CHAPTER 11 THE MODEL LIBRARIES FOR MULTI-COMPONENT, MULTI-PHASE EQUILIBRIUM CALCULATIONS

This chapter describes the models we provide to compute thermodynamic properties for multi-phase, multi-component vapor/liquid mixtures where we assume equilibrium exists among co-existing phases.

11.1 A DESCRIPTION OF THE LIBRARIES

In this section we describe the three libraries, *phases.a4l*, *components.a4l* and *thermodynamics.a4l*. These libraries contain many models, but the end user is only interested in a few of them. Our intention is that these few should be very simple to use, with the complexities buried inside the models.

first the phase definitions

The first contains the models we use to define the phases we allow for a mixture (i.e., vapor, liquid, vapor/liquid, liquid/liquid and vapor/liquid/liquid)[1].

then the components and their data

The second library contains the models having all the component physical properties for the components we include with ASCEND — e.g., there are property values for heat capacity, heat of vaporization, accentric factor and so forth for water, methanol, carbon dioxide, etc. There is also the very extensive list of group contribution data we need to use the UNIFAC method.

and finally the mixture thermodynamic models

The third provides the models we use to compute multi-component mixture thermodynamic properties for phases, such as ideal gas, Pitzer, UNIFAC, and Wilson. The final model in this library is the one to compute equilibrium conditions for multi-component, multi-phase systems. We provide both a constant relative volatility and a rigorous phase equilibrium model, with the ability to switch interactively between which one to use. Thus one can first assume constant relative

1. It should be noted that, while the models will correctly set up the data structures for the liquid/liquid and vapor/liquid/liquid options, we do not really support these alternatives at this time.

volatility to have a better chance to converge and then switch to the version that makes the chemical potential equal for a component in all phases.

11.1.1 THE *PHASES.A4L* LIBRARY

need to create only instances of *phases_data*

The *Phases.a4l* library, see Figure 11-1[2], has only one model in it, **phases_data**. The user creates an instance of this model, specifying which phases are to exist for a stream or holdup and which thermodynamic model the system should use to compute mixture properties for each phase. Compiling this instance then sets up the data structures required to characterize those phases for the system.

For example, suppose we want to model a flowsheet consisting of a single flash unit. Suppose further that we want to allow the feed to the flash unit to be vapor, liquid or vapor/liquid (i.e., 2 phase). The product streams from the flash unit will be a vapor phase mixture and a liquid phase mixture. We would define three instances of the phases_data model, one for each type of phase condition we wish to model. You can find the following statements in the model *testflashmodel* in the library *flash.a4l*.

```
pdV IS_A phases_data('V', 'ideal_vapor_mixture', 'none', 'none');
pdL IS_A phases_data('L', 'none', 'UNIFAC_liquid_mixture', 'none');
pdVL IS_A phases_data('VL', 'ideal_vapor_mixture', 'UNIFAC_liquid_mixture',
                      'none');
```

When compiled, *pdV*, *pdL* and *pdVL* contain the data structures the thermodynamic models require to model a vapor, liquid and vapor/liquid stream (or holdup).

---

2. In this and following figures, we represent each model as a rectangle. On the upper left is the name of the model. In Figure 11-1, the model is phases_data. On the left side we list in order the parameters for the model. These are shared objects a model containing an instance of phases_data will pass to that instance. An example would be

   ```
   pd IS_A phases_data(V, 'Pitzer_vapor_mixture', 'none', 'none')
   ```

   We list the parts defined locally within a model on the right side of the rectangle, including instances of models, atoms and sets. The slanted double-headed arrow indicates a set; thus, phases and other_phases are sets in phases_data.

   In Figure 11-3 we show lines connecting a model, call it *A*, to a part within another model, call it *B.part*. The connection is to the sides of both. This type of connection says *B.part* is an instance of model *A*. We also show connections from the bottom of one model, call it *C*, to the top of another, call it *D*; with this connection we indicate that the lower model *D* is a refinement of the upper model *C*.

the phase indicators and types

The first parameter is a character that indicates the phase option desired - 'M', 'V', 'L', 'VL', 'LL' and 'VLL'. ‘M’ is for a material only stream (no thermodynamic properties are to be computed), ‘V’ is for vapor and ‘L’ for liquid. This model always expects the user to supply in the last three parameters an ordered list giving the three single phase mixture models to be used: vapor, liquid1, liquid2. For a non-existent phase, the user should supply ‘none’ as the model. If there is only one liquid phase, liquid2 will not exist. The allowed models we can use to estimate multi-component phase mixture properties are in the third of the libraries we describe in this chapter, *thermodynamics.a4l*, which we discuss shortly in Section 11.1.3.

