An overview of the Systems Modeling Language for product and systems development -- Part 2: Structuring the Rain Sensing Wiper system

Laurent Balmelli, PhD

September 15, 2006

from The Rational Edge: The second in a three-part series, this article illustrates how to create a structure diagram for a system using the Systems Modeling Language (SysML), a general-purpose, graphical modeling language for product and systems development. Part 1 introduces the language and describes its requirements, use-case, and test-case diagrams. Part 3 describes how to apply SysML's behavioral diagrams and explains its allocation mechanism. A real-life example of an embedded system is used throughout.

Editor's note: The figures in this article, Part 2 of a three-part series, are numbered in continuation with the figures presented in Part 1.

This multipart article introduces SysML, a standard modeling language for systems engineering. SysML gives systems engineers and architects a much-needed way to collaborate using a common language. By enabling an electronic representation of product design, SysML improves communication across development teams, helps manage system complexity, and can serve as the basis for analytics to drive faster and more effective decision-making across all phases of the systems development lifecycle.

In Part 1 of this three-part article, I explained the basic purpose and value of SysML, related it to Unified Modeling Language (UML), and described its Requirements diagram, Use-Case diagram, and test-case representations. Here in Part 2, I'll cover SysML's structural diagrams and requirements allocation mechanism. Part 3 will explain how SysML can be used to model product functions.

Over the course of the entire three-part article, I'll explore all the diagrams available in SysML, as well as most of the constructs associated with each diagram. Throughout the article, I make
reference to a real-life example of an embedded system, a Rain Sensing Wiper for an automotive application.

Describing system structure

In building a structure for the Rain Sensing Wiper (RSW) system, the assumption is made that a set of subsystems and components have been identified through the requirements engineering process based on stakeholder concerns such as cost, performance, etc.

SysML provides a basic structural element called Block, whose aim is to provide a discipline-agnostic building block for systems. Blocks can be used to represent all types of system components; e.g., functional, physical, human, etc. Blocks assemble to form architectures that represent how different elements in the system coexist.

The SysML Block Definition Diagram (BDD) is the simplest way to describe the structure of the system. It is the equivalent to the Class diagram in UML. It is used to represent the system decomposition using, for example, associations and composition relationships. The BDD is ideal for displaying the features of a Block, such as its properties, and operations. SysML allows Blocks to own special types of properties: Block Properties and Distributed Properties. Block Properties impose additional constraints on classic UML Properties and can, for instance, own a SysML Value Type. Value Types are designed to hold units (e.g., physical units) and dimensions. Distributed Properties let the user apply a probability distribution to the values of the property. SysML proposes model libraries for possible values of units, dimensions, and probability distributions.

Figure 6 shows a BDD for the RSW. For the sake of the diagram's readability, it does not render the associations between the subsystems and the Rain Sensing Wiper element, although these associations exist in the model. Instead, it uses an illustrative box around each set of components (composite and external) and a black diamond shape over the composite component, as visual composition elements. The main components of the RSW are: an interface to actuate the wiper, an electronic control unit, a sensor, and the windshield element. Both the interface and the windshield can exist in the car with or without the RSW. Therefore, in SysML, they are so-called reference properties.

The properties and the operations for each Block are visible in Figure 6. Properties (more precisely SysML Block Properties, shown using the stereotype <<blockProperty>>) are used to model the physical characteristics of the components. The operations (sometimes called services) represent the functional aspects of the system.
Allocating requirements to Blocks

I'll now examine how the product structure and the product requirements can be related. One of the important consequences of having requirements as model elements is that it allows the designer to specify which components in the system satisfy a given set of requirements. This is called the allocation process. An example of requirements allocation is shown in Figure 7. The left-hand side of Figure 7 represents some elements of the RSW, while the right-hand side shows a hierarchy of requirements. One way to perform allocation is to use the Satisfy dependency. In Figure 7, for instance, the Rain Sensing Wiper model element is allocated to the requirement named "Automatic Wiping." Any element in SysML can be used to satisfy a requirement.

Another way to represent allocation is to use a dedicated compartment named "Requirement related." This compartment displays the status of a set of derived properties related to...
requirements. In Figure 7, the element ECU displays such a compartment. Note that the ECU element is allocated to the requirement named "Use dedicated ECU."

![Figure 7: Example of requirements allocation](Click to enlarge)

**The Internal Block Diagram**

The [SysML Internal Block Diagram](IBD) allows the designer to refine the structural aspect of the model. The IBD is the equivalent of the Composite Structure in UML. In the IBD, properties (or parts) are assembled to define how they collaborate to realize the behavior of the Block. A part represents the usage of another Block.

