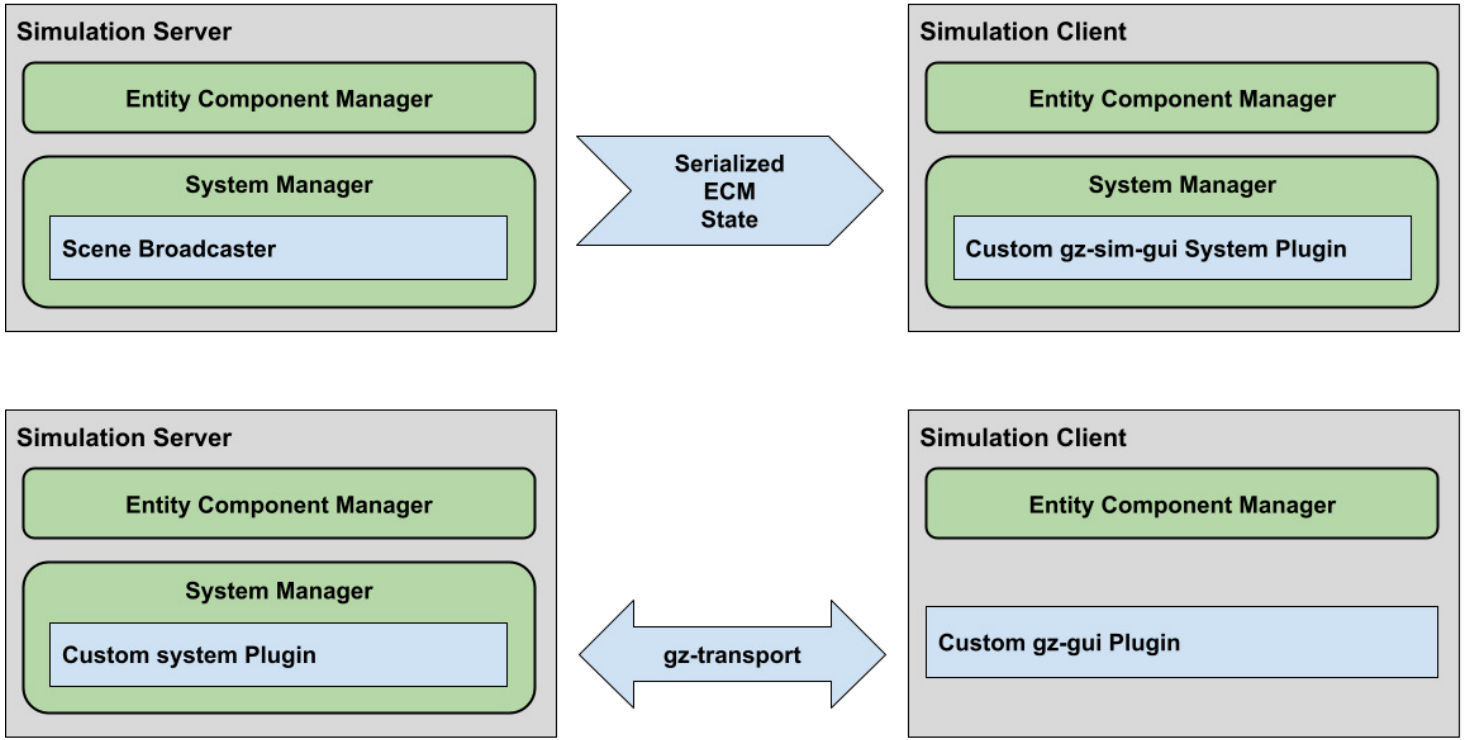
Other Plugin Types

- **Physics Engine Plugin-** Loaded by the physics system to provide a physics implementation (eg DART, Bullet)
- **Rendering Engine Plugin-** Loaded by the rendering system to provide a rendering implementation (eg OGRE, OGRE2)


Simulation Server
Entity Component Manager
Entity Storage
Entity 1
Entity 2
Entity 3
Component Storage
Pose 1
Pose 2
Pose 3
Velocity 1
Velocity 2
Velocity 3
Mesh 1
Mesh 2
Shape 1

GAZEBO

Client-Server Architecture



GAZEBO

What is a system?

In Gazebo, all behaviors are implemented in *systems*, which manipulate *entities* and *components*.

Systems are loaded either via specification in SDF files, programmatically via the Server API, or through the set of default system plugins.

Systems are packaged in shared libraries that are loaded at runtime.