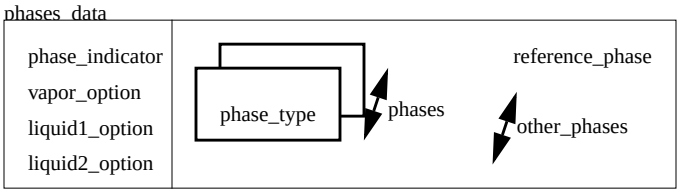

Figure 11-1 *Phases.a4l* models

11.1.2 THE *COMPONENTS.A4L* LIBRARY

In this library (see Figure 11-2) we provide the actual physical property data for the components supplied with ASCEND. The data we provide is that found in the tables at the back of Reid, Prausnitz and Poling, The Properties of Vapors & Liquids, 4th Ed, McGraw-Hill, New York (1986). For a few of the components, we have also identified their UNIFAC groups. We include a few Wilson binary mixture parameters.

need to create only instances of *components_data*

The purpose of this library is similar to the *phases.a4l* library. We wish to provide an easy-to-use model that will set up the data structures for the components in a mixture that the thermodynamic models will use when estimating mixture physical properties. All the user has to do is create an instance of the bottom-most model *components_data*, passing into it a list of the components in the mixture and the name of one of them which is to serve as the reference component. This model, having parts which are instances of the others present in this library, then compiles into the needed data structures.

An example of use is found in the model *testflashmodel* in the library *flash.a4l*:

```
cd IS_A components_data(['n_pentane','n_hexane','n_heptane'],'n_heptane');
```

When compiled *cd* has in it a data structure containing the physical properties for the three species listed.

reference component

The choice of which species to use as the reference component is up to the user. Usually a good choice is one that is plentiful in the mixture, but that need not be so.

