Config thoughts

- Building on Phil's idea, we could separate each of Module, App, System become their own python module, one file per "entity"
  - Module: Some task
  - App: Some collections of tasks that run on the same host
  - System: Many apps on different hosts
  - Also separate out utilities: Connection, JsonExporter, CommandMaker
- In minidaqapp, we effectively have Apps in different files that have the function generate(…)
  - Could make this link explicit, and create a "Trigger" App class directly?
  - For example:

```
#====================================================================
class Trigger(App):
    def __init__(# NETWORK_ENDPOINTS: list,
            NUMBER_OF_RAWDATA_PRODUCERS: int = 2,
            NUMBER_OF_TPSET_PRODUCERS: int = 2,

            ACTIVITY_PLUGIN: str = 'TriggerActivityMakerPrescalePlugin',
            ACTIVITY_CONFIG: dict = dict(prescale=10000),
```

trigger_app.py

mdapp_multiru_gen.py

```
#--------------------------------------------------------------------
# Trigger app
the_system.apps["trigger"] = Trigger(
    NUMBER_OF_RAWDATA_PRODUCERS = total_number_of_data_producers,
    NUMBER_OF_TPSET_PRODUCERS = total_number_of_data_producers if enable_software_tpg else 0,
    ACTIVITY_PLUGIN = trigger_activity_plugin,
    ACTIVITY_CONFIG = eval(trigger_activity_config),
    CANDIDATE_PLUGIN = trigger_candidate_plugin,
    CANDIDATE_CONFIG = eval(trigger_candidate_config),
    TOKEN_COUNT = trigemu_token_count,
    SYSTEM_TYPE = system_type,
    TTCM_S1=ttcm_s1,
    TTCM_S2=ttcm_s2,
    TRIGGER_WINDOW_BEFORE_TICKS = trigger_window_before_ticks,
    TRIGGER_WINDOW_AFTER_TICKS = trigger_window_after_ticks,
    HOST=host_trigger)

console.log("Trigger module graph:", the_system.apps['trigger'])
```

Config thoughts

- Most of the files present in minidaqapp could become an App subclass
  - Maybe these should go in their original repo, for example trigger_gen.py in the triggers repo (i.e. maintenance is delegated to the people who wrote the C++ code), and minidaqapp would just plug all the Apps together…?
  - I find it weird that minidaqapp has to change every time a new feature is included.
- Some Apps need to know the full state of the system to create connections to them, for example the MLT:
  - Could implement a System.finalise(), that goes into every app, module and execute user code.
  - In the case of the trigger, Trigger.finalise(system) could be setting all the MLT connections etc…
  - Some of this should disappear after network manager, I haven't looked at it

Config thoughts

- Discussion this morning with Alex…
- Layered configuration generation:
  - 1st step: local configuration (1 APA) → App class (right now generate functions)
  - 2nd step: “glue” them together, global configuration (150 APAs) → System class (right now in mdapp_multiru_gen.py)
  - 3rd step: assign host and end point (75 readout hosts) → K8s? (right now in mdapp_multiru_gen.py)
- The main idea is that you can only go downwards, but you can reuse earlier stages in different way
- Some complications
  - Difference between local and global is not so trivial
  - Where should command generation happen?
  - What do the intermediate configuration look like and how do we pass them around?
    - There should be a fair bit of placeholders for hosts, network endpoints…
    - First pass idea:
      - 1st step: save the arguments of the App.ctor (outside minidaqapp)
      - 2nd step: create an even bigger json which specifies the whole system (in minidaqapp), for that, maybe we can use networkx saving facility?
      - 3rd step: replace host and endpoint placeholders with reality (in minidaqapp)

mdapp_multiru_gen.py

```
# Trigger app
trigger_kwargs = json.load("trigger_conf.json")
trigger_kwargs.update({
    "NUMBER_OF_RAWDATA_PRODCERS": n_raw_prod_from_cli
    # ...
})
the_system.apps["trigger"] = Trigger(**trigger_kwargs)
# initialise all the apps here
#...

# Save the system with placeholders network endpoints and host
the_system.save("system.json")
```