9 Composite Structures

9.1 Overview

The term “structure” in this chapter refers to a composition of interconnected elements, representing run-time instances collaborating over communications links to achieve some common objectives.

Internal Structures

The InternalStructure subpackage provides mechanisms for specifying structures of interconnected elements that are created within an instance of a containing classifier. A structure of this type represents a decomposition of that classifier and is referred to as its “internal structure.”

Ports

The Ports subpackage provides mechanisms for isolating a classifier from its environment. This is achieved by providing a point for conducting interactions between the internals of the classifier and its environment. This interaction point is referred to as a “port.” Multiple ports can be defined for a classifier, enabling different interactions to be distinguished based on the port through which they occur. By decoupling the internals of the classifier from its environment, ports allow a classifier to be defined independently of its environment, making that classifier reusable in any environment that conforms to the interaction constraints imposed by its ports.

Collaborations

Objects in a system typically cooperate with each other to produce the behavior of a system. The behavior is the functionality that the system is required to implement.

A behavior of a collaboration will eventually be exhibited by a set of cooperating instances (specified by classifiers) that communicate with each other by sending signals or invoking operations. However, to understand the mechanisms used in a design, it may be important to describe only those aspects of these classifiers and their interactions that are involved in accomplishing a task or a related set of tasks, projected from these classifiers. *Collaborations* allow us to describe only the relevant aspects of the cooperation of a set of instances by identifying the specific roles that the instances will play. *Interfaces* allow the externally observable properties of an instance to be specified without determining the classifier that will eventually be used to specify this instance. Consequentially, the roles in a collaboration will often be typed by interfaces and will then prescribe properties that the participating instances must exhibit, but will not determine what class will specify the participating instances.

StructuredClasses

The StructuredClasses subpackage supports the representation of classes that may have ports as well as internal structure.

Actions

The Actions subpackage adds actions that are specific to the features introduced by composite structures (e.g., the sending of messages via ports).

9.2 Abstract syntax

Figure 9.1 shows the dependencies of the CompositeStructures packages.

