Real-Time Software Design For Embedded Systems

HASSAN
GOMAA

5

Structural Modeling for Real-Time Embedded Systems with SysML and UML

◈

This chapter describes how *structural modeling* can be used as an integrated approach for system and software modeling of embedded systems consisting of both hardware and software components. The structural view of a system is a *static modeling* view, which does not change with time. *A static model* describes the static structure of the system being modeled, first the static structure of the total hardware/software system followed by the static structure of the software system.

Since a *class* is a software concept describing a software element, a more general term is needed to refer to a system element. SysML uses the concept of a *block* as a system structural element, which is a broader modeling concept than a class that can be used to refer to a hardware, software, or person structural element. In this chapter, the term *structural element* is used to refer to either a block or class.

The SysML *block definition diagram* notation is used to depict the static model of the total hardware/software system and the UML *class diagram* notation is used to depict the static model of the software system. SysML block definition diagrams and UML class diagrams were first introduced in Chapter 2. For system modeling, this chapter describes system-wide structural modeling concepts including blocks, attributes of blocks, and relationships between blocks. For software modeling, this chapter describes software structural modeling concepts including classes, attributes of classes and relationships between classes. Software design concepts such as class operations (methods) are deferred to software class design as described in Chapter 14.

The objective of the model-based approach described in this chapter is to clearly delineate between total system (i.e., hardware and software) modeling and strictly software modeling, with a well-defined transition between the two modeling activities. This chapter starts with a brief description of static modeling, in particular the relationships between structural elements (blocks or classes) in Section 5.1. Three types of relationship are described: *associations*, *composition* and *aggregation* relationships, and *generalization/specialization* relationships. After the introduction to static modeling, this chapter addresses the categorization of blocks and classes using stereotypes in Section 5.2, structural modeling of the problem domain with SysML in Section 5.3, structural modeling of the system context in Section 5.4, hardware/software boundary modeling in Section 5.5, structural modeling of the software system context in Section 5.6, and defining hardware/software interfaces in Section 5.7. Finally, system deployment modeling is described in Section 5.8.

5.1 Static Modeling Concepts

This section provides an overview of static modeling concepts, which are used in the structural modeling of embedded systems. A *static model* defines the structural elements of a system in terms of blocks in the total hardware/software system and classes in the software system, as well as the attributes of the structural elements and the relationships between them. The concepts of objects and classes, as well as class attributes and operations, are described in Chapter 3. This section describes the three main types of relationships between *structural elements* (whether *system blocks* or *software classes*): associations, whole/part relationships, and generalization/specialization relationships. The relationships described in this section apply equally to relationships between UML classes and to relationships between SysML blocks. More information on the static modeling notation is given in Chapter 2. The following subsections describe each type of relationship in turn.

5.1.1 Associations

An **association** is a static, structural relationship between two or more structural elements. The **multiplicity** of an association specifies how many instances of one structural element can relate to a single instance of another structural element. The multiplicity of an association may be:

- **One-to-one (1..1)**. In a one-to-one association between two structural elements, the association is one-to-one in both directions.
- **One-to-many (1..*)**. In a one-to-many association, two structural elements have a one-to-many association in one direction and a one-to-one association in the opposite direction.
- **Numerically specified (m..n)**. A numerically specified association is an association that refers to a specific range of numbers.
- **Optional (0..1)**. In an optional association, two structural elements have a zero-to-one association in one direction and a one-to-one association in the opposite direction. This means that there might not always be a link from an instance of one structural element to an instance of the other.
- **Many-to-many (*)**. In a many-to-many association, two structural elements have a one-to-many association in each direction.

An example of structural elements that depicts classes and their associations in a factory automation system is given in Figure 5.1. A workflow plan defines the steps for manufacturing a part of a given type; it contains several manufacturing operations, where each operation defines a single manufacturing step. Consequently, there is a one-to-many association between the `Workflow Plan` class and the `Manufacturing Operation` class. A work order defines the number of parts to be manufactured of a given part type. Thus the `Work Order` class has a one-to-many association with the `Part` class. Because a workflow plan defines how all parts of a given part type are manufactured, the `Workflow Plan` class also has a one-to-many association with the `Part` class. The attributes of these classes are also shown in Figure 5.1. For example, the `Workflow Plan` class has attributes for the type of part to be manufactured, the raw material type to be used for manufacturing a part, and the number of manufacturing steps required to produce a part.

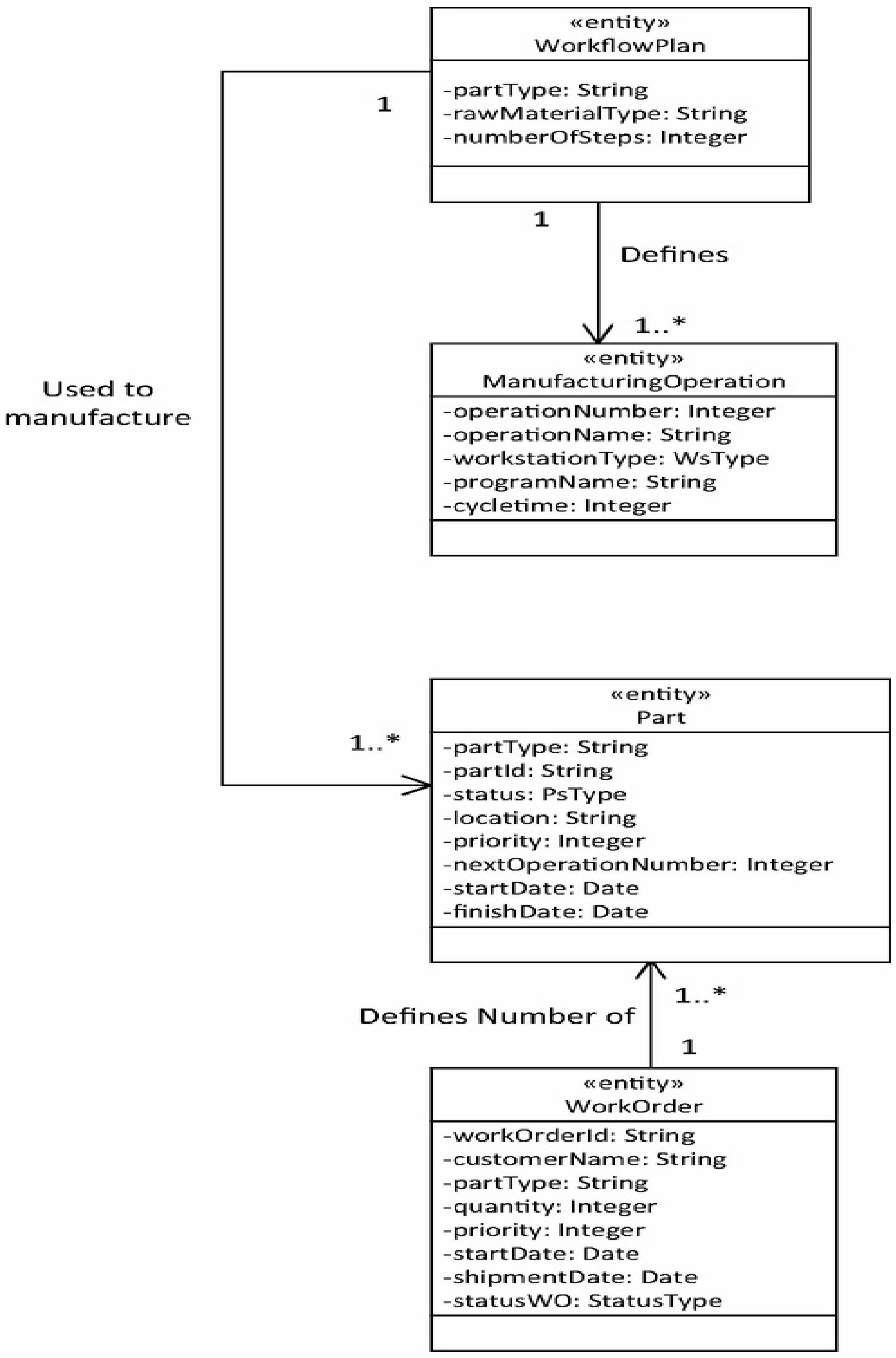
«entity»
WorkflowPlan
-partType: String
-rawMaterialType: String
-numberOfSteps: Integer
1
1
Defines
1..*
«entity»
ManufacturingOperation
-operationNumber: Integer
-operationName: String
-workstationType: WsType
-programName: String
-cycletime: Integer
Used to manufacture
1..*
«entity»
Part
-partType: String
-partId: String
-status: PsType
-location: String
-priority: Integer
-nextOperationNumber: Integer
-startDate: Date
-finishDate: Date
Defines Number of
1..*
1
«entity»
WorkOrder
-workOrderId: String
-customerName: String
-partType: String
-quantity: Integer
-priority: Integer
-startDate: Date
-shipmentDate: Date
-statusWO: StatusType

Figure 5.1. Example of classes, attributes, and associations on a class diagram.