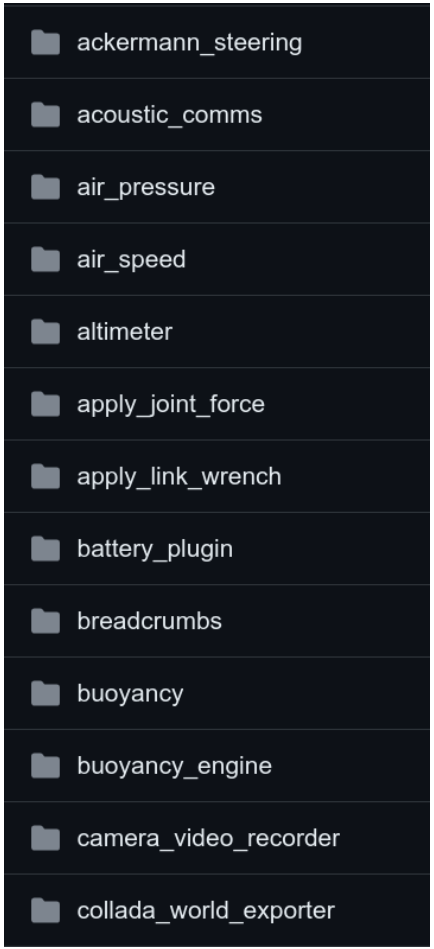
When should I develop a Gazebo System?

- Add *new* physics.
- Develop business logic.
- Load custom data
- Hardware-in-the loop
- Do lock-step physics simulation (for custom controllers)
- Custom GUI tools

When not to develop a system?

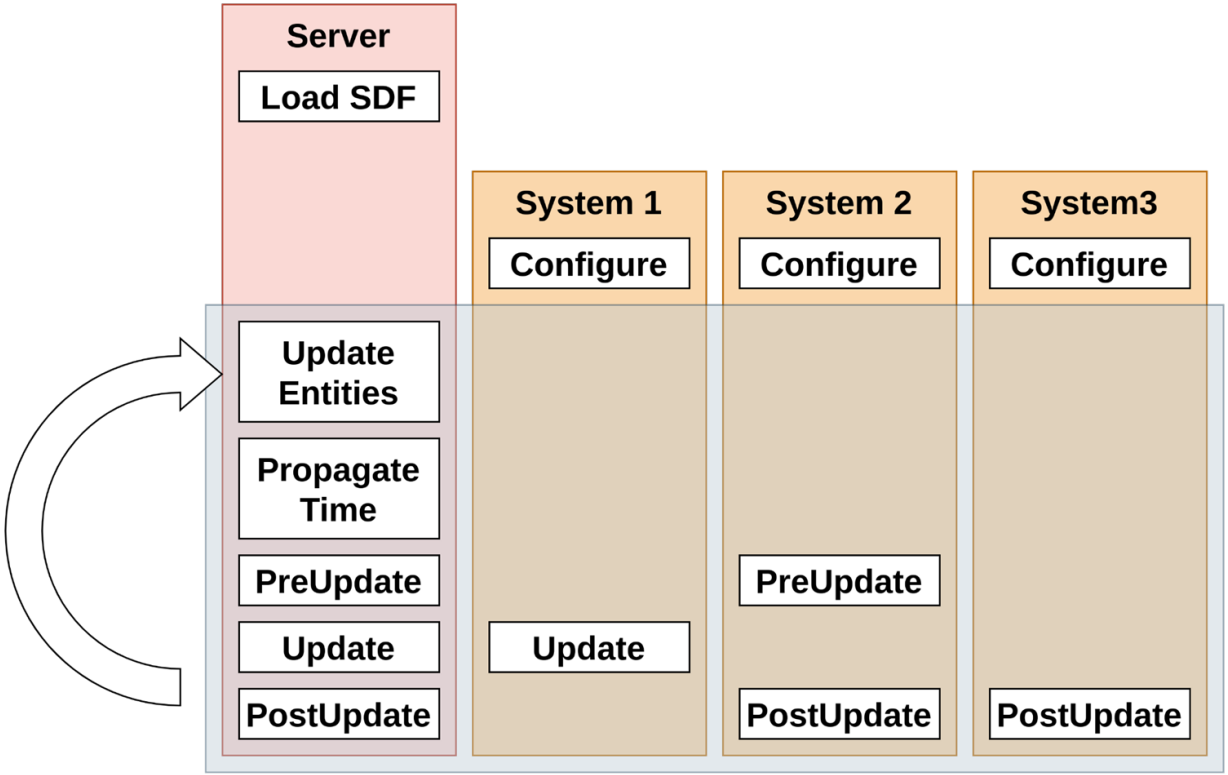
Before starting, verify that something doesn't already exist! (or closely match).