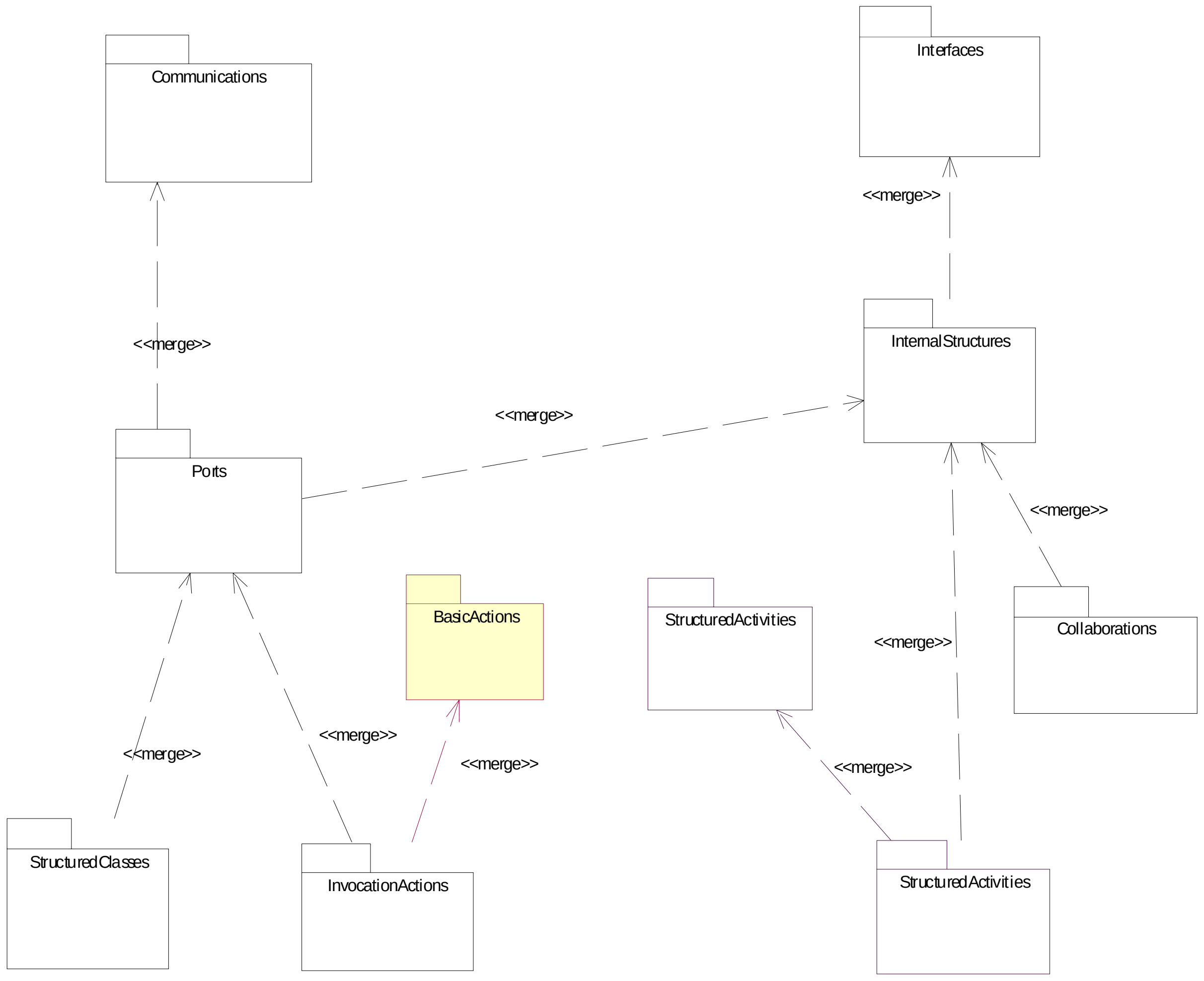


Figure 9.1 - Dependencies between packages described in this chapter

Package InternalStructures

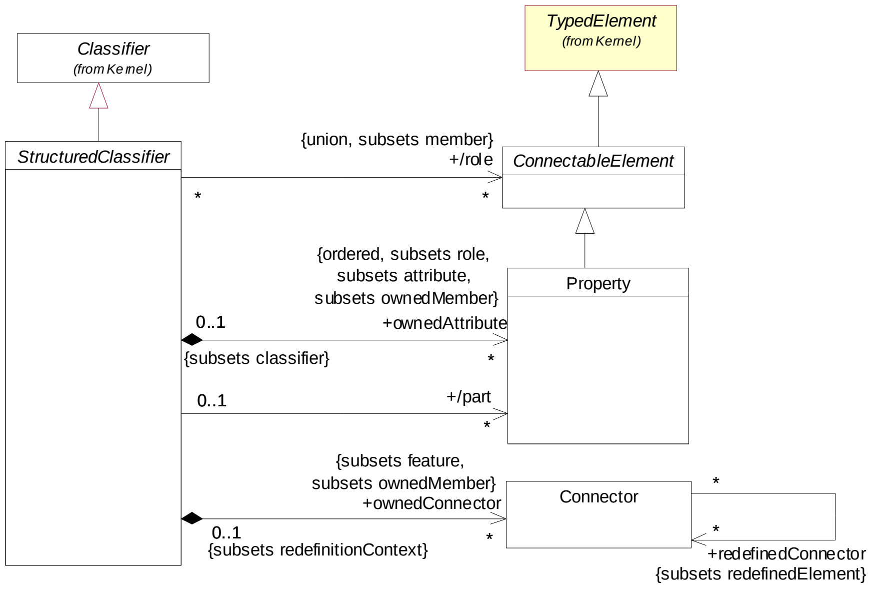


Figure 9.2 - Structured classifier

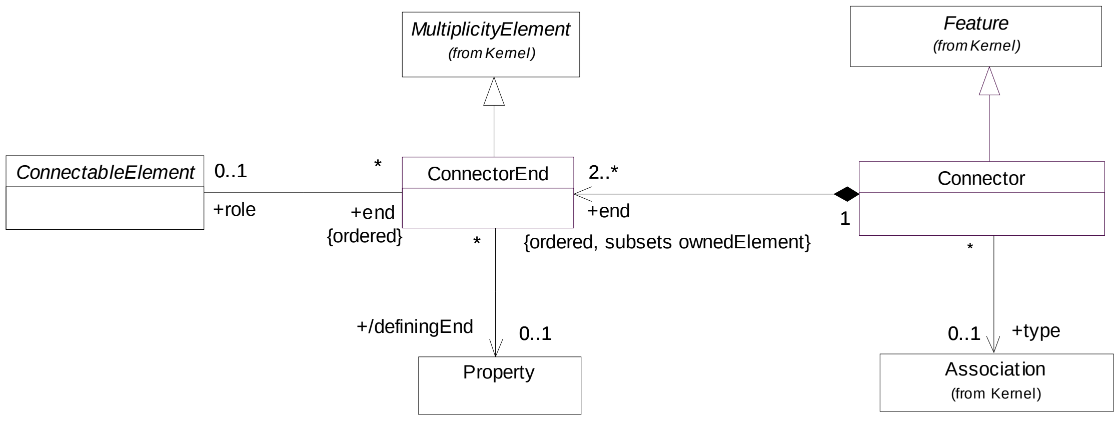


Figure 9.3 - Connectors

Package Ports

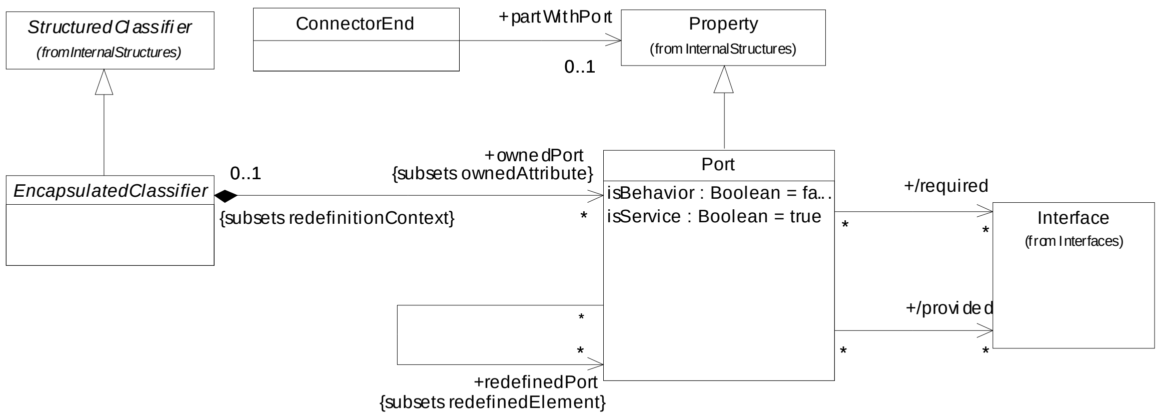


Figure 9.4 - The Port metaclass

Package StructuredClasses

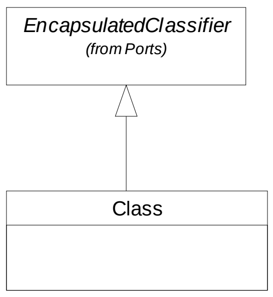


Figure 9.5 - Classes with internal structure

Package Collaborations

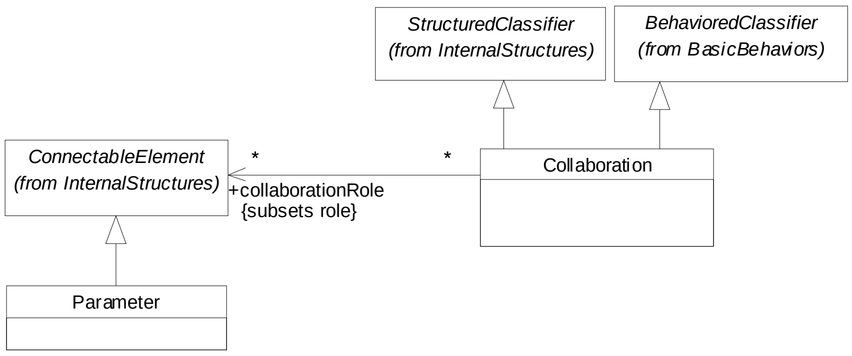


Figure 9.6 - Collaboration

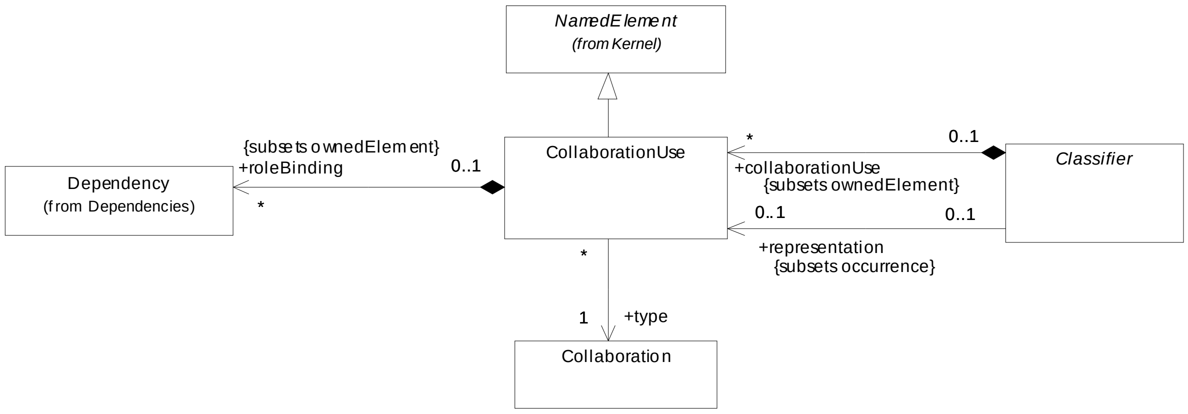


Figure 9.7 - Collaboration.use and role binding

Package InvocationActions

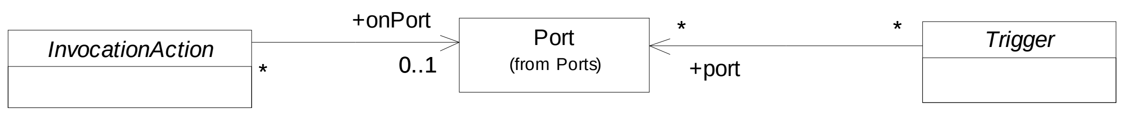


Figure 9.8 - Actions specific to composite structures

Package StructuredActivities

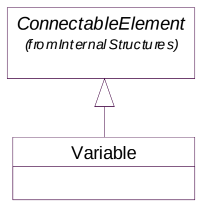


Figure 9.9 - Extension to Variable

9.3 Class Descriptions

9.3.1 Class (from StructuredClasses)

Generalizations

- “EncapsulatedClassifier (from Ports)” on page 173.

Description

Extends the metaclass Class with the capability to have an internal structure and ports.

Semantics

See “Property (from InternalStructures)” on page 179, “Connector (from InternalStructures)” on page 170, and “Port (from Ports)” on page 175 for the semantics of the features of Class. Initialization of the internal structure of a class is discussed in section “StructuredClassifier” on page 179.

A class acts as the namespace for various kinds of classifiers defined within its scope, including classes. Nesting of classifiers limits the visibility of the classifier to within the scope of the namespace of the containing class and is used for reasons of information hiding. Nested classifiers are used like any other classifier in the containing class.

Notation

See “Class (from Kernel)” on page 45, “StructuredClassifier” on page 179, and “Port” on page 175.

Presentation Options

A dashed arrow with an open arrowhead, optionally labeled with the keyword «create», may be used to relate an instance value to a constructor for a class, describing the single value returned by the constructor, which must have the class as its classifier. The arrowhead points at the operation defining the constructor. The constructor may reference parameters declared by the operation. A constructor is any operation having a single return result parameter of the type of the owning class. The instance value at the base of the arrow represents the default value of the single return result parameter of a constructor.