5.1.2 Composition and Aggregation Hierarchies

Composition and aggregation are special forms of relationships in which structural elements (blocks or classes) are bound by a **whole/part** relationship. Both composition and aggregation hierarchies address a structural element that is made up of other structural elements.

A composition is a stronger form of whole/part relationship than an aggregation, and an aggregation is stronger than an association. In particular, the **composition** relationship demonstrates a stronger relationship between the parts and the whole than does the aggregation relationship. A composition is also a relationship among instances. Thus the part objects are created, live, and die together with the whole. The part object can belong to only one whole.

An example of a composition hierarchy is the block `Microwave Oven Embedded System`, which represents the whole and is composed of several part blocks: `Door Sensor`, `Heating Element`, `Keypad`, `Display`, `Weight Sensor`, `Beeper`,`Lamp`,`Turntable`, and `Timer`. There is a one-to-one association between the `Microwave Oven Embedded System` composite block and each of the part blocks, as shown in Figure 5.2.

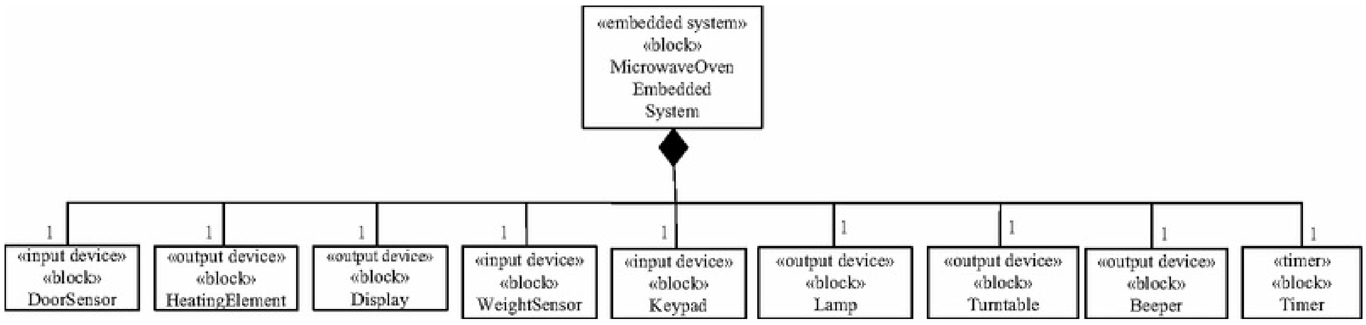

Figure 5.2. Example of a composition hierarchy.

The **aggregation** hierarchy is a weaker form of whole/part relationship. In an aggregation, part instances can be added to and removed from the aggregate whole. For this reason, aggregations are likely to be used to model conceptual structural elements rather than physical ones. In addition, a part may belong to more than one aggregation.

An example of an aggregation hierarchy is the `Automated Storage & Retrieval System (ASRS)`, which consists of one-to-many relationships with `ASRS Bin`, `ASRS Stand`, and `Forklift Truck` (see Figure 5.3). An ASRS consists of ASRS bins (where parts are stored), ASRS stands (where parts are placed after retrieval from an

ASRS bin or prior to storage in an ASRS bin), and forklift trucks (which move parts from the stands to the bins for storage and vice versa for retrieval). The reason that the ASRS is modeled as an aggregation is that it could be expanded to add more bins, stands, and trucks after it has been created. The attributes for the three part classes are also depicted in Figure 5.3. For example, `ASRS Bin` has attributes for the bin #, the ID of the part located in the bin, and the status of the bin (occupied or empty).

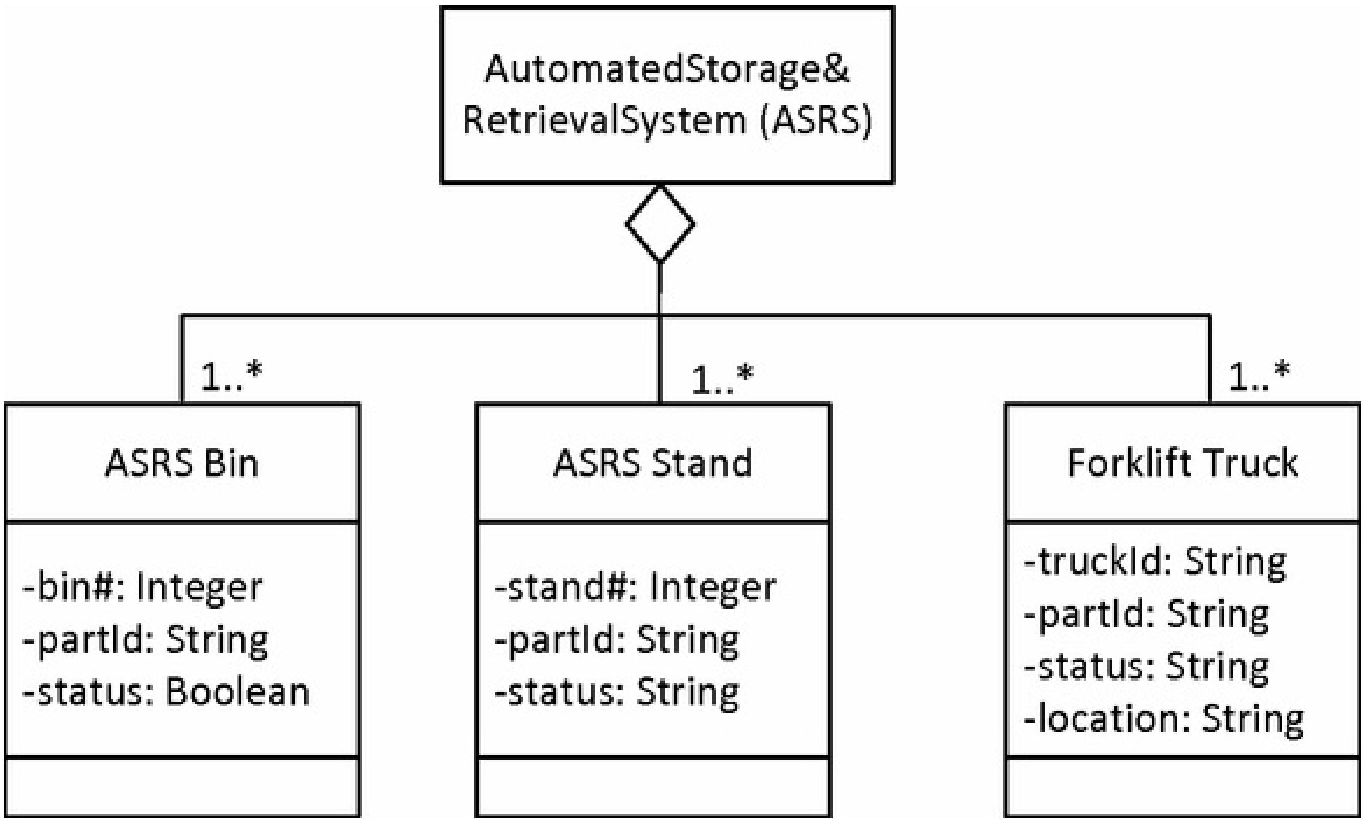

Figure 5.3. Example of an aggregation hierarchy.

5.1.3 Inheritance and Generalization/Specialization

Inheritance is a useful abstraction mechanism in structural modeling and design. Inheritance permits modeling of structural elements that are similar in some but not all respects, thus having some common properties but other unique properties that distinguish them. Inheritance is a classification mechanism that has been widely used in other fields. An example is the taxonomy of the animal kingdom, in which animals are classified as mammals, fish, reptiles, and so on. Cats and dogs have common properties that are generalized into the properties of mammals. However, they also have unique properties: A dog barks and a cat mews.