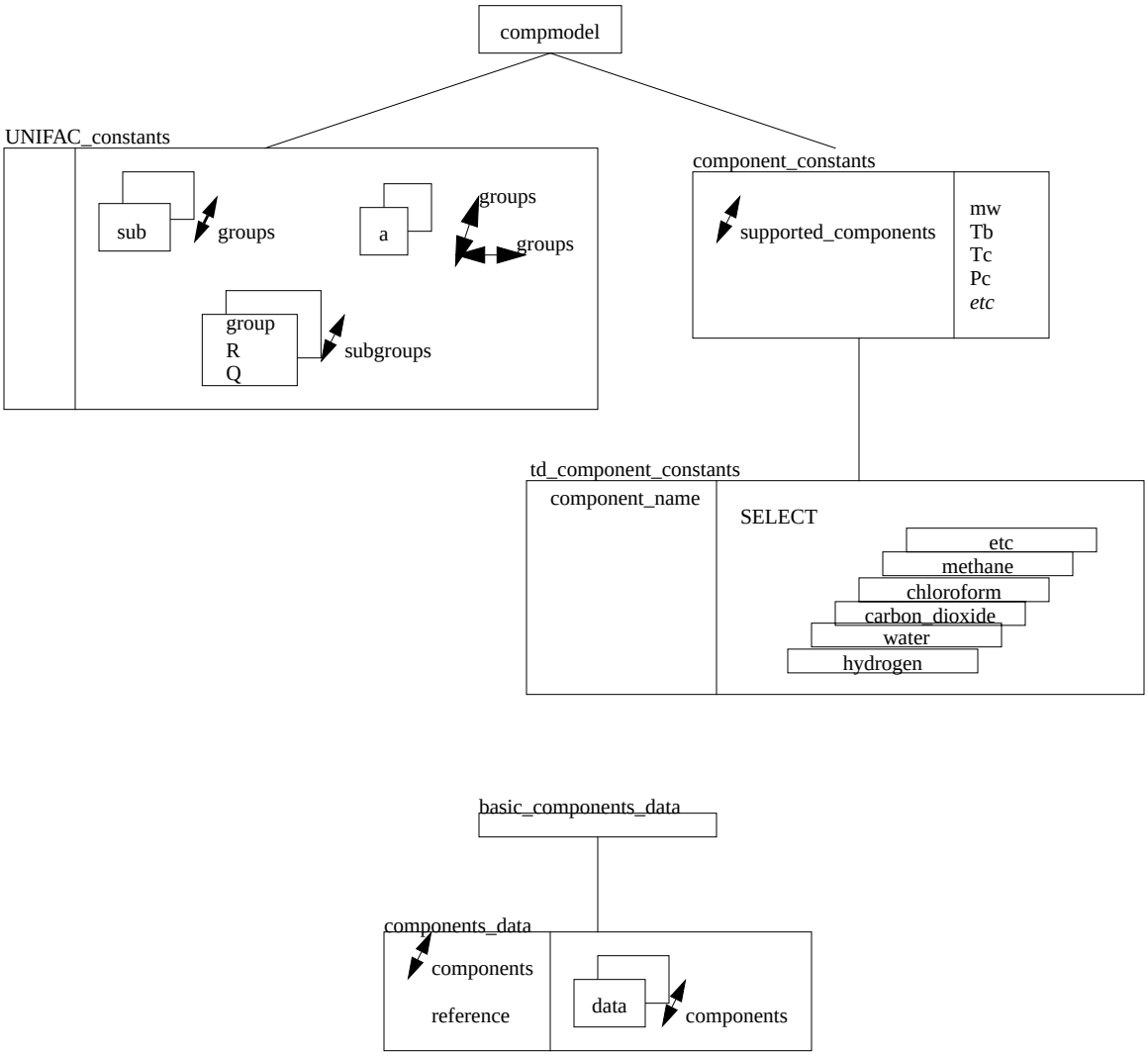

Figure 11-2 *components.a4l* models

adding a new component

One can add more components to this library as follows:

1. add the name of the new component to the list of *supported_components* at the beginning of the model *td_thermodynamic_constants* (part of the WHERE statement that causes the system to output a diagnostic if someone subsequently

misspells the name of a component)

2. add the component data as a CASE to the SELECT statement in *td_thermodynamic_constants* (for an example, look at how it is done for 'methanol')

adding UNIFAC group identifiers

3. Put the UNIFAC group identifiers for the new component into the set subgroups. To illustrate, this statement for methanol is:

```
subgroups      :== ['CH3', 'OH'];
```

You can find all the UNIFAC group identifiers possible in the model *UNIFAC_constants*. Then fill in the vector *nu* with a value for each of these groups (to indicate how many such groups are in the molecule). To illustrate, the values for methanol are:

```
nu['CH3']      :==1;
nu['OH']       :==1;
```

If you are entering the component without identifying its UNIFAC groups, then enter the subgroups statement and define it as empty — i.e., write

```
subgroups      :== [ ];
```

There should then be no entry for *nu* (see the CASE for hydrogen, for example). An activity coefficient estimated by the UNIFAC method will be unity for such a component.

adding Wilson parameters

4. To add Wilson parameters, first fill in the names of the other components for which you are adding data into the set *wilson_set*. For example, this set for methanol might be:

```
wilson_set     :== ['H2O','(CH3)2CO','CH3OH'];
```

Then fill in lambda and energy parameters into the arrays *lambda* and *del_ip*, one for each of the other components. Again, to illustrate, these arrays for methanol would be:

```
lambda['H2O']     :==0.43045;
lambda['(CH3)2CO']:==0.77204;
lambda['CH3OH']   :==1.0;
del_ip['(CH3)2CO']:==2.6493E+002 {J/g_mole};
del_ip['H2O']     :==1.1944E+002 {J/g_mole};
del_ip['CH3OH']   :==0.0 {J/g_mole};
```

Finally for each of these other components, go to its CASE statement, add the name of the new component to its *wilson_set* and then add statements to set the corresponding lambda and energy data. BEN, IS THIS RIGHT????