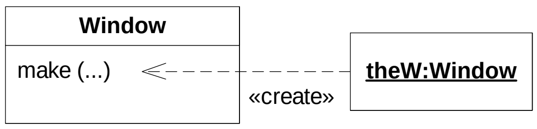


Figure 9.10 - Instance specification describes the return value of an operation

Changes from previous UML

Class has been extended with internal structure and ports.

9.3.2 Classifier (from Collaborations)

Generalizations

- 7.3.8, "Classifier (from Kernel, Dependencies, PowerTypes)," on page 48

Description

Classifier is extended with the capability to own collaboration uses. These collaboration uses link a collaboration with the classifier to give a description of the workings of the classifier.

Associations

• collaborationUse: CollaborationUse	References the collaboration uses owned by the classifier. (Subsets *Element.ownedElement*)
• representation: CollaborationUse [0..1]	References a collaboration use which indicates the collaboration that represents this classifier. (Subsets *Classifier.occurrence*)

Semantics

A classifier can own collaboration uses that relate (aspects of) this classifier to a collaboration. The collaboration describes those aspects of this classifier.

One of the collaboration uses owned by a classifier may be singled out as representing the behavior of the classifier as a whole. The collaboration that is related to the classifier by this collaboration use shows how the instances corresponding to the structural features of this classifier (e.g., its attributes and parts) interact to generate the overall behavior of the classifier. The representing collaboration may be used to provide a description of the behavior of the classifier at a different level of abstraction than is offered by the internal structure of the classifier. The properties of the classifier are mapped to roles in the collaboration by the role bindings of the collaboration use.

Notation

See “CollaborationUse (from Collaborations)” on page 166

Changes from previous UML

Replaces and widens the applicability of *Collaboration.usedCollaboration*. Together with the newly introduced concept of internal structure replaces *Collaboration.representedClassifier.*

9.3.3 Collaboration (from Collaborations)

A collaboration describes a structure of collaborating elements (roles), each performing a specialized function, which collectively accomplish some desired functionality. Its primary purpose is to explain how a system works and, therefore, it typically only incorporates those aspects of reality that are deemed relevant to the explanation. Thus, details, such as the identity or precise class of the actual participating instances are suppressed.

Generalizations

- “BehavioredClassifier (from BasicBehaviors, Communications)” on page 419
- “StructuredClassifier (from InternalStructures)” on page 182

Description

A collaboration is represented as a kind of classifier and defines a set of cooperating entities to be played by instances (its roles), as well as a set of connectors that define communication paths between the participating instances. The cooperating entities are the properties of the collaboration (see “Property (from InternalStructures)” on page 179).

A collaboration specifies a view (or projection) of a set of cooperating classifiers. It describes the required links between instances that play the roles of the collaboration, as well as the features required of the classifiers that specify the participating instances. Several collaborations may describe different projections of the same set of classifiers.

Attributes

No additional attributes

Associations

• collaborationRole: ConnectableElement [*]	References connectable elements (possibly owned by other classifiers), which represent roles that instances may play in this collaboration. (Subsets *StructuredClassifier.role*)

Constraints

No additional constraints

Semantics

Collaborations are generally used to explain how a collection of cooperating instances achieve a joint task or set of tasks. Therefore, a collaboration typically incorporates only those aspects that are necessary for its explanation and suppresses everything else. Thus, a given object may be simultaneously playing roles in multiple different collaborations, but each collaboration would only represent those aspects of that object that are relevant to its purpose.

A collaboration defines a set of cooperating participants that are needed for a given task. The roles of a collaboration will be played by instances when interacting with each other. Their relationships relevant for the given task are shown as connectors between the roles. Roles of collaborations define a usage of instances, while the classifiers typing these roles specify all required properties of these instances. Thus, a collaboration specifies what properties instances must have to be able to participate in the collaboration. A role specifies (through its type) the required set of features a participating instance must have. The connectors between the roles specify what communication paths must exist between the participating instances.

Neither all features nor all contents of the participating instances nor all links between these instances are always required in a particular collaboration. Therefore, a collaboration is often defined in terms of roles typed by interfaces (see "Interface (from Interfaces)" on page 82). An interface is a description of a set of properties (externally observable features) required or provided by an instance. An interface can be viewed as a projection of the externally observable features of a classifier realizing the interface. Instances of different classifiers can play a role defined by a given interface, as long as these classifiers realize the interface (i.e., have all the required properties). Several interfaces may be realized by the same classifier, even in the same context, but their features may be different subsets of the features of the realizing classifier.

Collaborations may be specialized from other collaborations. If a role is extended in the specialization, the type of a role in the specialized collaboration must conform to the type of the role in the general collaboration. The specialization of the types of the roles does not imply corresponding specialization of the classifiers that realize those roles. It is sufficient that they conform to the constraints defined by those roles.

A collaboration may be attached to an operation or a classifier through a CollaborationUse. A collaboration used in this way describes how this operation or this classifier is realized by a set of cooperating instances. The connectors defined within the collaboration specify links between the instances when they perform the behavior specified in the classifier. The collaboration specifies the context in which behavior is performed. Such a collaboration may constrain the set of valid interactions that may occur between the instances that are connected by a link.

A collaboration is not directly instantiable. Instead, the cooperation defined by the collaboration comes about as a consequence of the actual cooperation between the instances that play the roles defined in the collaboration (the collaboration is a selective view of that situation).