The following description is in terms of software classes, but it can also be applied to system blocks. **Inheritance** is a mechanism for sharing properties between classes. A child class inherits the properties (e.g., encapsulated data) of a parent class. It can then modify the structure (i.e., attributes) of its parent class by adding new attributes. The parent class is referred to as a **superclass** or *base class*. The child class is referred to as a **subclass** or *derived class*. The adaptation of a parent class to form a child class is referred to as *specialization*. Child classes may be further specialized, allowing the creation of class hierarchies, also referred to as **generalization/specialization** hierarchies.

Consider an example from a factory automation system given in Figure 5.4. There are three types of factory workstations – receiving workstations, line workstations, and shipping workstations – so they are modeled as a generalization/specialization hierarchy; that is, the `Factory Workstation` class is specialized into three subclasses: `Receiving Workstation`, `Shipping Workstation`, and `Line Workstation`. All factory workstations have attributes for `workstation name`, `workstation ID`, and `location`, which are therefore attributes of the superclass and are inherited by the subclasses. Since factory workstations are physically laid out in an assembly line, the `Receiving Workstation` class has a `next workstation ID`, the `Shipping Workstation` has a `previous workstation ID`, while a `Line Workstation` has both a `previous workstation ID` and a `next workstation ID`. Because of these differences, previous and next workstation IDs are attributes of the subclasses, as shown in Figure 5.4.

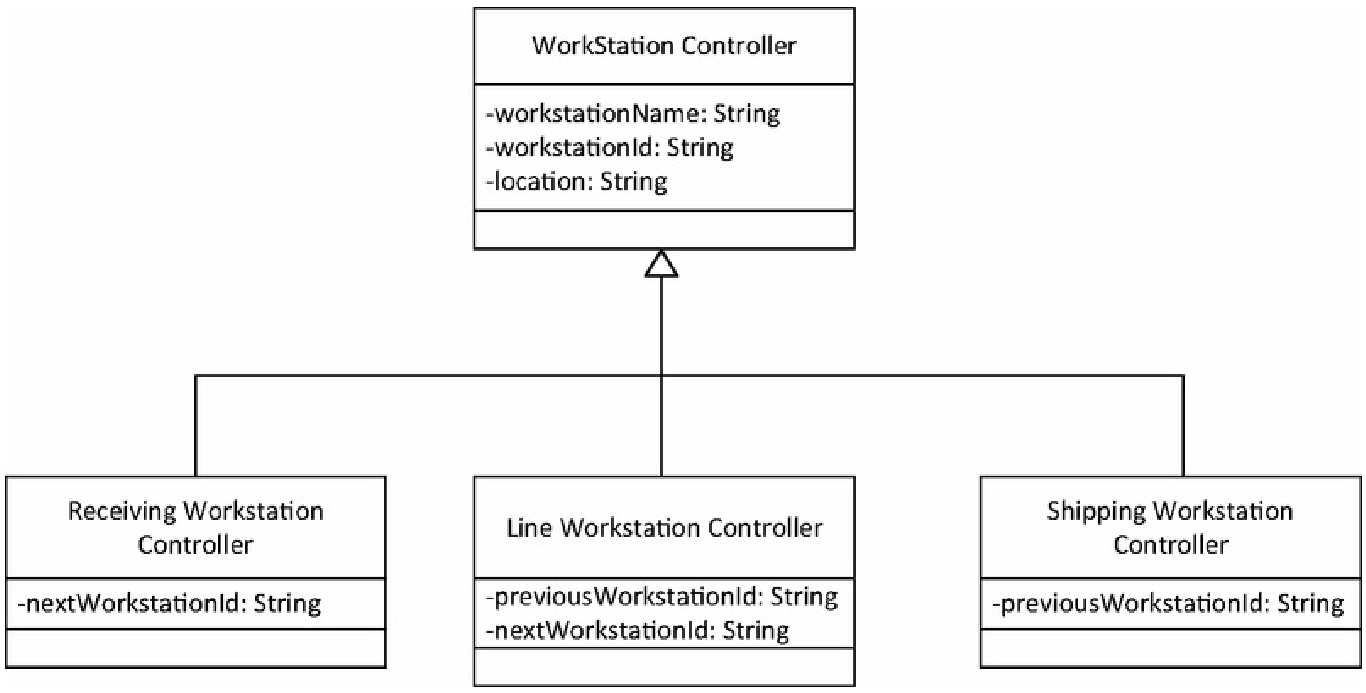
WorkStation Controller
-workstationName: String
-workstationId: String
-location: String
Receiving Workstation Controller
-nextWorkstationId: String
Line Workstation Controller
-previousWorkstationId: String
-nextWorkstationId: String
Shipping Workstation Controller
-previousWorkstationId: String

Figure 5.4. Example of a generalization/specialization hierarchy.

5.2 Categorization of Blocks and Classes using Stereotypes

This section describes how blocks and classes can be categorized (i.e., grouped together) using a classification approach. The dictionary definition of *category* is “a specifically defined division in a system of classification.” Whereas classification based on inheritance is an objective of object-oriented modeling, it is essentially tactical in nature. Thus, classifying the `Factory Workstation` class into a `Receiving Workstation`, `Shipping Workstation`, and `Line Workstation`is a good idea because `Receiving Workstation`, `Shipping Workstation`, and `Line Workstation` have some properties (e.g., attributes) in common and others that differ. Categorization, however, is a strategic classification – a decision to organize classes into certain groups because most software systems have these kinds of classes and categorizing classes in this way helps to better understand the system being developed.

In UML and SysML, stereotypes are used to distinguish among the various kinds of modeling elements. A **stereotype** is a subclass of an existing modeling element (for example an application or external class), which is used to represent a usage distinction (for example the kind of application or external class). In the UML notation, a stereotype is enclosed by guillemets, like this: «input device».

Examples shown in Figure 5.5 from the microwave oven system are the input devices `Door Sensor and Weight Sensor`, the output devices `Heating Element` and `Lamp`, and the timer `Oven Timer`.

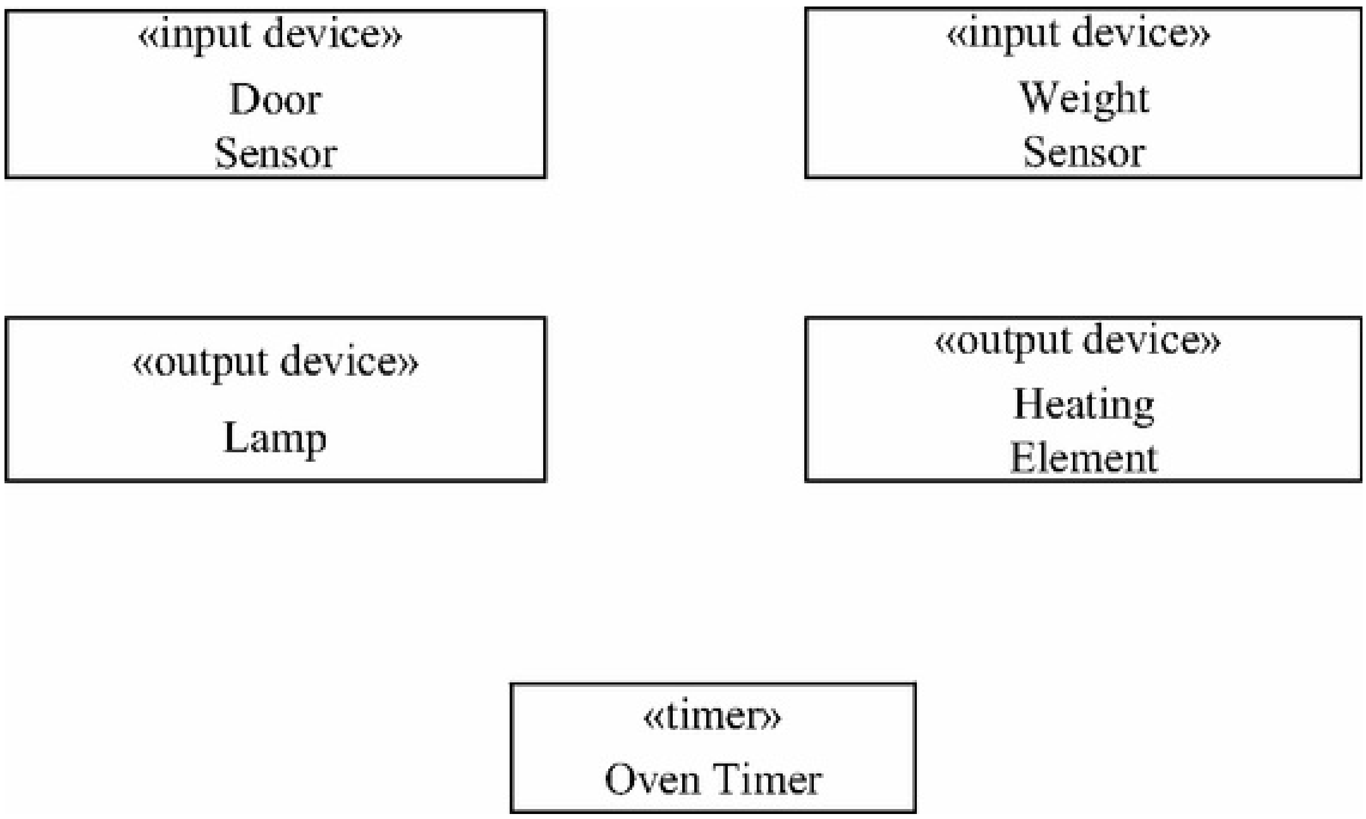
«input device» Door Sensor
«input device» Weight Sensor
«output device» Lamp
«output device» Heating Element
«timer» Oven Timer

Figure 5.5. Example of UML modeling elements and their stereotypes.

