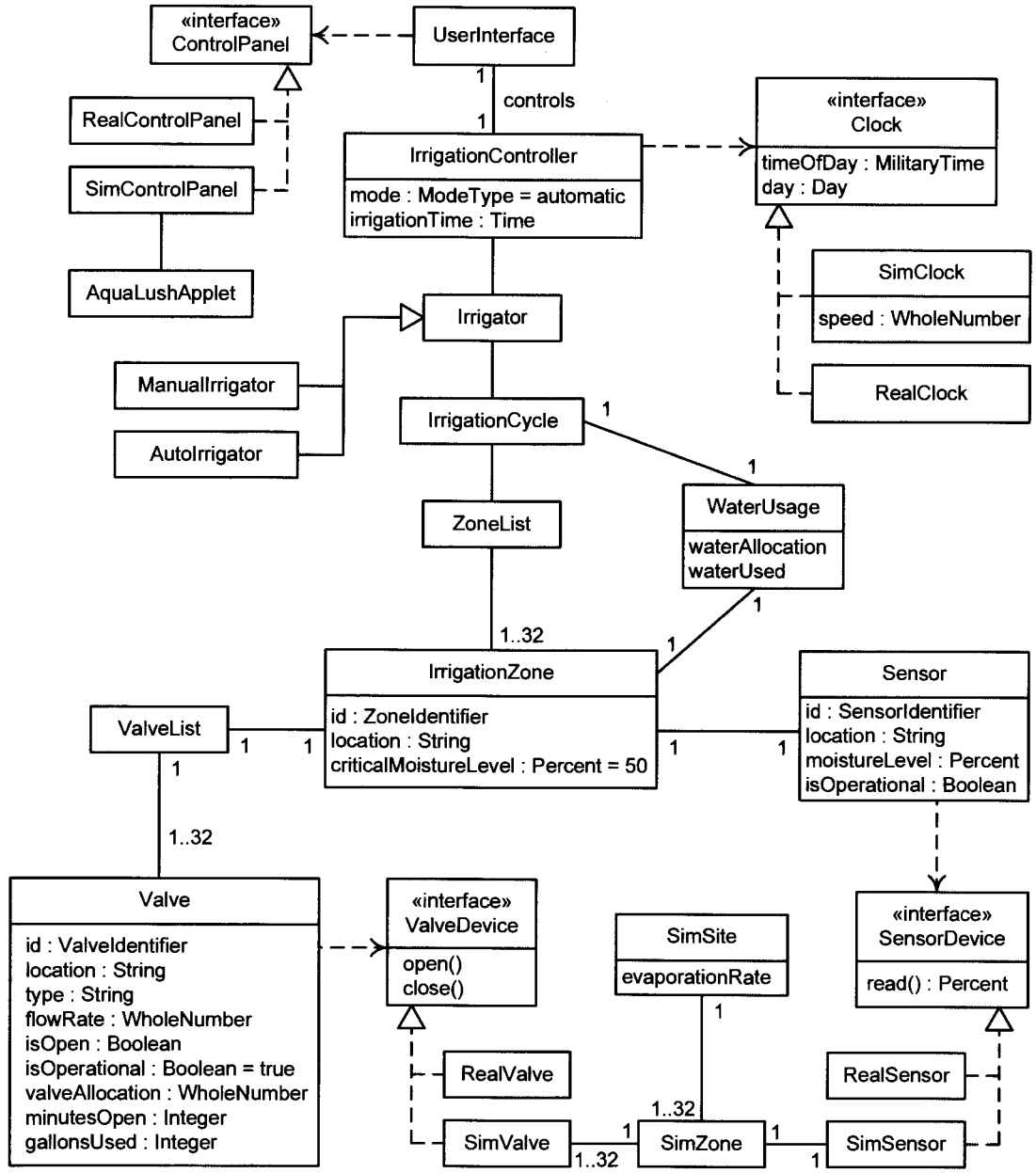

Figure 11-3-4 Draft AquaLush Design Class Diagram (Creational)

A Transformational Technique

An important feature of the object-oriented paradigm is that it provides solutions closely resembling problems. This has several advantages. First, it makes the connection between the problem and the solution easy to understand, which makes both development and maintenance easier. Second, important entities in the problem tend not to change very much over time, though their behaviors and characteristics may. If these stable

entities are reflected in the program, then program entities will also tend not to change over time; only their attributes and operations may change. This makes it easier to modify the program in response to new or changed requirements. Thus, it is advisable to have program entities that mimic problem entities.