Notation

A collaboration is shown as a dashed ellipse icon containing the name of the collaboration. The internal structure of a collaboration as comprised by roles and connectors may be shown in a compartment within the dashed ellipse icon. Alternatively, a composite structure diagram can be used.

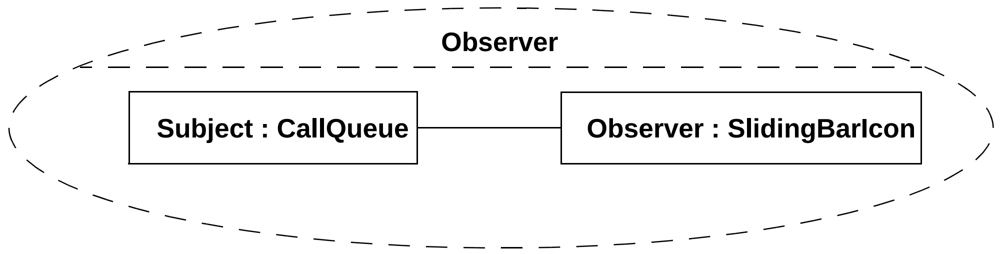


Figure 9.11 - The internal structure of the Observer collaboration shown inside the collaboration icon (a connection is shown between the Subject and the Observer role).

Using an alternative notation for properties, a line may be drawn from the collaboration icon to each of the symbols denoting classifiers that are the types of properties of the collaboration. Each line is labeled by the name of the property. In this manner, a collaboration icon can show the use of a collaboration together with the actual classifiers that occur in that particular use of the collaboration (see Figure 9.12).

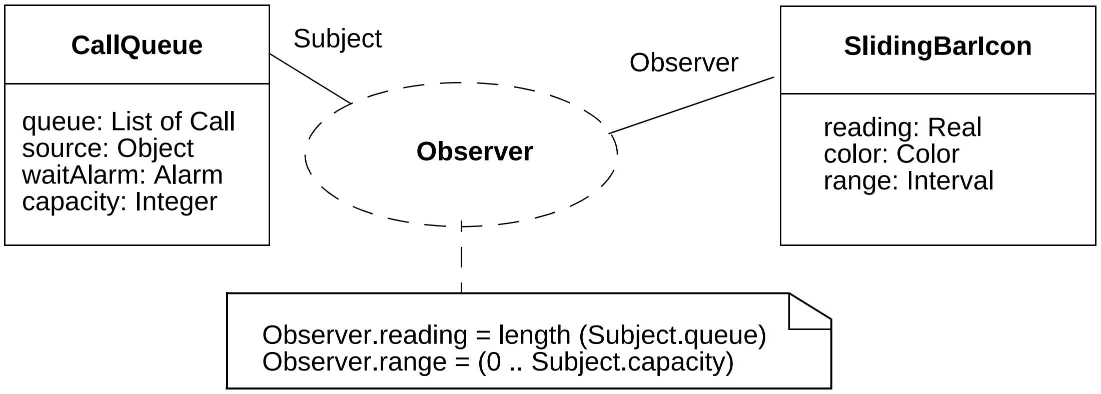


Figure 9.12 - In the Observer collaboration two roles, a Subject and an Observer, collaborate to produce the desired behavior. Any instance playing the Subject role must possess the properties specified by CallQueue, and similarly for the Observer role.

Rationale

The primary purpose of collaborations is to explain how a system of communicating entities collectively accomplish a specific task or set of tasks without necessarily having to incorporate detail that is irrelevant to the explanation. It is particularly useful as a means for capturing standard design patterns.

Changes from previous UML

The contents of a collaboration is specified as its internal structure relying on roles and connectors; the concepts of ClassifierRole, AssociationRole, and AssociationEndRole have been superseded. A collaboration in UML 2.0 is a kind of classifier, and can have any kind of behavioral descriptions associated. There is no loss in modeling capabilities.

9.3.4 CollaborationUse (from Collaborations)

A collaboration use represents the application of the pattern described by a collaboration to a specific situation involving specific classes or instances playing the roles of the collaboration.

Generalizations

- “NamedElement (from Kernel, Dependencies)” on page 93

Description

A collaboration use represents one particular use of a collaboration to explain the relationships between the properties of a classifier. A collaboration use shows how the pattern described by a collaboration is applied in a given context, by binding specific entities from that context to the roles of the collaboration. Depending on the context, these entities could be structural features of a classifier, instance specifications, or even roles in some containing collaboration. There may be multiple occurrences of a given collaboration within a classifier, each involving a different set of roles and connectors. A given role or connector may be involved in multiple occurrences of the same or different collaborations.

Associated dependencies map features of the collaboration type to features in the classifier. These dependencies indicate which role in the classifier plays which role in the collaboration.

Attributes

No additional attributes

Associations

• type: Collaboration [1]	The collaboration that is used in this occurrence. The collaboration defines the cooperation between its roles that are mapped to properties of the classifier owning the collaboration use.
• roleBinding: Dependency [*]	A mapping between features of the collaboration type and features of the classifier or operation. This mapping indicates which connectable element of the classifier or operation plays which role(s) in the collaboration. A connectable element may be bound to multiple roles in the same collaboration use (that is, it may play multiple roles).