5.3 Structural Modeling of the Problem Domain with SysML

Structural modeling of the problem domain for real-time embedded systems refers to modeling the external entities that interface to the embedded system to be developed, as well as the hardware and software structural elements of the embedded system. In this structural modeling, the embedded system refers to the total hardware/software system, consisting of hardware elements, such as sensors and actuators, and software elements. The software system refers to the software elements, in particular the software components that compose the software system to be developed.

5.3.1 Modeling Real-World Entities in the Problem Domain

With structural modeling of the problem domain for real-time embedded systems, the designer uses SysML block definition diagrams (see Section 2.12) to depict real-world structural elements (such as hardware elements, software elements, or people) as blocks and defines the relationships among these blocks. A block definition diagram is equivalent to a class diagram in which the classes have been stereotyped as blocks, thereby allowing a block definition diagram to depict the same modeling relationships as a class diagram.

In structural modeling of the problem domain, the initial emphasis is on modeling real-world entities to create a conceptual static model, which includes relevant systems, users, physical entities and information entities. Relevant real-world entities in the problem domain of embedded systems include:

1. **Physical entity**. A physical entity is an entity in the problem domain that has physical characteristics – that is, it can be seen or touched. Such entities include physical devices, which are often part of the problem domain in embedded applications. For example, in the railroad crossing system, the train is a physical entity that must be detected by the system. Other relevant physical entities controlled by the system are the railroad crossing barrier, the warning flashing lights, and the audio warning alarm.

2. **Human user**. A human user of the system interacts with the system, providing inputs to the system and receiving outputs from the system. For example, the microwave user is a human user.

3. **Human observer**. A human observer views the outputs of the system but does not interact directly with the system, that is, does not provide any inputs to the system. An example of a human observer is a vehicle driver or pedestrian who is alerted of the imminent train arrival by the closing of the barrier, the flashing lights, and the audio alarm.

4. **Relevant system**. A relevant system is the system to be developed or any other system that interfaces to it. Relevant systems can be *embedded systems*, *information systems*, or *external systems*.

5. **Information entity**. An information entity is a conceptual data-intensive entity that is often persistent – that is, long-living. Information entities are particularly

prevalent in information systems (e.g., in a banking application, examples include accounts and transactions) but may also be needed by some real-time systems (for example to store status information or system configuration information). Information entities are modeled as UML classes as described in Section 5.3.3.

An example of a conceptual structural model of the problem domain is shown in the block definition diagram for the Railroad Crossing Embedded System in Figure 5.6. From a total system perspective, the problem domain for Railroad Crossing Embedded System consists of the following blocks:

- `Railroad Crossing Embedded System`, which is the *embedded system* to be developed;
- `Train`, which is a *physical entity* detected by the system;
- `Barrier`, which is a *physical entity* controlled by the system and which consists of a barrier actuator and a barrier sensor;
- `Warning Alarm`, which consists of Warning Lights and Warning Audio and which is a *physical entity* controlled by the system;
- `Observer` (who waits at the railroad crossing), which is an *observer* of the system;
- `Rail Operations Service`, which is an *external system* that is notified of the status of the railroad crossing.

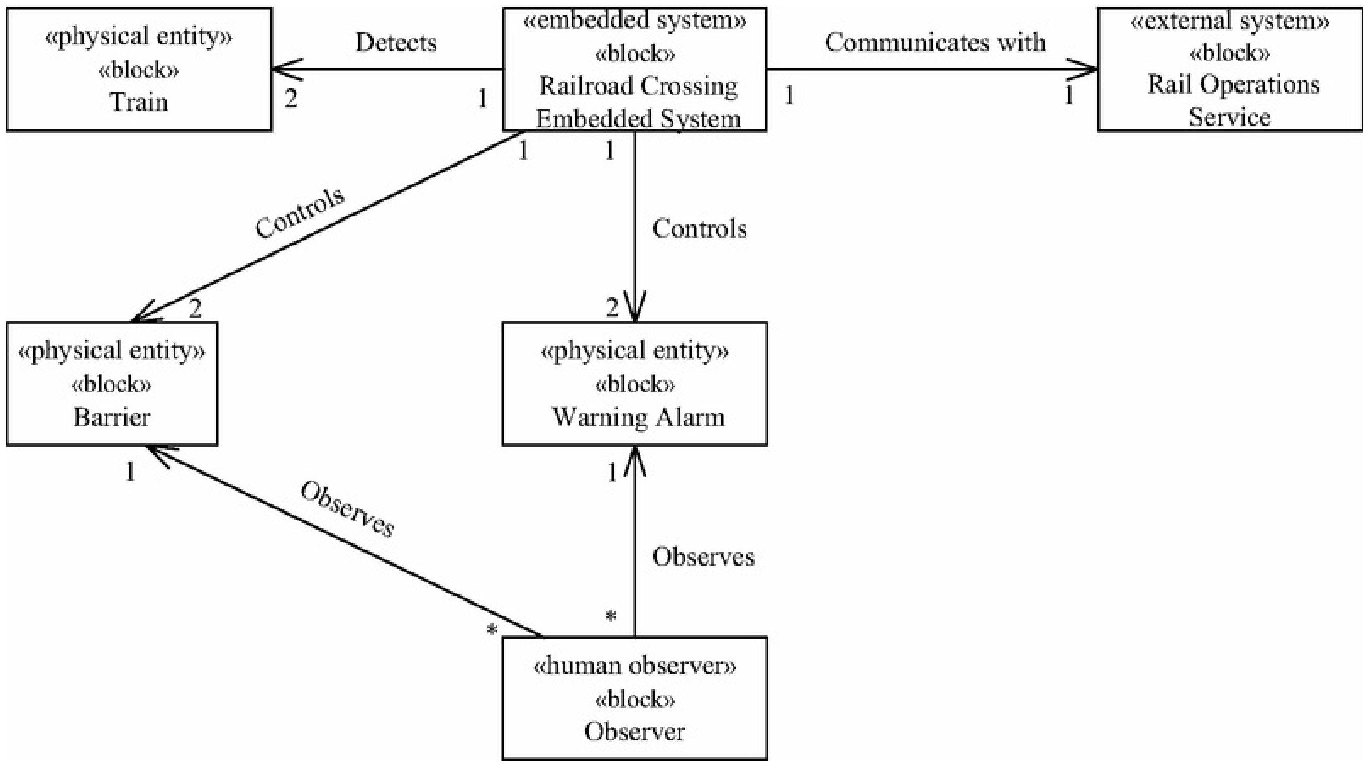
«physical entity» «block» Train
Detects
2
1
«embedded system» «block» Railroad Crossing Embedded System
Communicates with
1
1
«external system» «block» Rail Operations Service
1
1
Controls
Controls
2
2
«physical entity» «block» Barrier
«physical entity» «block» Warning Alarm
1
1
Observes
Observes
*
*
«human observer» «block» Observer

Figure 5.6. Example of conceptual structural model of problem domain.