A conceptual model describes the important concepts in the engineering design problem. If it is taken as a starting point for problem solution, then the resulting design will contain many entities reflecting the problem domain. Transformational techniques take advantage of this insight by starting with a conceptual class model and modifying it to make a draft design class model.

Our transformation technique uses the following heuristics to convert a conceptual class model into a design class model. These are illustrated by the AquaLush draft design model in Figure 11-3-5 on page 343, which is the result of transforming the AquaLush conceptual model in Appendix B.

Change actors to interface classes. Conceptual models contain classes representing actors. Actors are not part of the design solution, but interfaces to actors are. Hence, each actor class should be converted into an interface class for that actor. For example, the AquaLush conceptual model actors **User**, **Clock**, **Control Panel**, **Valve**, and **Sensor** are converted into **UserInterface**, **Clock**, **ControlPanel**, **ValveDevice**, and **SensorDevice** in the design class model.

Add actor domain classes. A program may also record data about actors. Any conceptual model actor class about which the program needs to record data should also be made into an application domain class with the appropriate attributes. AquaLush needs to keep track of **Valves** and **Sensors**, so these are included as domain classes in the model.

Add a startup class. Many programs need a place to begin, so a startup class may need to be added to the design class model. AquaLush needs to configure itself and restore settings from persistent store, so it certainly needs a startup class to handle this chore. The **Configurator** class plays this role.

Convert or add controllers and coordinators. Classes that control and coordinate program activities may already be present as conceptual classes—these are simply converted to design classes. Some controllers and coordinators may need to be added; some may become evident only when interactions are modeled. AquaLush needs irrigation controllers and a simulation controller, but these are already present in the conceptual model and are simply carried over into the design model.

Add classes for data types. Conceptual models often include types with a great deal of structure and behavior, and they must be implemented as data types in the program. Classes to implement data types should be added to the design class model. For example, AquaLush has two irrigation modes. These are modeled by an enumeration class in the design class model.