If you are not adding any Wilson data, enter the statement:

```
wilson_set     :== [ ];
```

11.1.3 THE *THERMODYNAMICS.A4L* LIBRARY

create instances only of *phase_partials* and *thermodynamics*

Figure 11-3 shows all the models in this library and how they are related to each other. There are two models in this library that the user has to worry about: *phase_partials* and *thermodynamics*. The user creates one instance of *thermodynamics* for every stream or holdup in a process model. Each instance, when compiled has parts which are instances of the other models in this library and which are create the equations to compute the thermodynamic properties for a multi-component, multi-phase mixture.

However, the user must pass each instance of a thermodynamics model an array of instances of *phase_partials*, one for each phase in the mixture. One *phase_partials* model must exist for each phase in each stream or holdup in the process model as it provides the equations modeling that phase.

Each of the models in the array of phase_partials must be refined to be one of the possible models for computing properties for a single phase mixture, i.e., one of the models lying below the *phase_paritals* model in Figure 11-3: *ideal_vapor_mixture*, *Pitzer_vapor_mixture*, *UNIFAC_liquid_mixture* or *Wilson_liquid_mixture*. I

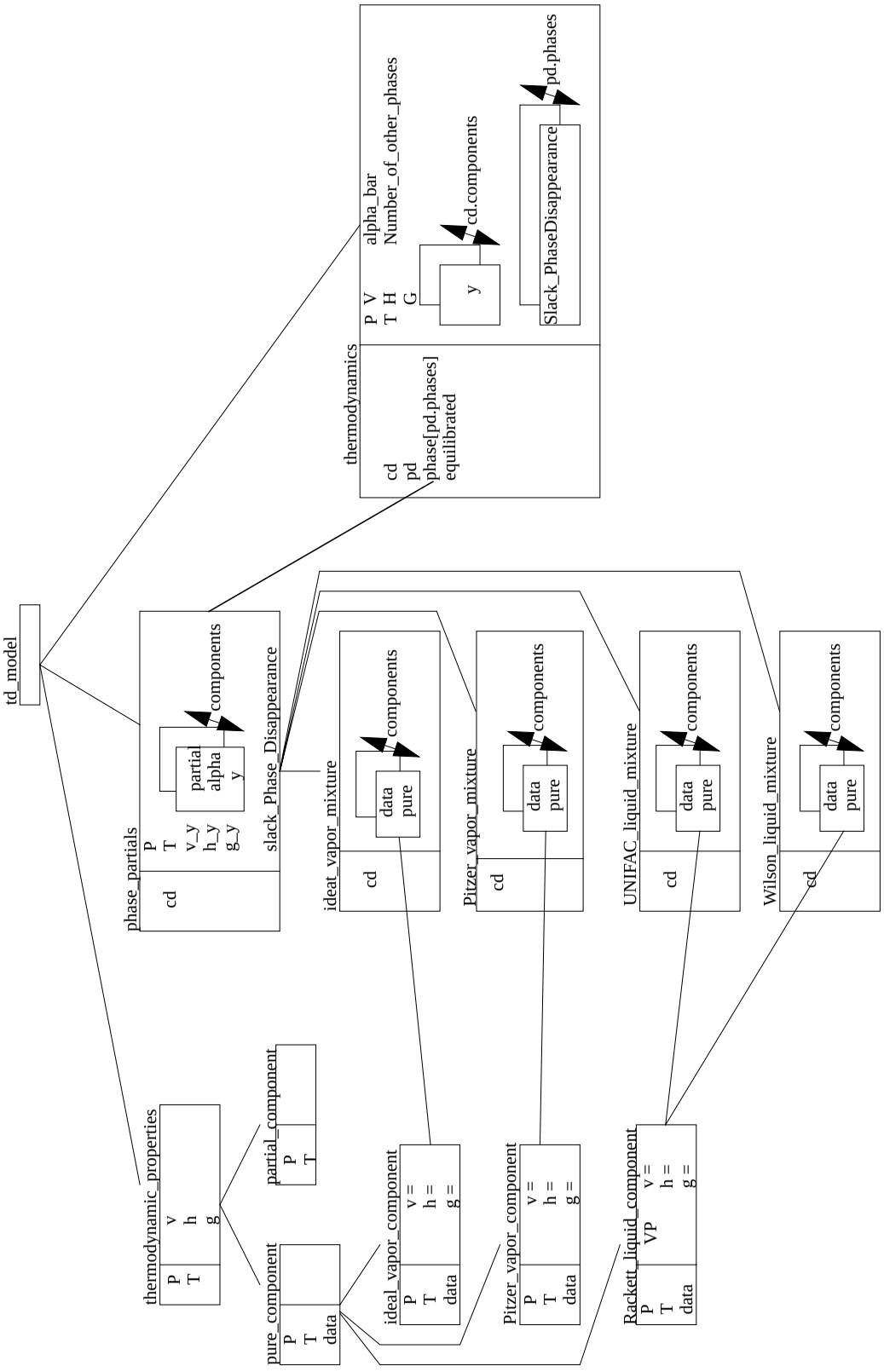

Figure 11-3 Models in *thermodynamic.a4l*

11.1.3.1 CREATING AN INSTANCE OF A *PHASE_PARTIALS* ARRAY

The information in an instance of a *phases_data* model allows us to construct this array of *phase_partials*. We extract the following code from the library *stream_holdup.a4l* to illustrate how we have created such a model, given a phases_data model.

```
MODEL select_mixture_type(
    cd WILL_BE components_data;
    type WILL_BE symbol_constant;
) REFINES sh_base;
   phase IS_A phase_partials(cd);
   SELECT (type)
      CASE 'ideal_vapor_mixture':
         phase IS_REFINED_TO ideal_vapor_mixture(cd);
      CASE 'Pitzer_vapor_mixture':
         phase IS_REFINED_TO Pitzer_vapor_mixture(cd);
      CASE 'UNIFAC_liquid_mixture':
         phase IS_REFINED_TO UNIFAC_liquid_mixture(cd);
      CASE 'Wilson_liquid_mixture':
         phase IS_REFINED_TO Wilson_liquid_mixture(cd);