5.3.2 Modeling the Embedded System

In embedded systems, in which there are several physical devices such as sensors and actuators, block definition diagrams can help with modeling these real-world devices. In the microwave oven system, for example, it is useful to model real-world devices (such as the door, heating element, weight sensor, turntable, beeper, display, keypad, lamp, and timer), their associations, and the multiplicity of the associations. Composite blocks are often used to show how a real-world composite modeling element, such as the Microwave Oven Embedded System composite block (see Figure 5.2), is composed of other blocks. Individual blocks are categorized as input devices, output devices, timers, and systems and are depicted on block definition diagrams using stereotypes. An example of a structural model is for the Microwave Oven System, which is an embedded system described in the case study in Chapter 19.

5.3.3 Modeling Information Entities as Entity Classes

Information Entities are modeled as entity classes, which are depicted as UML classes with the stereotype «entity». Entity classes are conceptual data-intensive classes. Some entity classes store persistent (i.e., long-lasting) data that, during execution, is typically accessed by several objects. Entity classes are particularly prevalent in information systems; however, many real-time and distribution applications have significant data-intensive functionality.

During static modeling of the problem domain, the emphasis is on determining the entity classes that are defined in the problem, their attributes, and their relationships. For example, in a Factory Automation System, there are parts, workflow plans, manufacturing operations, and work orders all mentioned in the problem description. Each of these real-world conceptual entities is modeled as an entity class and depicted with the stereotype «entity», as depicted in Figure 5.1. The attributes of each entity class are determined and the relationships among entity classes are defined, as described in Section 5.1.

5.4 Structural Modeling of the System Context

It is very important to understand the *system context*, that is the scope of a computer system – in particular, what is to be included inside the system and what is to be excluded from the system. Context modeling explicitly identifies what is inside the system and what is outside. Context modeling can be done at the total system (hardware and software) level or at the software system (software only) level. The system context is determined after modeling and understanding the problem domain, as described in Section 5.3.

A **system context diagram** is a block definition diagram that explicitly depicts the boundary between the system (hardware and software), which is modeled as one block, and the external environment. By contrast, a **software system context diagram** explicitly shows the boundary between the software system, also modeled as one block, and the external environment, which now includes the hardware.

When developing the system context (which is depicted on a block definition diagram) it is necessary to consider the context of the total hardware/software system before considering the context of the software system. In considering the total hardware/software system, only users and external system modeling elements are outside the system, while hardware and software modeling elements are internal to the system. Thus, I/O devices are part of the hardware of the system and are therefore part of the total hardware/software system.

5.4.1 Modeling External Entities of the Embedded System

When modeling an embedded hardware/software system, many of the real-world entities described in Section 5.3.1 are external entities that interface to the embedded system. Possible external entities are:

1. **External physical entity**. A physical entity is an external entity that the system has to detect and/or control. For example, in the railroad crossing system, the train is an external physical entity that has to be detected by the system. Other external physical entities, which are controlled by the system, are the railroad crossing barrier, the warning flashing lights, and the audio warning alarm.Some external physical entities, such as smart devices, might provide input to or receive output from the system.

2. **External system**. An external system is a separate system that interfaces to and communicates with the system under development. An external system might be an existing system that was previously developed or a new system that is to be developed by a different organization. An external system typically sends input messages to the system under development and/or receives output messages from the system.

3. **External user**. An external user is a human user of the system who interacts with the system, providing inputs to the system and receiving outputs from the system. For example, the microwave user is an external user.

4. **External observer**. An external observer is a human being who views the outputs of the system but does not interact directly with the system, that is, does not provide any inputs to the system. An example of an external observer is a vehicle driver or pedestrian who is alerted of the imminent train arrival by the closing of the barrier, the flashing lights, and the audio alarm.

Using the SysML notation, the system context is depicted showing the hardware/software system as an aggregate block with the stereotype «embedded system». The external environment is depicted in terms of external entities, depicted as blocks, to which the system has to interface. Stereotypes are used to differentiate between the different kinds of external blocks. For the system context diagram, an external block could be an «external system», «external physical entity», «external user», or an «external observer».

5.4.2 Modeling Associations on the System Context Diagram

The associations between the embedded system block and the external blocks are depicted on the system context diagram, showing in particular the multiplicity of the associations between the external blocks and the embedded system. These can be *one-to-one* or *one-to-many* associations. In addition, each association is given a standard name, which describes what the association is between the embedded system and the external block. The standard association names on system context block diagrams are *Inputs to*, *Outputs to*, *Communicates with*, *Interacts with*, *Detects*, *Controls*, and *Observes*. Note that in some cases, there is more than one standard association name between an external block and the embedded system, if different associations between them are possible. These associations are used as follows:

«embedded system» *Outputs to* «external user»
«external physical entity» *Inputs to* «embedded system»
«embedded system» *Detects* «external physical entity»
«embedded system» *Controls* «external physical entity»
«external observer» *Observes* «embedded system»
«external user» *Interacts with* «embedded system»
«external system» *Communicates with* «embedded system»

Examples of associations on system context block diagrams are as follows:

Factory Automation System *Outputs to* Operator
Smart Device *Inputs to* Factory Automation System
Railroad Crossing System *Detects* Train
Railroad Crossing System *Controls* Barrier
Observer *Observes* Barrier
User *Interacts with* Microwave Oven System
Railroad Crossing System *Communicates with* Rail Operations System

5.4.3 Example of System Context Diagrams

As an example of a system context diagram, consider the Railroad Crossing Embedded System, which is depicted on the block definition diagram in Figure 5.7, which is derived from Figure 5.6. Whereas the conceptual structural model in Figure 5.6 is a model of the problem domain, the Figure 5.7 focuses on the boundary of the system to be developed, The Railroad Crossing Embedded System is categorized as an «embedded system» «block». From a total system perspective, the system interfaces to four external blocks:

- `Train`, which is an *external physical entity* detected by the system;
- `Barrier`, which is an *external physical entity* controlled by the system and which consists of a barrier actuator (to raise or lower the barrier) and barrier sensor (to detect that the barrier has been raised or lowered;
- `Warning Alarm`, which consists of `Warning Lights` and `Warning Audio` and which is an *external physical entity* controlled by the system;
- `Rail Operations Service`, which is an *external system* that is notified of the status of the railroad crossing.

Note that `Observer` (vehicle driver, cyclist, or pedestrian who stops at the railroad crossing) is an *external observer* of the system.

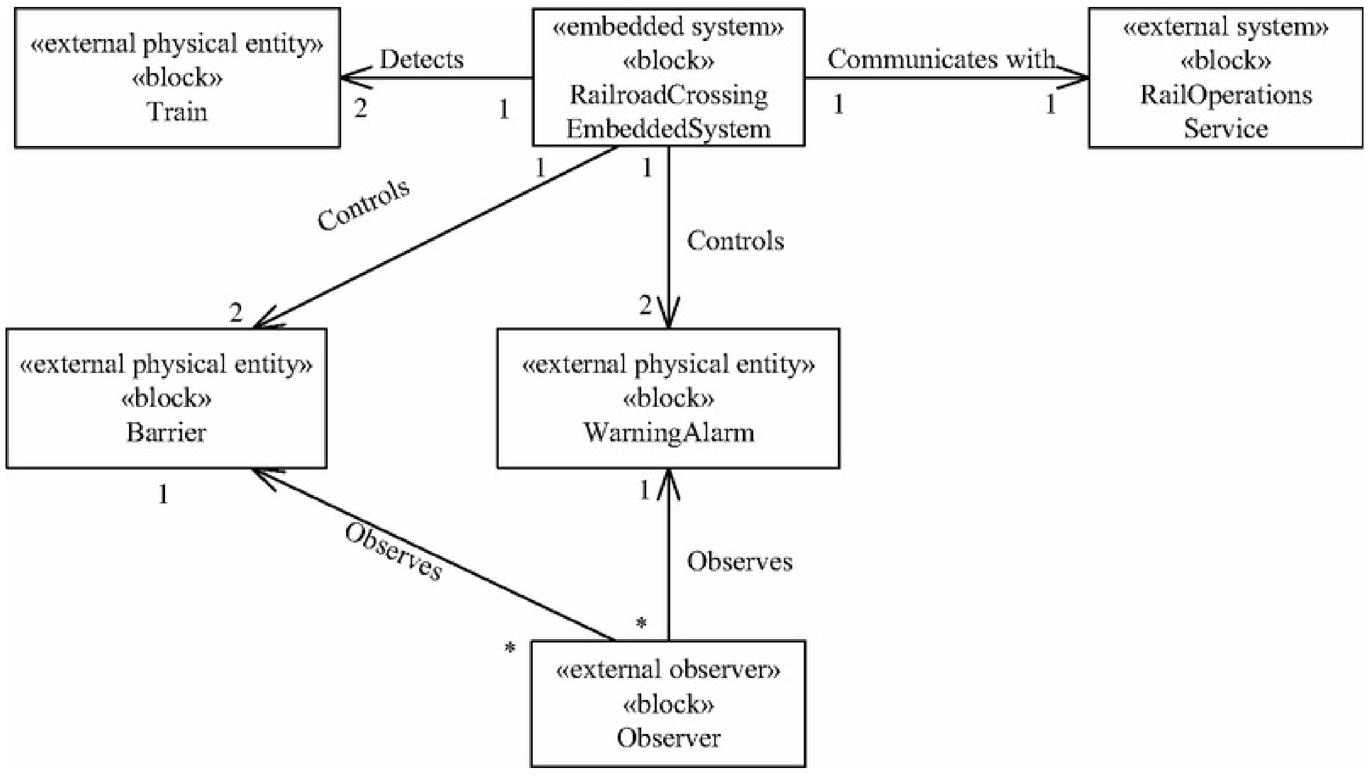
«external physical entity» «block» Train
Detects
2
1
«embedded system» «block» RailroadCrossing EmbeddedSystem
Communicates with
1
1
«external system» «block» RailOperations Service
1
1
Controls
Controls
2
2
«external physical entity» «block» Barrier
«external physical entity» «block» WarningAlarm
1
1
Observes
Observes
*
*
«external observer» «block» Observer