Constraints

[1] All the client elements of a *roleBinding* are in one classifier and all supplier elements of a *roleBinding* are in one collaboration and they are compatible.

[2] Every role in the collaboration is bound within the collaboration use to a connectable element within the classifier or operation.

[3] The connectors in the classifier connect according to the connectors in the collaboration.

Semantics

A collaboration use relates a feature in its collaboration type to a connectable element in the classifier or operation that owns the collaboration use.

Any behavior attached to the collaboration type applies to the set of roles and connectors bound within a given collaboration use. For example, an interaction among parts of a collaboration applies to the classifier parts bound to a single collaboration use. If the same connectable element is used in both the collaboration and the represented element, no role binding is required.

Semantic Variation Points

It is a semantic variation when client and supplier elements in role bindings are compatible.

Notation

A collaboration use is shown by a dashed ellipse containing the name of the occurrence, a colon, and the name of the collaboration type. For every role binding, there is a dashed line from the ellipse to the client element; the dashed line is labeled on the client end with the name of the supplier element.

Presentation Options

A dashed arrow with an open arrowhead may be used to show that a collaboration is used in a classifier, optionally labeled with the keyword «represents». A dashed arrow with an open arrowhead may also be used to show that a collaboration represents a classifier, optionally labeled with the keyword «occurrence». The arrowhead points at the owning classifier. When using this presentation option, the role bindings are shown explicitly as dependencies.

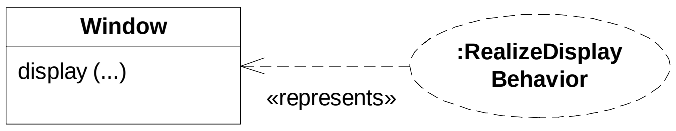


Figure 9.13 - Collaboration occurrence relates a classifier to a collaboration

Examples

This example shows the definition of two collaborations, *Sale* (Figure 9.14) and *BrokeredSale* (Figure 9.15). *Sale* is used twice as part of the definition of *BrokeredSale*. *Sale* is a collaboration among two roles, a *seller* and a *buyer.* An interaction, or other behavior specification, could be attached to *Sale* to specify the steps involved in making a *Sale*.

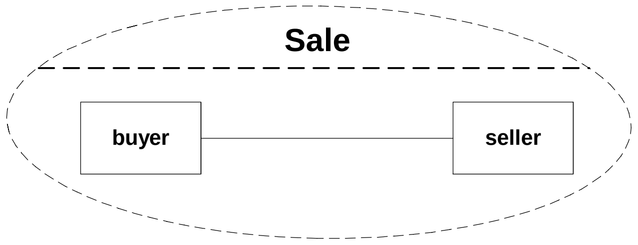


Figure 9.14 - The Sale collaboration

BrokeredSale is a collaboration among three roles, a *producer*, a *broker*, and a *consumer.* The specification of *BrokeredSale* shows that it consists of two occurrences of the *Sale* collaboration, indicated by the dashed ellipses. The occurrence *wholesale* indicates a *Sale* in which the *producer* is the *seller* and the *broker* is the *buyer.* The occurrence *retail* indicates a *Sale* in which the *broker* is the *seller* and the *consumer* is the *buyer.* The connectors between *sellers* and *buyers* are not shown in the two occurrences; these connectors are implicit in the *BrokeredSale* collaboration in virtue of them being comprised of *Sale*. The *BrokeredSale* collaboration could itself be used as part of a larger collaboration.

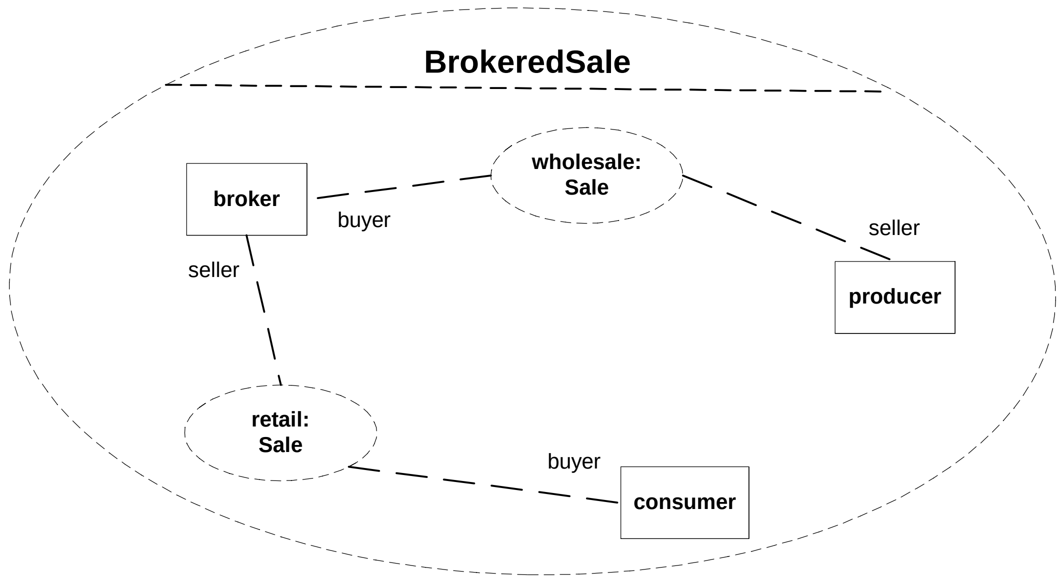


Figure 9.15 - The BrokeredSale collaboration

Figure 9.16 shows part of the *BrokeredSale* collaboration in a presentation option.

