Proposal for higher level abstraction of flowsheet models, flowsheet units, and automated flowsheet construction

Introduction: There is a need for an abstraction layer that can support developing preconfigured unit models and flowsheets, that could enable automated construction of complex flowsheets and their use in GUI. Currently, unit models in IDAES/WaterTAP, etc are designed as abstract units that can be modified/used in an infinite number of ways, but many users want standard units that can be used right away for typical process simulation and analysis (example: Reverse osmosis, requires pump, RO unit and maybe ERD. This means a user needs to build a flowsheet with at least these three unit models to do a basic analysis, or adapt another flowsheet)

Furthermore, there is a need to be able to easily assemble flowsheet units in any number of configurations, for example multi-stage processes or where unit order is changed. Right now this is done manually, resulting in the development of bespoke flowsheets and unit model configurations that are not portable, in many cases these flowsheets grow in complexity and become impossible to maintain or navigate.

If a standard rigid structure is adopted for how flowsheet units are configured and built, it is possible to:

1. Develop flowsheet units that can be duplicated and used in different places on a flowsheet
2. Provide intuitive API for UI software or higher-level code-based API to interact with unit models
3. Provide an automated method to assemble flowsheets, initialize them, and perform analysis.

I propose the development of three classes within IDAES to support this capability:

1. **FlowsheetBase** – provides base configuration for the flowsheet
    - This houses utilities/properties/functions accessible to all unit models on the flowsheet, and all unit models are built on the FlowsheetBase
2. **FlowsheetUnit** – This houses all information for unit designed to be built on a flowsheet base, unlike a normal unit model, these units have to be preconfigured for a standard rigid operation expected of the underlying sub-unit model.
3. **FlowsheetBuilder** - This is a tool that would manage the FlowsheetBase and FlowsheetUnits to provide automated construction, initialization, and scaling of unit models in order requested by user using simple API calls (Similar to flowsheet_builder tool – for example using strings to configure order of unit operations), this would act as code based UI.
    - This might not be a block constructed on a model, but a separate tool entirely…

Basic concept

The basic concept of FlowsheetBase and FlowsheetUnit works on the concept of binding the given treatment train and subunits in boxes that have starting and terminating points that user needs to connect. This is functionally identical to port structure we use in IDAES right now.

The general idea, is that the flowsheetbase should never know what units are built on it, and flowsheetunits should never know what unit is upstream or downstream of their starting/termination points. This means that any initialization routing, scaling routine, and configuration have to be defined based on the starting point configuration. This bodes well for any UI implementation, where you don't know a-priori how a user will connect any unit models together, and forces developers to think about how to configure their model to work with other units, when starting point composition is not explicitly known beyond an expected reasonable range. This also in genral will provide more robust unit models and improve analysis as it forces developers to provide more flexible bounds on the models, as well as pay more careful attention to scaling. The advantage of this structure is now its possible to change order of unit operations, add new unit operations, and recycle unit models for new analysis with ease.

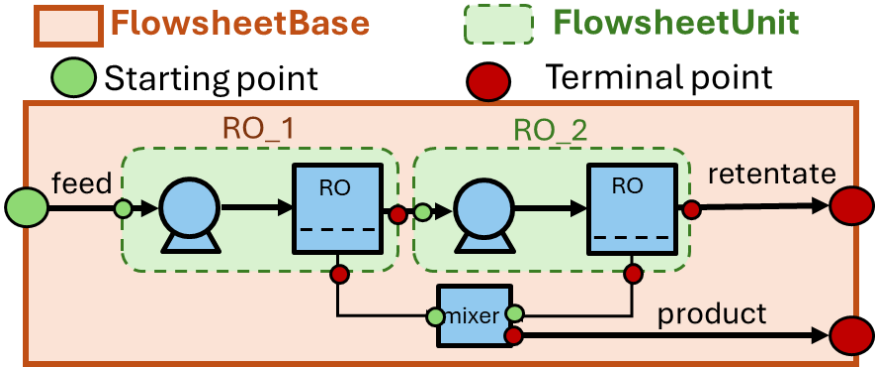

Figure 1: Conceptual drawing of flowsheetbase and flowsheetunit bounding boxes, starting, and termination points that need to be declared.

General requirements for FlowsheetBase and FlowsheetUnit:

- FlowsheetBase defines flowsheet level information
    - Property packages
    - Costing packages
    - Tool/information for unit models to access – (maybe location of data files, or optional arguments)
    - Defines global objectives and constraints
        - Costing minimization objective
        - Water recovery constraint
        - Function to drive developers mad
- FlowsheetUnits are developed for specific FlowsheetBase
    - FlowsheetBase defines global property, costing, etc. packages, unit models using different property, costing, etc. packages would need to provide translation blocks designed to work with FlowsheetBase global property packages.

- FlowsheetUnit and FlowsheetBase have defined starting and termination points, the starting point are used for initialization and scaling.
- Flowsheets are constructed from FlowsheetBase starting point to FlowsheetBase termination point.