The most important aspect of the IBD is that it allows the designer to refine the definition of the interaction between the usages of Blocks by defining *ports*, as explained below.

Ports are parts available for connection from outside the owning Block. Ports are categorized according to type by the interfaces or Blocks that define what can be exchanged through them. Ports are connected using *connectors* that represent the use of an association in the IBD.

Two types of ports are available in SysML: Standard Ports handle the requests and invocations of services (i.e., function calls) with other Blocks, and Flow Ports let Blocks exchange flows of information or material. For Standard Ports, an *interface* class is used to list the services offered by the Block. For Flow Ports, a *Flow Specification* is created to list the type of data that can flow through the port. When only a single type of object can flow through a port, then the object's type is directly assigned as the port's type. Such a port is called an Atomic Port. The class *Item Flow* is used to represent what does actually flow between Blocks in a particular usage context. I refer the interested reader to the SysML standard specification[^1] for more details on item flows. An example of IBD is shown in Figure 8.

[^1]: An overview of the Systems Modeling Language for product and systems development -- Part 2: Structuring the Rain Sensing Wiper system
In Figure 8, I refine our initial description of the RSW by showing how parts are interacting inside the Block named Rain Sensing Wiper. Prior to constructing the IBD, it is necessary to define a model for the associations characterizing the relationships among the different Blocks. Additional Blocks are also defined, for example, to type the ports. I show this model in another BDD that can be found in the Appendix.

The central part of Figure 8 shows the parts of the system that represent the embedded hardware. The parts underneath are used for mounting the system in the car. The ones above represent the software. A set of standard ports and interfaces are defined to represent the functional aspect of the communication between the parts. For example, the processing unit accesses the actuation interface of the wiper through the interface WiperECUCommunication. Details about the interfaces used in this IBD are discussed in the next installment of this article, Part 3.

The processing unit communicates with the sensor using a Flow Port. The data exchanged is two bitstreams, one containing the measurements from the sensor and another containing synchronization data. The port is typed with a specification of these flows using the element SensorECUCommunication (see Part 3). Note the direction of the flows in the definition.

For convenience, a Flow Port can be conjugated, in the sense that its input and outputs are reversed (flows declared as “in” become “out” and vice-versa) with respect to the definition of the interface. This is useful when connecting two systems whose Flow Ports are conjugated with respect to each other. This is the case, for instance, between the processing unit and the sensor in Figure 8. A conjugated Flow Port is represented in black. Since the synchronization data flow is declared as “inout,” the conjugation of the port has no effect on it.
Note that, in Figure 8, connectors between ports link parts defined within the Block. SysML actually allows direct connections between ports defined at different levels of granularity; for example, between two ports defined inside a part. This type of connector is called a nested connector. Please see the SysML standard specification for more details about these connectors.

Flow Ports are also useful to define physical contact between parts. For example, the SensorAttachment unit is fixed to the windshield using an adhesive. The Block representing the adhesive material (AttachmentAdhesive in Figure App-1 in the Appendix) is used to type the Flow Port connecting these parts.

**Requirements allocation in IBDs**

Requirements are classifiers and therefore cannot be represented on an IBD. For this type of diagram, the compartment notation as introduced in Figure 7 is used to represent requirements allocation.

An example is shown in Figure 8, where the parts representing the windshield and the sensor attachment are used to satisfy the requirement named "Use Sensor on Windshield" (see Figure 3 in Part 1). The satisfaction of these requirements is displayed in each part.

A large and complex model is composed of hundreds, maybe even thousands of elements. Hence, such a model is laid out over a set of IBDs. Most design methodologies advocate the use of viewpoints to organize the content of the model; for example, according to stakeholder's interests. SysML provides a model element Viewpoint that allows users to capture the characteristics of a viewpoint (for example, targeted stakeholders, concerns addressed, construction rules, etc.). A container element called View is then used to organize the model according to the viewpoint description.