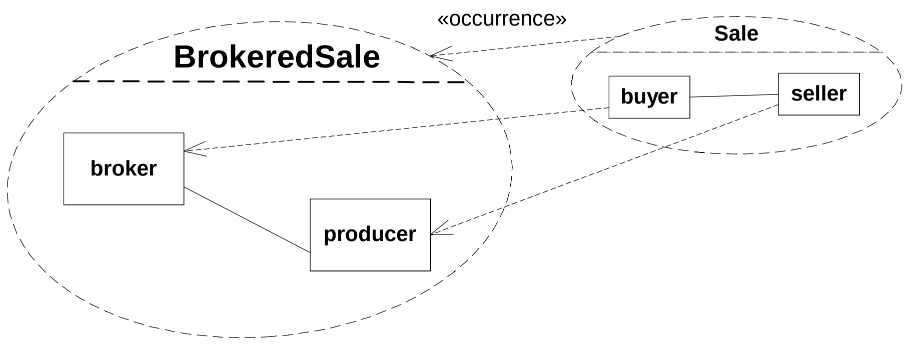


Figure 9.16 - A subset of the BrokeredSale collaboration

Rationale

A collaboration use is used to specify the application of a pattern specified by a collaboration to a specific situation. In that regard, it acts as the invocation of a macro with specific values used for the parameters (roles).

Changes from previous UML

This metaclass has been added.

9.3.5 ConnectableElement (from InternalStructures)

Generalizations

- “TypedElement (from Kernel)” on page 131

Description

A ConnectableElement is an abstract metaclass representing a set of instances that play roles of a classifier. Connectable elements may be joined by attached connectors and specify configurations of linked instances to be created within an instance of the containing classifier.

Attributes

No additional attributes

Associations

• end: ConnectorEnd	Denotes a connector that attaches to this connectable element.

Constraints

No additional constraints

Semantics

The semantics of ConnectableElement is given by its concrete subtypes.

Notation

None

Rationale

This metaclass supports factoring out the ability of a model element to be linked by a connector.

Changes from previous UML

This metaclass generalizes the concept of classifier role from 1.x.

9.3.6 Connector (from InternalStructures)

Specifies a link that enables communication between two or more instances. This link may be an instance of an association, or it may represent the possibility of the instances being able to communicate because their identities are known by virtue of being passed in as parameters, held in variables or slots, or because the communicating instances are the same instance. The link may be realized by something as simple as a pointer or by something as complex as a network connection. In contrast to associations, which specify links between any instance of the associated classifiers, connectors specify links between instances playing the connected parts only.

Generalizations

- “Feature (from Kernel)” on page 66

Description

Each connector may be attached to two or more connectable elements, each representing a set of instances. Each connector end is distinct in the sense that it plays a distinct role in the communication realized over a connector. The communications realized over a connector may be constrained by various constraints (including type constraints) that apply to the attached connectable elements.

Attributes

No additional attributes

Associations

- end: ConnectorEnd [2..*]
  A connector consists of at least two connector ends, each representing the participation of instances of the classifiers typing the connectable elements attached to this end. The set of connector ends is ordered. (Subsets*Element.ownedElement*)
- type: Association [0..1]
  An optional association that specifies the link corresponding to this connector.
- redefinedConnector: Connector [0..*]
  A connector may be redefined when its containing classifier is specialized. The redefining connector may have a type that specializes the type of the redefined connector. The types of the connector ends of the redefining connector may specialize the types of the connector ends of the redefined connector. The properties of the connector ends of the redefining connector may be replaced. (Subsets *Element.redefinedElement*)

Constraints

[1] The types of the connectable elements that the ends of a connector are attached to must conform to the types of the association ends of the association that types the connector, if any.

[2] The connectable elements attached to the ends of a connector must be compatible.

[3] The ConnectableElements attached as roles to each ConnectorEnd owned by a Connector must be roles of the Classifier that owned the Connector, or they must be ports of such roles.

Semantics

If a connector between two roles of a classifier is a feature of an instantiable classifier, it declares that a link may exist within an instance of that classifier. If a connector between two roles of a classifier is a feature of an uninstantiable classifier, it declares that links may exist within an instance of the classifier that realizes the original classifier. These links will connect instances corresponding to the parts joined by the connector.

Links corresponding to connectors may be created upon the creation of the instance of the containing classifier (see “StructuredClassifier” on page 179). The set of links is a subset of the total set of links specified by the association typing the connector. All links are destroyed when the containing classifier instance is destroyed.

If the type of the connector is omitted, the type is inferred based on the connector, as follows: If the type of a role (i.e, the connectable element attached to a connector end) realizes an interface that has a unique association to another interface which is realized by the type of another role (or an interface compatible to that interface is realized by the type of another role), then that association is the type of the connector between these parts. If the connector realizes a collaboration (that is, a collaboration use maps the connector to a connector in an associated collaboration through role bindings), then the type of the connector is an anonymous association with association ends corresponding to each connector end. The type of each association end is the classifier that realizes the parts connected to the matching connector in the collaboration.

Any adornments on the connector ends (either the original connector or the connector in the collaboration) specify adornments of the ends of the inferred association. Otherwise, the type of the connector is an anonymously named association with association ends corresponding to each connector end. The type of each association end is the type of the part that each corresponding connector end is attached to. Any adornments on the connector ends specify adornments of the ends of the inferred association. Any inferred associations are always bidirectionally navigable and are owned by the containing classifier.