Figure 5.7 System context diagram for Railroad Crossing Embedded System.

5.5 Hardware/Software Boundary Modeling

To determine the boundary between the hardware and software blocks in preparation for modeling the software system context diagram, the modeler starts with the system context diagram and then determines the decomposition into hardware and software blocks.

From a software engineering perspective, some external blocks are modeled in the same way as in the systems engineering perspective, while others are modeled differently. In the former category are external system blocks and external users who interact with the system using standard I/O devices; these external blocks are depicted on the software system context diagram in the same way as on the system context diagram.

External blocks that are modeled differently from a software engineering perspective are external physical entity blocks that often do not physically connect to a system, and therefore need sensors or actuators to make the physical connection. As described in Section 5.4.2, the association between the embedded system and such a physical entity is *detects* and/or *controls*. Detection of physical entities is done by means of sensors while control of physical entities is done by means of actuators. Consider the external physical entities in the Railroad Crossing Embedded System. The arrival of a train is detected by an arrival sensor and the departure is detected by a departure sensor.

5.6 Structural Modeling of the Software System Context

As described in Section 5.4, the system context diagram depicts the systems and users that are external to the total hardware/software system, which is modeled as one composite block. The hardware blocks (such as sensors and actuators) and software blocks are internal to the system and are therefore not depicted on the system context diagram. Together with the hardware/software boundary modeling described in the previous section, this is the starting point for the software context modeling.

A **software system context diagram** is a block definition diagram that explicitly depicts the boundary between the software system, which is modeled as one block with the stereotype «software system», and the external environment. The software system context diagram is determined by analyzing the external blocks that connect to the software system. In particular, physical hardware devices (such as sensors and actuators) are external to the software system.

The software system is depicted on the software system context diagram as an aggregate block with the stereotypes «software system» «block», and the external environment is depicted as external blocks to which the software system has to interface.

5.6.1 Modeling External Entities of the Software System

For a real-time embedded system, it is desirable to identify low-level external blocks that correspond to all the external elements that the system has to interface to and communicate with, including physical I/O devices, external timers, external systems, and external users. External blocks are categorized by stereotype, as described in Section 5.7. Figure 5.8 depicts the classification of external blocks using inheritance, in which stereotypes are used to distinguish among the different kinds of external blocks. Thus, an external block is classified as an «external user» block, an «external device» block, an «external system» block, or an «external timer» block. Only external users and external systems can be external to the total system. Hardware devices and timers are part of the total (hardware and software) system but are external to the software system. Thus Figure 5.8 categorizes external blocks from the software system's perspective.

An external device block is classified further as follows:

- **External input device**. A device that only provides input to the system – for example, a sensor;
- **External output device**. A device that only receives output from the system – for example, an actuator;
- **External input/output device**. A device that both provides input to the system and receives output from the system – for example, a card reader for an automated teller machine.

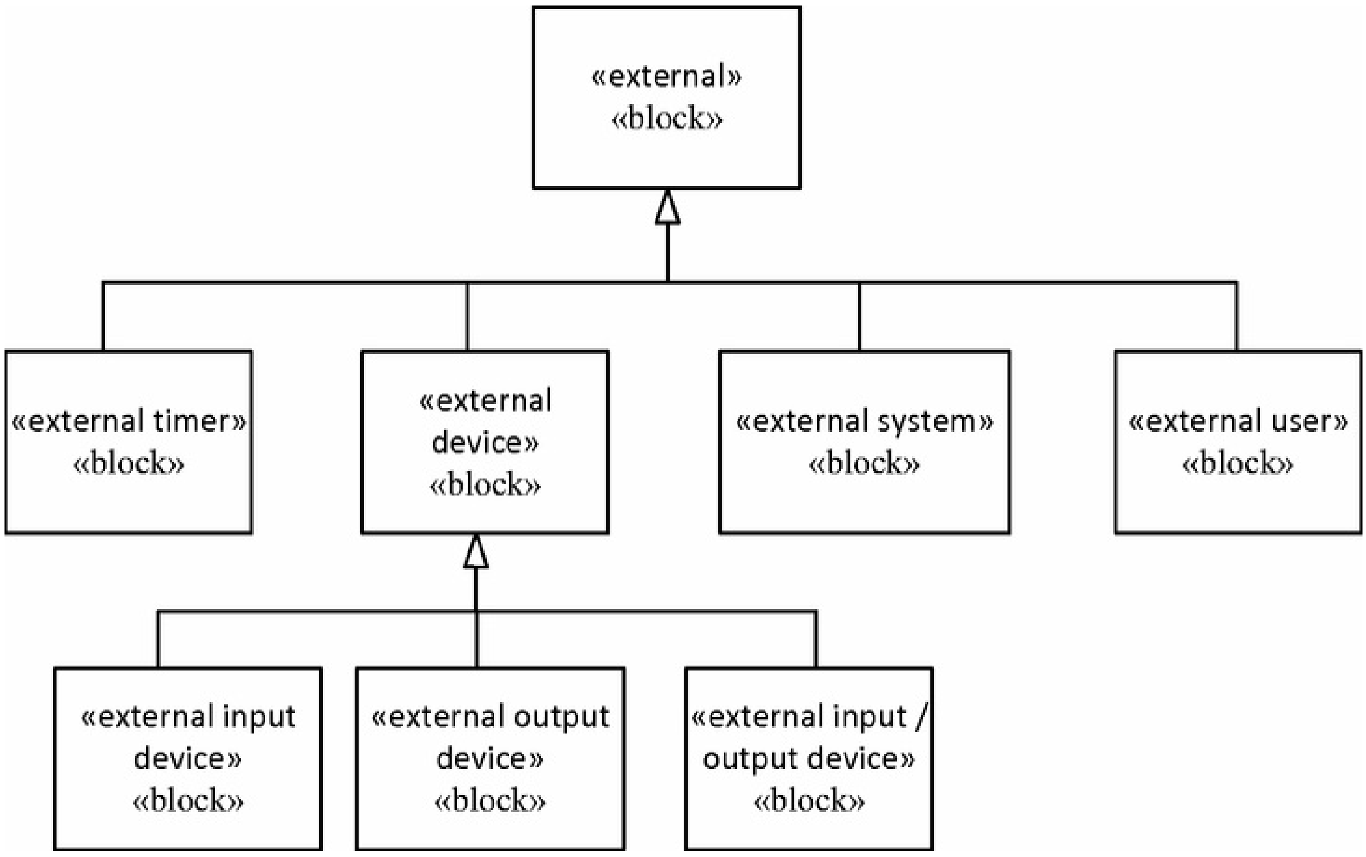

Figure 5.8. Classification of external blocks by stereotype.

These external blocks are depicted with the stereotypes «external input device», «external output device», and «external input/output device». Examples are the `Door Sensor` external input device and the `Heating Element` external output device in the microwave oven system (see Figure 5.9). Other examples are the Arrival Sensor external input device and the Motor external output device in the Train Control System.