Figure 9 is a two-part figure that illustrates separation of concerns using viewpoint modeling. Figure 9a represents a model for the definition of a viewpoint. In the sample model, the elements of the system are contained in a central package called Systems. Some elements are imported from this package into a view called RSW Power, whose purpose is to group elements playing a role in the power consumption of the system. The view conforms to the definition given by the viewpoint description. Within the view, an element Power System RSW is defined to describe how the various imported elements are collaborating within the scope of the power consumption of the system.
Figure 9a: Definition of the RSW Power view
Click to enlarge

Figure 9b: An IBD for the "Power System RSW" Block
Click to enlarge

Figure 9b describes the roles of the imported elements in the context of the power subsystem of the car. The car's electrical system powers the parts using Atomic Flow Ports typed using the Power Supply Channel Block (illustrated in Figure App-1 in the Appendix). In this case, the direction of the port (in or out) is specified in one of the port's attributes.

Defining relationships among attributes using constraints

So far, I've shown how attributes are defined for Blocks in order to represent their physical characteristics. Often the attributes of a set of systems are not independent. Consider two subsystems A and B having attributes a and b, respectively, and that the constraint \{A.a greater than B.b\} must hold true. SysML ConstraintBlocks allows the engineer to define any relationships (e.g., analytical) between the system attributes. These constraints form networks of expressions that are typically leveraged in simulations, such as for requirements verification. Note that Constraint Blocks are not instantiated as runtime objects, but rather, used to type special properties of Blocks, as explained below.

Constraints are properties in subsystems (i.e. Blocks) named ConstraintProperty and are typed by ConstraintBlocks. A Constraint Block defines an expression and the attributes that represent...
its parameters. SysML does not mandate any language to represent the expressions or provide a solver for them. These elements are typically offered by a particular tool vendor.

The RSW uses a set of analytical constraints to verify that the system is properly calibrated (requirement "System Calibration" in Figure 7). Three constraints are shown in Figure 10:

- The constraint `SensorEffectiveRange` computes an operational range for the infrared sensor based on some of its parameters.
- Similarly, the constraint `WindshieldIREffectiveRange` computes an operating range for the windshield, which can be compared with the one computed for the sensor.
- The constraint `SensorWindshieldRangeCompare` is used to compare the above values.

![Figure 10: Definition of Constraint Blocks for the Rain Sensing Wiper system](Click to enlarge)

The SysML **Parametric Diagram (PD)** is used to represent the usage of Constraint Blocks as `ConstraintProperties`. Syntactically, the PD is actually similar to the IBD. In a PD, Constraint Properties are connected to each other through the parameters defined in the Constraint Block of which they represent a usage. In turn, they connect to other properties (Block Properties, Distributed Properties, etc.) in the context of their owning Block. These other properties must be directly bound to parameters of the Constraint Properties because they can only be used as input for their parameters.

An example PD is shown in Figure 11. Constraint Properties are represented by boxes with rounded corners. In this diagram, both the sensor and windshield parts compute an operational range that is compared by the "compare" property. These values are also fed to the part representing the configuration file (at the bottom of Figure 11). If the sensor and the windshield are compatible, the flag `IsCalibrated` (exposed as a port) is set to true. The verification of the calibration requirement is hence reduced to testing the value of this port. The system is therefore resilient to changes in windshield and sensor characteristics.
Requirements allocation is shown in PDs using compartments: In Figure 11, the requirements allocation compartment is displayed in both the constraint used for comparison and the part representing the configuration file. In this example, these elements satisfy the System Calibration requirement.

The usage of the Constraint Blocks WindshieldIREffectiveRange and SensorEffectiveRange can be seen in the diagrams in the Appendix. They are nested in the parts named RainSensor and CarWindshield (see comments in the figures). Note that the parametric diagrams in Figure 11 are used to partially implement the test case presented in Figure 5 in Part 1.

An attractive aspect of Constraint Blocks is that they provide a reusable mechanism to define types of constraints. Hence, the same constraint can be used several times in a model. It is important to note that a constraint does not specify which variable is an input or an output. Values are assigned by the context, and a numerical solver will provide results for the variables of the system. The remarkable work of Dr. Peak et al. is well worth examining on the topic of SysML constraints.²

Next month

In this month’s installment, Part 2, I’ve shown how to use SysML to create a product structure and allocate requirements to it. In the final installment, Part 3, I’ll present an overview of SysML’s capabilities for modeling the operational behavior of systems.
Notes


3 Read all three parts of this series:

1. An overview of the Systems Modeling Language for product and systems development -- Part 1: Requirements, use-case, and test-case modeling
2. An overview of the Systems Modeling Language for product and systems development -- Part 2: Structuring the Rain Sensing Wiper system
3. An Overview of the Systems Modeling Language for product and systems development -- Part 3: Modeling system behavior

© Copyright IBM Corporation 2006

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)