      OTHERWISE:
   END SELECT;
   boundwidth IS_A bound_width;
   ...
   ...
   ...

END select_mixture_type;

MODEL stream( .......
   ...
   ...
   ...

   FOR j IN phases CREATE
      smt[j] IS_A select_mixture_type(cd, pd.phase_type[j]);
   END FOR;
   FOR j IN phases CREATE
      phase[j] ALIASES smt[j].phase;
   END FOR;
   state IS_A thermodynamics(cd, pd, phase, equilibrated);
   ...
   ...
   ...
   ...
```

cannot directly embed *SELECT* statements in *FOR* loops

We had to be a bit tricky, but we hope we have not been so devious that you cannot understand what we have done if we explain it to you here. Look first at the code we extracted from the model *stream*. The models *cd* and *pd* are instances of a *components_data* and a *phases_data* model respectively. If we look inside *pd*, we will find it contains an array called *phase_type*, with one entry for each phase that gives the type (name) of the model to be used to set up the equations for that phase. ASCEND does not allow *SELECT* statements to be embedded directly within a *FOR* loop — thus we need a bit of deviousness. For each phase *j* we create *smt[j]* as an instance of a *select_mixture_type* model. We parameterize the *select_mixture_type* with the components data *cd* and the type (name) *pd.phase_type[j]* of the model to be used to generate its equations. Then we embed the select statement within the *select_mixture_type* model, something ASCEND does allow.

The model *select_mixture_type* appears first in this code. It uses the *type* (name) it is passed to select and then to instance the desired refinement of the *phase_partials* model.

Returning to the code extracted from the *flash* model, the second *FOR* loop creates the desired array by aliasing the array element *phase[j]* with the phase model created within the corresponding *smt* instance.

disappearing phases

The multi-phase model handles the case where a phase disappears by using a complementarity formulation. This formulation relaxes the constraint for a phase that its mole fractions must sum to unity when it disappears. Thus the vapor/liquid model will correctly alter the model to handle the situation when the mixture becomes a superheated vapor or a subcooled liquid.