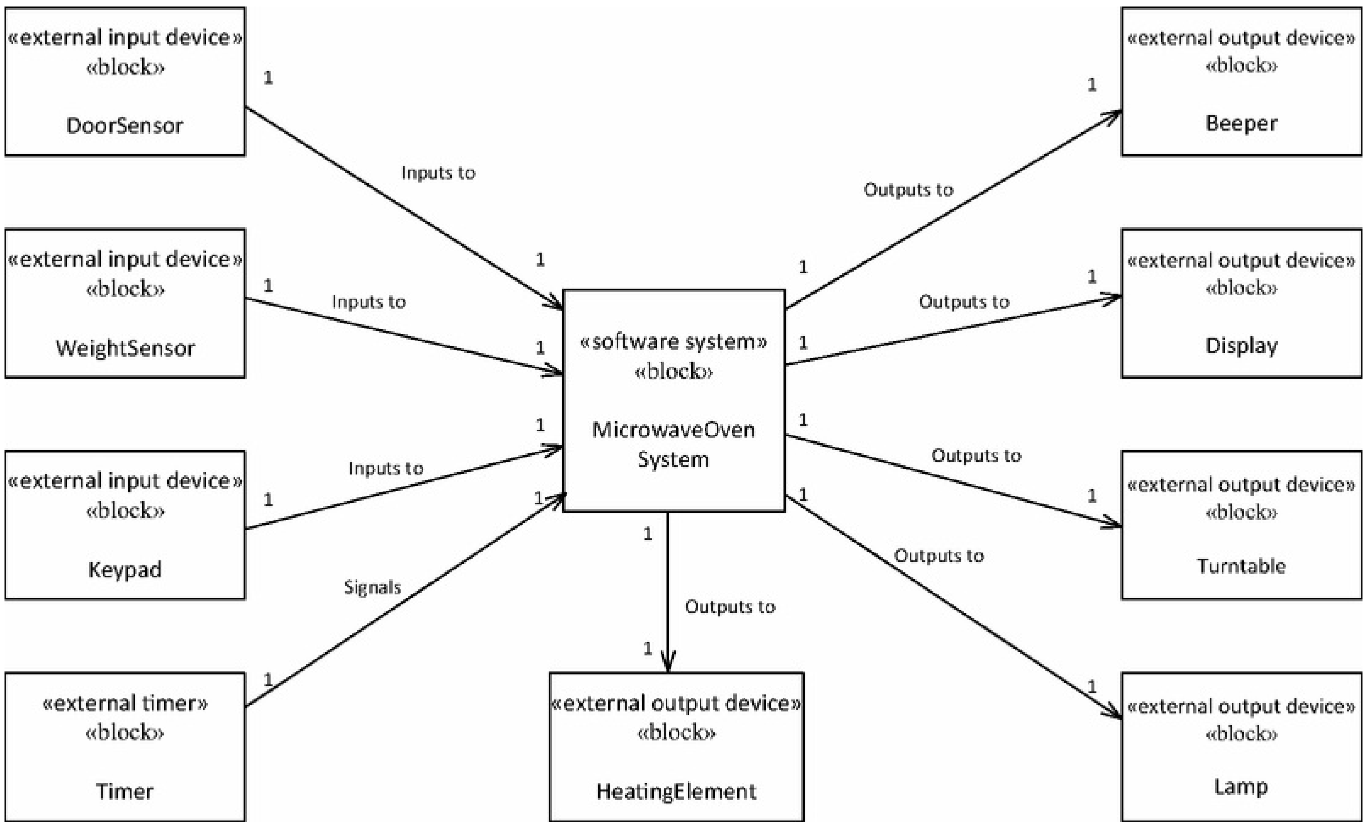
«external input device» «block» DoorSensor
«external input device» «block» WeightSensor
«external input device» «block» Keypad
«external timer» «block» Timer
«software system» «block» MicrowaveOven System
«external output device» «block» Beeper
«external output device» «block» Display
«external output device» «block» Turntable
«external output device» «block» Lamp
«external output device» «block» HeatingElement
Inputs to
Inputs to
Inputs to
Signals
Outputs to
Outputs to
Outputs to
Outputs to
Outputs to
1

Figure 5.9. Microwave Oven System software system context diagram

A human user often interacts with the system by means of standard I/O devices such as a keyboard/display and mouse. The characteristics of these standard I/O devices are of no interest because they are handled by the operating system. The interface to the user is of much greater interest in terms of what information is being output to the user and what information is being input from the user. For this reason, an external user interacting with the system via standard I/O devices is depicted as an «external user». An example is the `Factory Operator` in the factory automation system.

A general guideline is that a human user should be represented as an external user block only if the user interacts with the system via standard I/O devices. However, if the user interacts with the system via application-specific I/O devices, these I/O devices should be represented as external I/O device blocks.

An «external timer» block is used if the application needs to keep track of time and/or if it needs external timer events to initiate certain actions in the system. External timer blocks are frequently needed in real-time embedded systems. An example from the Microwave Oven System is the external `Timer`. It is needed because the system needs to keep track of elapsed time to determine the cooking time for food placed in the oven and count down the remaining cooking time, which it displays to the user. When the time remaining reaches zero, the system needs to stop cooking. In the Train Control System, time is needed to compute the speed of the train. Sometimes the need for periodic activities only becomes apparent during design.

An «external system» block is needed when the system interfaces to other systems, to either send data or receive data. Thus, in the Factory Automation System, the system interfaces to two external systems: the `Pick & Place Robot` and the `Assembly Robot`.

5.6.2 Modeling Associations on the Software System Context Diagram

The associations between the software system aggregate block and the external blocks are depicted on the software system context diagram, showing in particular the multiplicity of the associations and the name of the association. The standard association names on software system context diagrams are *Inputs to*, *Outputs to*, *Communicates with*, *Interacts with*, and *Signals*. These associations are used as follows:

```
«external input device» Inputs to «software system»
«software system» Outputs to «external output device»
«external user» Interacts with «software system»
«external system» Communicates with «software system»
«external timer» Signals «software system»
```

Examples of associations on software system context diagrams are as follows:

```
Door Sensor Inputs to Microwave Oven Software System
Microwave Oven Software System Outputs to Heating Element
Factory Operator Interacts with Factory Automation Software System
Pick & Place Robot Communicates with Factory Automation Software Syst
Clock Signals Microwave Oven Software System
```

5.6.3 Examples of Software System Context Modeling

An example of a software system context diagram is depicted in the block definition diagram in Figure 5.9, which shows the external blocks to which the Microwave Oven System has to interface. From the total system perspective – that is, both hardware and software – the microwave oven user is external to the system, whereas the I/O devices, which include the door sensor, weight sensor, heating element and lamp, are part of the system. The software system context diagram is modeled from the perspective of the software system to be developed, the Microwave Oven System, which is depicted with the stereotypes «software system» «block».

From the software system point of view, the hardware sensors and actuators are external to the software system and interface to the software system. Thus the blocks outside the software system are the external input and output devices, and the external timer, as depicted in Figure 5.9. In the example, there are three external input device blocks: the `Door Sensor`, the `Weight Sensor`, and the `Keypad`. There are also five external output device blocks, the `Heating Element`, `Display`, `Beeper`, `Turntable`, and `Lamp`, as well as one timer, namely `Timer`. There is one instance of each of these external blocks for a given microwave oven. This example is described in more detail for the Microwave Oven Control System case study in Chapter 19.

A second example of a software system context diagram is from the Railroad Crossing Control System, as depicted in Figure 5.10. This system has three external input devices representing different sensors: `Arrival Sensor`, `Departure Sensor`, and `Barrier Detection Sensor`. There are three output devices representing different actuators: `Barrier Actuator`, `Warning Light Actuator`, and the `Warning Audio Actuator`. There is also an external `Timer`. This example is described in more detail for the Railroad Crossing Control System case study in Chapter 20.

