- clients state their required interfaces. Source and sink data need not have the same size or joint ordering.

- arbitrary combinations of sources and sinks are allowed, eg. a joint_states publisher containing only sinks (not shown).

- data layout is that of a structure of arrays (SoA), which allows clients to be written using fewer explicit loops.

- interfaces created dynamically when a client is loaded.

- only required joints and sinks/sources are exposed.

- data vectors list joints in the order required by the client.

- not all resources need to be used (see gray fields).

- below sinks have exclusive ownership, ie. can't be shared by multiple clients.

- below sources can have multiple clients.

Motivation: exposing joint control interdaces for a new hardware platform should be much easier, mainly consisting of 1. having a robot model description, 2. coding a platform-specific actuator_manager (driver + joint_controller_manager interface), and 3. setting up the deployment of the desired controllers by means of scripting / configuration files.

Example scenario: This figure corresponds to a robot consisting of a mobile base with two velocity-controlled joints, a position/velocity-controlled pan-tilt unit (ptu) and two arms with four effort-controlled joints each. Only one arm is being controlled.

Comments:
- sink/source types (position, velocity, acceleration, effort, stiffness, etc.) are not limited to a predefined set, but can be arbitrary, as long as the actuator manager, transmissions and client controllers know how to make sense of them.

- clients can be queried for the required interfaces, eg. (translated to human-readable) "position and velocity sink and source for joints head_1 and head_2". This part of the API is not shown in the figure, which focuses on the data flow.

- the joint_controller_manager expose an interface (also not shown) for loading/unloading and starting/stopping clients. It can optionally be exposed as a ROS service interface compatible with that of the pr2_controller_manager.

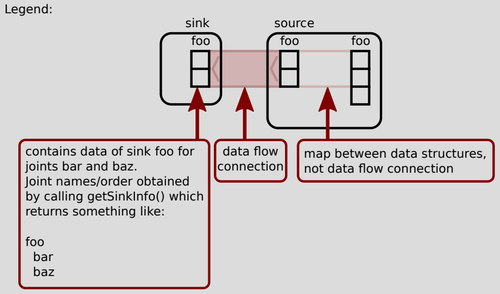
- client controllers can be chosen (at deployment time) to run serialized in the joint controller manager thread, or in a dedicated thread, allowing different update frequencies and even non-periodic updates. Also, running unstable/experimental clients in separate threads prevents them from interfering with mission-critical code paths (eg. mode switches, overruns).

- the time source and joint_controller_manager update rate should allow to be driven by wall and simulated clocks.

- a reference impletemtation of the computational parts (light gray blocks) should be provided, eg. separate spline interpolation and splicing code from current FollowJointTrajectoryActionController into a controller-agnostic unit that can be unit-tested in isolation.

- the proposed joint_controller_manager implementation is based on Orocos RTT 2.x.

Legend:



the joints controlled by this client don't have pos/vel sinks, but rather an effort sink...

...so an additional module is chained to convert pos/vel commands to effort commands: a PID controller.

- maps from required client interfaces to internal robot representation.

- maps are setup/cleaned up on client load/unload.

actuator source and sink ordering is the same as in actuator_manager.

available actuators and control interfaces.