11.1.3.2 CREATING AN INSTANCE OF A *THERMODYNAMICS* MODEL

We are now ready to create an instance of a *thermodynamics* model. When compiled this instance contains all the equations needed to estimate the phase conditions for a multi-phase, multi-component mixture assuming equilibrium exists among the phases. The following line of code, extracted from the *stream* model referred to above, illustrates its use:

```
state IS_A thermodynamics(cd, pd, phase, equilibrated);
```

where *cd* is an instance of a *components_data* model, *pd* of a *phases_data* model, *phase* an array of instances of *phase_partials*, and *equilibrated* a *boolean* variable. When *equilibrated* is *FALSE*, the model will generate the equations assuming constant relative volatilities (the user must estimate these volatilities). When *TRUE*, the

model generates the equations assuming the chemical potentials for a component are equal in all phases.

11.2 USING THE THERMODYNAMICS MODELS

There are several libraries of models which use the libraries we have just described. The first library to examine is stream_holdup.a4l. This library contains steady-state models for a stream and a holdup. The following gives the parameter list for a user to create an instance of a stream.

11.2.1 STREAMS AND HOLDUPS

```
MODEL stream(                                    84
   cd WILL_BE components_data;                   85
   pd WILL_BE phases_data;                       86
   equilibrated WILL_BE boolean;                 87
) REFINES sh_base;                               88
```

The model sh_base is a dummy model to tie all models into this library back to a common root model. The user need do nothing because of this refinement. What you should note is that all you need to do to create a stream is create a *components_data* model and a *phases_data* model. One supplies the boolean variable *equilibrated* as a variable that one can set interactively or in a method or a script when running the model to decide how to model equilibrium, as we have discussed above. A holdup is equally as easy to model.

11.2.2 FLASH UNITS AND VARIANTS THEREOF

From streams and holdups, we can move on to unit operation models. The library flash.a4l provide us with a flash model. The parameter list for the flash model is:

```
MODEL vapor_liquid_flash(
   Qin WILL_BE energy_rate;
   equilibrated WILL_BE boolean;
   feed WILL_BE stream;
   vapout WILL_BE stream;
   liqout WILL_BE stream;
) WHERE (
   feed, vapout, liqout WILL_NOT_BE_THE_SAME;
   feed.cd, vapout.cd, liqout.cd WILL_BE_THE_SAME;
   vapout.pd.phase_indicator == 'V';
```

```
   liqout.pd.phase_indicator == 'L';
   (feed.pd.phase_indicator IN ['V','L','VL','VLL']) ==
TRUE;
) REFINES flash_base;
```

Again we see that to create a *flash* unit, we need to create the variable *Qin* for the heat input to the unit, a boolean *equilibrated* and three streams, *feed*, *vapout* and *liqout*. The three streams must all be different streams. They must have the same components in them. The stream *vapout* must be a vapor stream and the stream *liqout* a liquid stream. The feed stream can be of any kind.

Hopefully with the above information, creating a flash unit should not now seem particularly difficult.

If you examine this library further, you will see it contains models which are variations of the flash unit for: *detailed_tray, tray, feed_tray, total_condenser* and *simple_reboiler.*

11.2.3 DISTILLATION COLUMNS

We provide two libraries that allow you to model distillation columns: *column.a4l* and *collocation.a4l*. The library *column.a4l* first models a tray stack and then a simple column using that model. A third model extracts the profiles for pressure, temperature, a parameter that indicates the deviation from constant molar overflow conditions, total vapor and liquid flows and component compositions against tray number. This information may then be used for plotting these profiles using the ASCEND plotting capability.

The library *collocation.a4l* provides collocation models for simple columns. With collocation models, one models composition profiles as smooth functions of tray number in a column section. Columns with a large number of trays are modeled with relatively small collocation models. Also the number of trays becomes a continuous variable, aiding in optimization studies where the number of trays in each section is to be computed.

11.2.4 DYNAMIC UNIT MODELS

ASCEND contains models for simulating the dynamic behavior of units. Their use is described in Chapter xxxx.

11.3 DISCUSSION

We have presented a description of the libraries that allow one to model the equations providing thermodynamic properties for multi-component, multi-phase mixtures when one assume equilibrium exists among co-existing phases. With this description, we hope that these models become much less difficult to use. We end this chapter by describing other libraries that build on the property estimation libraries, models for streams and holdups, for flash units and variations thereof, and for columns.