Full List of Systems:
https://github.com/gazebosim/gz-sim/tree/gz-sim8/src/systems



GAZEBO

The Gazebo simulation loop



GAZEBO

Loading Systems

```
<world>
<gui fullscreen="0">
    <plugin filename="Plot3D"
            name="Plot 3D">
     ...
   </plugin>
</gui>

<model name="car">
  <pose>0 0 0.35 0 0 0</pose>

  <include merge="true">
    <uri>car</uri>
  </include>

  <plugin
    filename="gz-sim-joint-state-publisher-system"
    name="gz::sim::systems::JointStatePublisher">
    <frame_id>car<frame_id>
  </plugin>
</model>
</world>
```

GUI plugins are loaded from `GZ_GUI_PLUGIN_PATH`

Models are loaded from `GZ_SIM_RESOURCE_PATH`

System Plugins are loaded from `GZ_SIM_SYSTEM_PLUGIN_PATH`

Passed to `System::Configure()`

System Run-Loop

Pre-update, Update, Post-update are run in sequence each simulation step

```
# PreUpdates get READ/WRITE access to the ECM
void PreUpdate(const gz::sim::UpdateInfo &_info,
               gz::sim::EntityComponentManager &_ecm;

# Reserved for physics update
void Update(const gz::sim::UpdateInfo &_info,
            gz::sim::EntityComponentManager &_ecm);

# PostUpdates can only READ state, run in parallel
void PostUpdate(const gz::sim::UpdateInfo &_info,
                const gz::sim::EntityComponentManager &_ecm);
```

Using Helper Classes

```
// Individual components are always available from the ECM
_ecm.Each<components::Model, components::ParentEntity>(
      [&](const Entity &_entity, const components::Model *,
          const components::ParentEntity *_parent) -> bool
      {
        auto pose = _ecm.Component<components::Pose>(_parent->Data())->Data();
      }
```

```
// or via "Helper" classes
auto model = gz::sim::Model(entity);
auto pose = model.Pose(_ecm);
```

Helper classes exist for: Model, Link, Light, Joint, and more

Testing your system

```
gz::sim::TestFixture fixture("../gravity.sdf");

fixture
   // Use configure callback to get values at startup
  .OnConfigure([](const gz::sim::Entity &_worldEntity,
      const std::shared_ptr<const sdf::Element> &/*_sdf*/,
      gz::sim::EntityComponentManager &_ecm,
      gz::sim::EventManager &/*_eventMgr*/)
    {
          // Make assertions about pre-conditions here
    })
  // Use post-update callback to get values at the end of every iteration
  .OnPostUpdate([](const gz::sim::UpdateInfo &_info,
      const gz::sim::EntityComponentManager &_ecm)
    {
      // Make assertions about runtime conditions here
    })
  // The moment we finalize, the configure callback is called
  .Finalize();

  // Run the server
fixture.Server()->Run(true, 1000, false);
```

Preview: Simple Simulation API

Motivation: Current API is either too high-level (gz::sim::Server) or non-deterministic (gz-transport) to write compact tests and conveniently embed Gazebo.

- Allow for an incremental design and rewrite in place.
- Allow for more deterministic execution
- Allow for more test coverage
- Create an API with first-class scripting support

Preview: Simple Simulation API

```
using SimulationBuilder = gz::sim::simulation::SimulationBuilder;

auto ecm = gz::sim::EntityComponentManager();
auto sim = SimulationBuilder()
    .World(std::filesystem::path("my_simple_world.sdf")
    .EntityComponentManager(&ecm)
    .Build();

auto model = sim.ModelByName(“sphere”);

for (size_t ii = 0; ii < 100; ++ii)
{
  std::cout << sim->IterationCount() << std::endl;
  sim->Step();
  std::cout << model->Pose() << std::endl;
}

sim->Reset();

std::cout << model->Pose() << std::endl;
```

Conclusion and Resources

- **Find systems that already exist:**
  https://github.com/gazebosim/gz-sim/tree/gz-sim8/src/systems
- **Get started quickly with system templates in ROS:**
  https://github.com/gazebosim/ros_gz_project_template/tree/main/ros_gz_example_gazebo
- **Learn how to migrate plugins from Gazebo classic:**
  https://gazebosim.org/api/sim/8/migrationplugins.html



GAZEBO