Hydra render farm narrative

This document presents a series of stories about Hydra. The purpose is to work out what Hydra does, from the user's point of view as well as internally, and what objects and entities arise from these explanations. This will provide the framework for the initial implementation.

1. The user has a Maya scene open. The Maya project lives on a file server. All file references are relative or UNC paths. There are no drive letters.
2. The user presses the **Hydra Submit button** in Maya to bring up the **Submission GUI**.
3. The Submission GUI shows per-scene and per-layer parameters pertaining to rendering, e.g., frame range, which renderer to use, batch size, base job name, whether to render on the farm or locally. **This data** is stored in the Maya scene file as a Python object.
4. The user presses the **Submit button** on the Submission GUI. A **Submission object** is created in the **Database**.
5. Hydra interprets the Submission object as a tree of Job objects with a scene-layer-frame heierarchy. The leaves of the tree are **Render tasks,** which go into the **Render queue**.
6. The **Queue manager** sends render tasks to idle **Render nodes**.
7. Render nodes run Render tasks and update the Render task **status** in the Database.
8. Status updates percolate upwards, from frames, to layers, to jobs.
9. The user monitors job status via a **Job status GUI**.
10. The user monitors Render node activity via a **Farm status GUI**.

Background

Maya scenes typically have multiple render layers, for example to render objects, shadows, and specular highlights separately. This allows the elements to be adjusted later, for example to make shadows softer. This is where the scene-layer-frame heierarchy comes from: the scene has multiple layers, and each layer renders as a sequence of frames.

The user looks at the Job status GUI to answer the question: how is my job doing?

The user looks at the Farm status GUI to answer the question: what is the render farm doing?

All Hydra data is stored in a central database. Python objects are stored as database records, and one field of the record is the pickled object. For convenience some data members of the object may be stored as separate fields so that they can be searched or manipulated by database operations, for example when a job status needs to be updated.

All programs in the system connect to the same database. Instead of one program communicating data to another, the preferred method is for one program to put data in the database and send a short message to another, telling it to go look in the database. The design should be robust in the face of network failures, but not database failures.