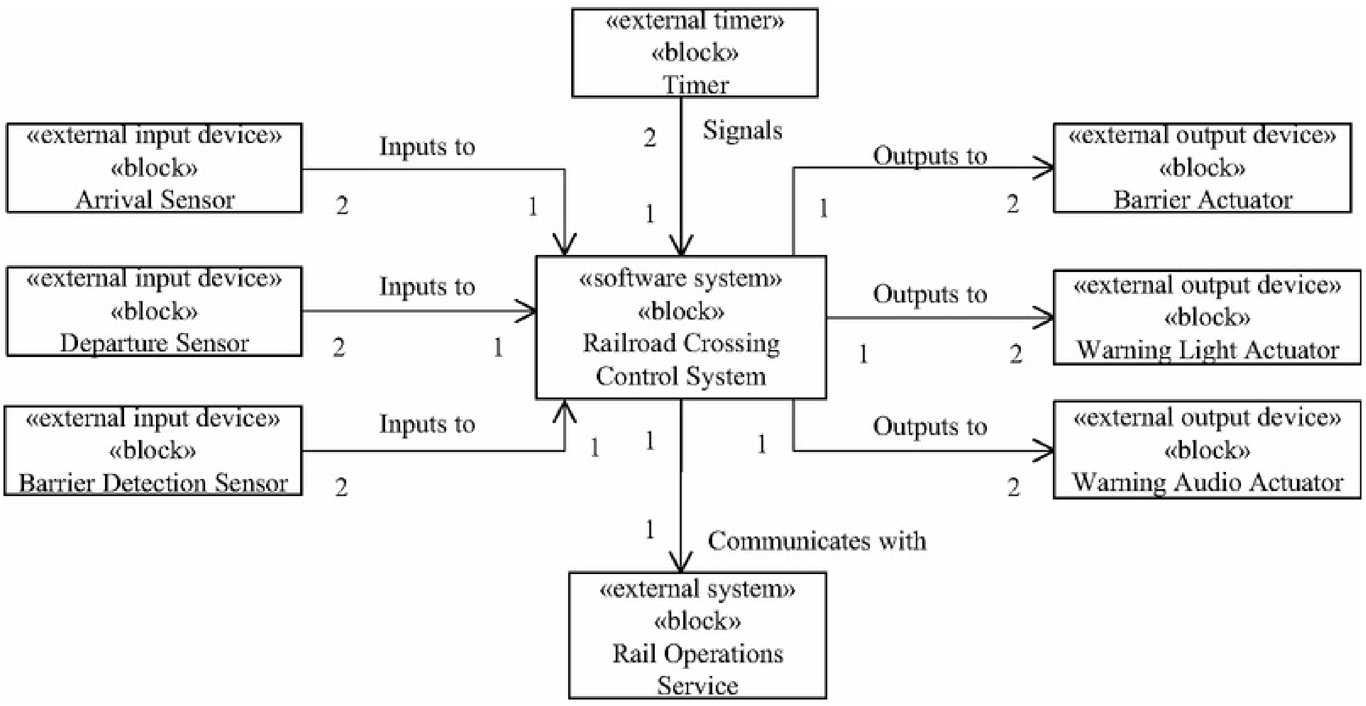
«external timer» «block» Timer
«external input device» «block» Arrival Sensor
«external input device» «block» Departure Sensor
«external input device» «block» Barrier Detection Sensor
«software system» «block» Railroad Crossing Control System
«external output device» «block» Barrier Actuator
«external output device» «block» Warning Light Actuator
«external output device» «block» Warning Audio Actuator
«external system» «block» Rail Operations Service
Signals
Inputs to
Inputs to
Inputs to
Outputs to
Outputs to
Outputs to
Communicates with
1
2

Figure 5.10. Railroad Crossing Control System software system context diagram.

5.7 Defining Hardware/Software Interfaces

In defining the hardware/software boundary, it is also necessary to define the interface between each hardware input and output device and the software system. For example, in the Railroad Crossing Control System, the `Arrival Sensor` input device sends arrival event inputs to the software system. The software system sends switch on and switch off outputs to the `Warning Light Actuator` output device. Specification of the hardware/software boundary needs to clearly describe the function of each I/O device and its interface to the software system. The template for an I/O specifica-tion is:

Name of I/O device:

Type of I/O device:

Function of I/O device:

Inputs from device to software system:

Outputs from software system to device:

An I/O device boundary specification can also be depicted as a table. Examples of input and output device boundary specifications for the Railroad Crossing Control System (Figure 5.10) are given in Table 5.1.

Table 5.1. I/O Device Boundary Specification

Device name	Device type	Device function	Inputs from device	Outputs to device
Arrival Sensor	Input	Signals when train arrives	Arrival Event	
Departure Sensor	Input	Signals when train departs	Departure Event	
Barrier Detection Sensor	Input	Signals when barrier has been raised or lowered	Barrier Lowered Event, Barrier Raised Event	
Barrier Actuator	Output	Raises and lowers barrier		Raise Barrier, Lower

			Barrier
Warning Light Actuator	Output	Switches warning lights on and off	Switch On, Switch Off
Warning Audio Actuator	Output	Switches audio warning on and off	Switch On, Switch Off

5.8 System Deployment Modeling

The next step is to consider the physical deployment of the system (hardware and software) blocks of the embedded system. One possible configuration for the Distributed Light Rail System is depicted in the UML deployment diagram in Figure 5.11, in which the system blocks of the distributed embedded system are deployed to different physical nodes, which are connected by means of a wide area network. The blocks are `Railroad Crossing Control`, which has one node per railroad crossing, `Train Control`, which has one node per train, `Wayside Monitoring`, which has one node per wayside location, `Rail Operations Service`, which has one node, and `Rail Operations Interaction`, which has one node per operator.

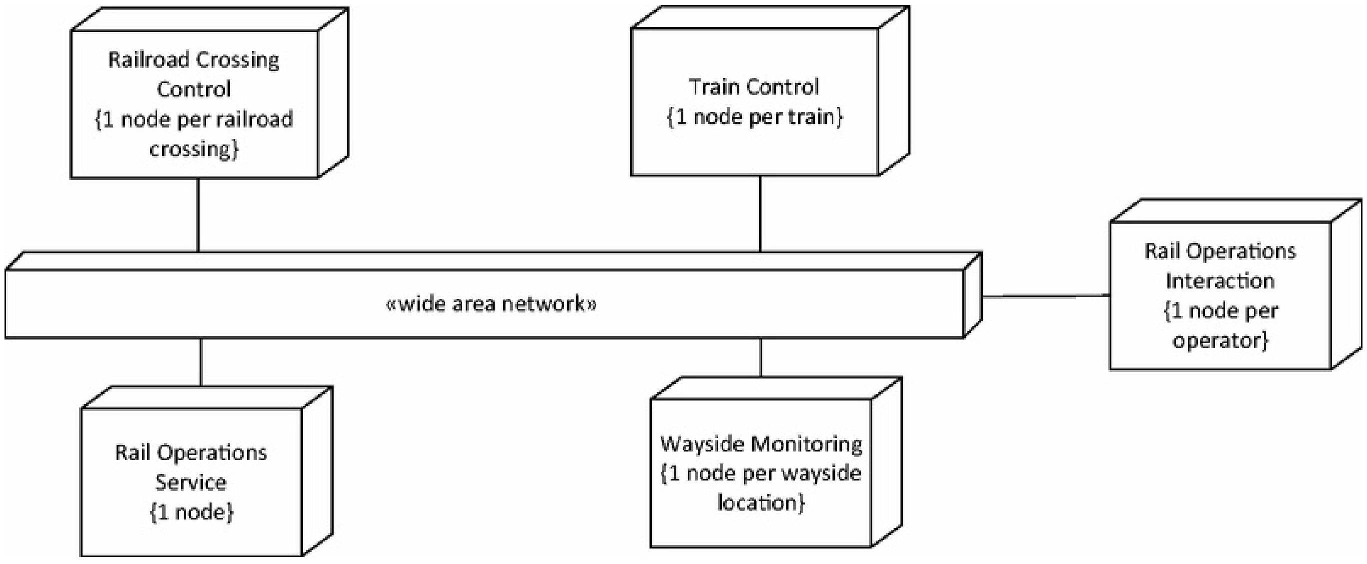

Figure 5.11. Deployment diagram for Distributed Light Rail Embedded System.

5.9 Summary

This chapter has described how *structural modeling* using SysML and UML can be used as an integrated approach for system and software modeling of embedded systems consisting of both hardware and software modeling elements. This chapter started by describing some of the basic concepts of static modeling, including using blocks to depict system modeling elements and classes to depict software modeling elements, as well as defining the relationships between structural modeling elements. Three types of relationships have been described: *associations*, *composition/aggregation* relationships, and *generalization/specialization* relationships. This chapter then described categorization of blocks using stereotypes, structural modeling of the problem domain, system context modeling, developing the hardware/software boundary of a system, software system context modeling, designing the interface between hardware and software blocks, and system deployment modeling. The categorization of software classes using stereotypes is described in Chapter 8.