Functions for FlowsheetBase and FlowsheetUnit classes:

- **Metadata (this would be a separate class)**: Provide methods for registering metadata. This information would be stored in a structure that is accessible to control API (e.g. FlowsheetBuilder, GUI)
    - Starting and termination points
        - Defines standard names and ports which are act starting point for flowsheet or unit and end points
    - Possible configurations (e.g. different default values, different building options)
        - For units that expect N number of starting/terminal points provide a default methods of declaring number/names for such points
            - E.g mixer/splitter – the number of starting/terminal points will depend on flowsheet design and would have to be constructed on the fly.
    - Variables that users should have access to control unit operation
        - Fixed variables for initialization and optimization
        - Constrain activation/deactivation
        - Information variables (those that should be printed or reported to user)
    - The metadata should be available before unit models are constructed, as control API might modify variables or parts of metadata before building a unit (for example, in case of a splitter or mixer unit, providing the list of streams that should be constructed based on flowsheet design)
        - This might not be possible, as variables that will be constructed do not exist, and cannot be registered in the metadata, a solution is provide a holding class, that holds the expected variables and their values, and registers the actual pyomo vars/constraints upon unit construction. This might mean more over head for the developer, but might be more reliable for debugging/testing, and control type API needs – where we might want to access variables and their options before building a unit model or full flowsheet…
- **Build method:**
    - This is a method to build the model, this should accept user-provided model configuration options to ensure it is constructed correctly
- **Initialization method:**
    - This should do standard initialization routines, but any guess for the model should be derived on the starting point state, the initialization function should also have access to the configuration of the unit model, in case the initialization routine is affected (for example by the addition of a sub-unit model)
        - For FlowsheetBase we will need to have 2 initialization functions:
            - Starting point – initialize anything before unit models (e.g. feed etc.)

        - Termination points – initialize anything after unit models (e.g. product, costing etc.)
- **Scaling before initialization:**
    - This should scale unit models based on assumed process operation, typically using the feed composition on the flowsheet-base and projected unit operation
        - This is really necessary to support iscale.calculate_scaling_factors(m). Ideally, scaling would be provided sequentially as each model is initialized.
- **Scaling after initialization:**
    - This should be used to update scaling after a model is initialized or after a solve.
- **Set unit operation mode (this ideally will be automated through the use of provided metadata)**:
    - This method would configure the unit for specific operations selected by the user or API, for example, for box solve (0DOF) or optimization
        - Requires 0DOF operational mode
        - Optimization operational mode is optional (not all models are optimizable)
    - Ideally, we use provided metadata to fix/unfix variables and activate/deactivate specified constraints to achieve the desired operation based on configuration
        - Each "configuration" would have its own 0DOF and optimization modes
- **Reporting method (this ideally is automated through provided metadata)**:
    - Method to print out model results and details
- FlowsheetUnit should have access to FlowsheetBase at all times (e.g within any function of FlowsheetUnit it should be possible to access base model level and any properties/function/variables constructed on it)

FlowsheetBuilder requirements:

This is a control API class for constructing flowsheets, and should imitate in a way a GUI would interact with FlowsheetBase or FlowsheetUnits. This is a code base API for more straightforward construction of full flowsheets:

Functions for FlowsheetBuilder:

- **Registration methods for available flowsheet bases and flowsheet units**
    - Registers FlowsheetBase and configuration for it
    - Registers FlowsheetUnit and configuration for it
    - Each configuration can be given a unique name to identify a unique unit configuration
        - RO (Just Pump+RO unit)
        - BWRO (Pump+RO unit with max pressure of 40 bar)
        - SWRO (Pump+RO+ERD unit with max pressure of 80 bar)
        - All there use same FlowsheetUnit model of RO, but accept configuration options to build them in a distinct manner. This is similar to user selecting a unit model in a UI, selecting configuration options for it, and giving it a poor name that causes errors (e.g. 1337R0~*&#@)

    - Special unit registration methods:
        - Mixer registration method – connects unit models when a mixer is requested
        - Splitter registration method – splits streams when requested in flowsheet
        - Terminal point mixer – used to mix terminal points from multiple units and connect them to final terminal point on FlowsheetBase – this unit might have different requirements compared to normal mixer.
        - Splitter/Mixers will need to have required configurational arguments for constructing mixed/split streams by control API
            - Refer to notes on Flowsheet Unit requirements above
            - Each registered mixer/splitter can have a unique configuration – e.g. more than 1 mixer can be registered to support multiple roles on the flowsheet.
- **Method for designing and building flowsheet:**
    - This should be a method that takes in an input that defines how a flowsheet is constructed, specific to its unit operation order etc.
        - An example might be using a string feed>RO>ERD>retentate, but a more sophisticated and less error-prone method would be preferred.
        - This would not be done in interactive method…
- **Method for initialization of flowsheet:**
    - This would go through the initialization of the flowsheet using the provided construction order.
- **Reporting method:**
    - Simple printout from all constructed unit models
- **Configure fixed unit operation method:**
    - This would configure unit models to 0DOF operation for initialization or simulation
- **Configure optimization operation**
    - This would configure operation for optimization with the objective

Ideally, the flowsheet builder would construct a symbolic representation of the flowsheet and verify all connections before building the actual unit models and initialization order. This would simplify debugging. We might also want to use FlowsheetBuilder to provide flowsheet visualization capability.