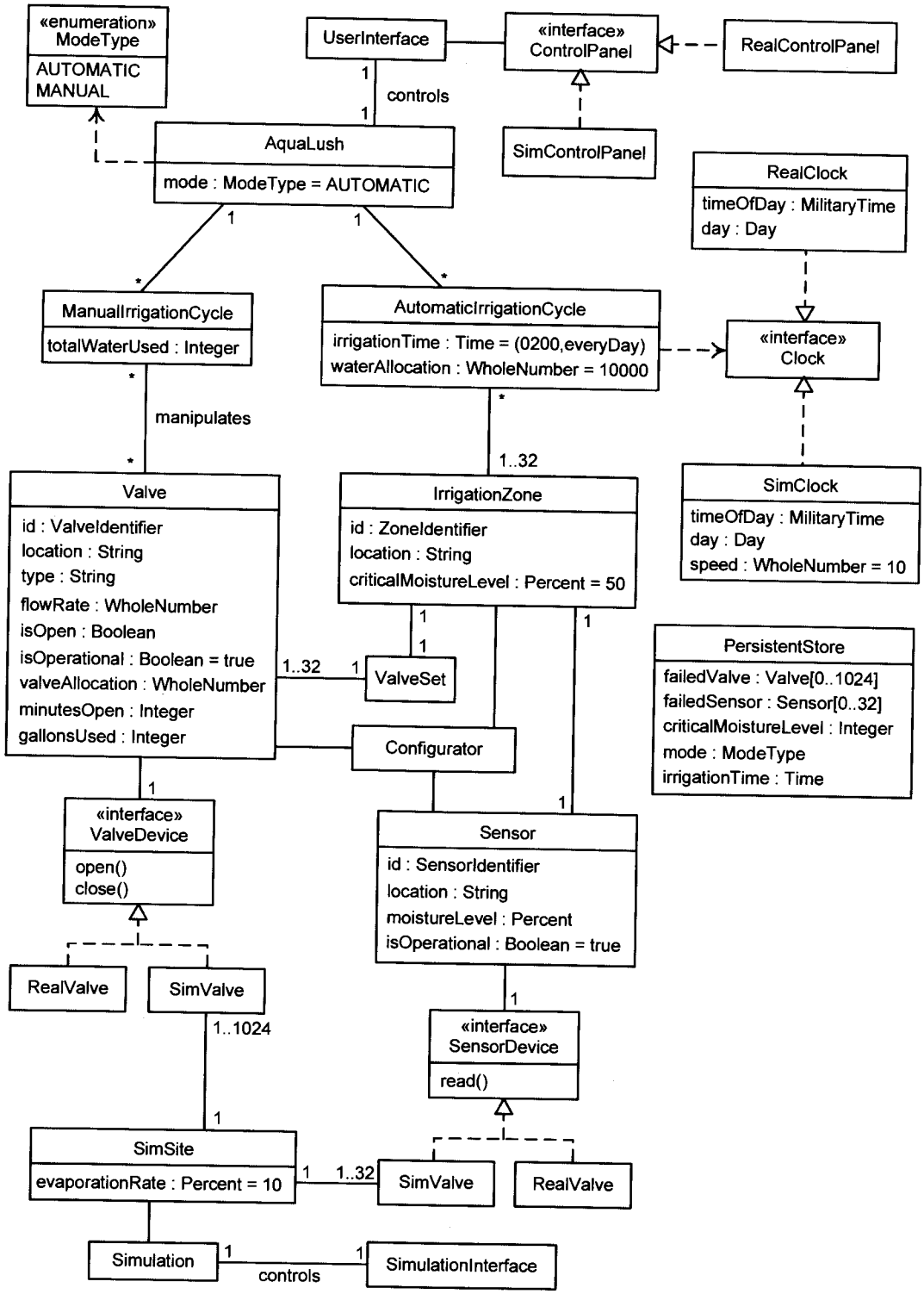

Figure 11-3-5 Draft AquaLush Design Class Diagram (Transformational)

Convert or add container classes. A **container class** has instances that hold a collection of objects. Containers may already be present in the conceptual model, or they may be added to the design class model. This is a somewhat low-level design detail, so often decisions about particular containers can be skipped in mid-level design. As an example, AquaLush irrigation zones have between 1 and 32 valves. The design class model has a ValveSet class to contain the valves for each irrigation zone.

Convert or add engineering design relationships. Generalization relationships, interfaces and interface realization relationships, dependencies, and so forth may already be present in the conceptual model or can be added as appropriate. Many of these will be added as the design progresses to dynamic modeling. In our example, the realization relationships between virtual device interfaces and the device drivers that realize them for simulated and actual devices are shown in the model.

Comparing Creational and Transformational Techniques

The AquaLush models produced using the generational and transformational techniques have much in common, but they have some differences as well. Many core classes, such as Sensor, Valve, IrrigationZone, and Clock, are identical. The classes for managing irrigation, however, are different. In fact, either of these design alternatives could have been produced by either a creational or a transformational technique: There is nothing about them that is a consequence of the way that they were created. There is no particular advantage to using one technique over the other in terms of results.

One technique does have an advantage in terms of effort, however. It is much easier to transform a conceptual model into a draft design class model than it is to generate one from scratch. Consequently, a transformational technique should be used if a conceptual model is available. If there is no conceptual model, then one can be created and then transformed into a design class model, or a generational technique can be used to create a design class model from scratch.

Heuristics Summary

Figures 11-3-6 and 11-3-7 summarize the class diagram generation heuristics discussed in this section.

- Change actors to interface classes.
- Add actor domain classes.
- Add a startup class.
- Convert or add controllers and coordinators.
- Add classes for data types.
- Convert or add container classes.
- Convert or add engineering design relationships.

Figure 11-3-6 Conceptual Model Transformation Heuristics

Generating Classes from Themes

- Look for entities in charge of program tasks.
- Look for actors.
- Look for things about which the program stores data.
- Look for structures and collections of objects.

Evaluating and Selecting Candidate Classes

- Discard classes with vague names or murky responsibilities.
- Rework candidate classes with overlapping responsibilities to divide their responsibilities cleanly.
- Discard classes that do something out of scope.

Evaluating and Improving the Class Diagram

- Check each class for important but overlooked attributes, operations, or associations.
- Combine common attributes and operations from similar classes into a common super-class.
- Apply design patterns where appropriate.

Figure 11-3-7 Theme-Based Decomposition Heuristics

Section Summary

- Drafting a design class model is a starting point for mid-level design.
- Various creational or transformational techniques can produce a draft design class model by either refining the architecture or working directly from the SRS and analysis models.
- A theme-based creational technique works by writing a design story, extracting its themes, using the themes to produce candidate classes with clear responsibilities, and drawing and improving a class diagram.
- A transformational technique works by applying rules to produce a design class model from a conceptual class model.

Review Quiz 11.3

1. Describe two creational and two transformational techniques for generating a mid-level design model.
2. What is a design story and what is it used for?
3. Explain what a container class is and list two examples.

11.4 Static Modeling Heuristics

Responsibilities

We have often mentioned responsibilities without talking much about what they are. A responsibility is an obligation; that is, something that one is bound to do.