Semantic Variation Points

What makes connectable elements compatible is a semantic variation point.

Notation

A connector is drawn using the notation for association (see “Association (from Kernel)” on page 36). The optional name string of the connector obeys the following syntax:

([name] ‘:’ <classname>) | <name>

where <*name*> is the name of the connector, and <*classname*> is the name of the association that is its type. A stereotype keyword within guillemets may be placed above or in front of the connector name. A property string may be placed after or below the connector name.

Examples

Examples are shown in “StructuredClassifier” on page 179.

Changes from previous UML

Connector has been added in UML 2.0. The UML 1.4 concept of association roles is subsumed by connectors.

9.3.7 ConnectorEnd (from InternalStructures, Ports)

Generalizations

- “MultiplicityElement (from Kernel)” on page 90

Description

A connector end is an endpoint of a connector, which attaches the connector to a connectable element. Each connector end is part of one connector.

Attributes

No additional attributes

Associations

InternalStructures

• role: ConnectableElement [1]	The connectable element attached at this connector end. When an instance of the containing classifier is created, a link may (depending on the multiplicities) be created to an instance of the classifier that types this connectable element.

• definingEnd: Property [0..1]	A derived association referencing the corresponding association end on the association that types the connector owing this connector end. This association is derived by selecting the association end at the same place in the ordering of association ends as this connector end.

Ports

• partWithPort: Property [0..1]	Indicates the role of the internal structure of a classifier with the port to which the connector end is attached.

Constraints

[1] If a connector end is attached to a port of the containing classifier, *partWithPort* will be empty.

[2] If a connector end references both a *role* and a *partWithPort*, then the *role* must be a port that is defined by the type of the *partWithPort*.

[3] The property held in self.partWithPort must not be a Port.

Semantics

InternalStructures

A connector end describes which connectable element is attached to the connector owning that end. Its multiplicity indicates the number of instances that may be linked to each instance of the property connected on the other end.

Notation

InternalStructures

Adornments may be shown on the connector end corresponding to adornments on association ends (see “Association (from Kernel)” on page 36). These adornments specify properties of the association typing the connector. The multiplicity indicates the number of instances that may be connected to each instance of the role on the other end. If no multiplicity is specified, the multiplicity matches the multiplicity of the role the end is attached to.

Ports

If the end is attached to a port on a part of the internal structure and no multiplicity is specified, the multiplicity matches the multiplicity of the port multiplied by the multiplicity of the part (if any).

Changes from previous UML

Connector end has been added in UML 2.0. The UML 1.4 concept of association end roles is subsumed by connector ends.

9.3.8 EncapsulatedClassifier (from Ports)

Generalizations

- “StructuredClassifier (from InternalStructures)” on page 182

Description

Extends a classifier with the ability to own ports as specific and type checked interaction points.

Attributes

No additional attributes

Associations

• ownedPort: Port	References a set of ports that an encapsulated classifier owns. (Subsets *Classifier.feature* and *Namespace.ownedMember*)

Constraints

No additional constraints

Semantics

See "Port" on page 175.

Notation

See "Port" on page 175.

Changes from previous UML

This metaclass has been added to UML.

9.3.9 InvocationAction (from Actions)

Generalizations

- "InvocationAction (from BasicActions)" on page 249 *(merge increment)*

Description

In addition to targeting an object, invocation actions can also invoke behavioral features on ports from where the invocation requests are routed onwards on links deriving from attached connectors. Invocation actions may also be sent to a target via a given port, either on the sending object or on another object.

Associations

• onPort: Port [0..1]	An optional port of the receiver object on which the behavioral feature is invoked.

Constraints

[1] The *onPort* must be a port on the receiver object.

Semantics

The target value of an invocation action may also be a port. In this case, the invocation request is sent to the object owning this port as identified by the port identity, and is, upon arrival, handled as described in "Port" on page 175.

Notation

The optional port is identified by the phrase "via <port>" in the name string of the icon denoting the particular invocation action (for example, see "CallOperationAction (from BasicActions)" on page 239).

9.3.10 Parameter (from Collaborations)

Generalizations

- "ConnectableElement (from InternalStructures)" on page 170
- "Parameter (from Kernel, AssociationClasses)" on page 115 *(merge increment)*

Description

Parameters are allowed to be treated as connectable elements.

Constraints

[1] A parameter may only be associated with a connector end within the context of a collaboration.

9.3.11 Port (from Ports)

A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts. Ports are connected to properties of the classifier by connectors through which requests can be made to invoke the behavioral features of a classifier. A Port may specify the services a classifier provides (offers) to its environment as well as the services that a classifier expects (requires) of its environment.

Generalizations

- "Property (from InternalStructures)" on page 179

Description

Ports represent interaction points between a classifier and its environment. The interfaces associated with a port specify the nature of the interactions that may occur over a port. The required interfaces of a port characterize the requests that may be made from the classifier to its environment through this port. The provided interfaces of a port characterize requests to the classifier that its environment may make through this port.

A port has the ability to specify that any requests arriving at this port are handled by the behavior of the instance of the owning classifier, rather than being forwarded to any contained instances, if any.

Attributes

- isService: Boolean
  If *true*, indicates that this port is used to provide the published functionality of a classifier. If *false*, this port is used to implement the classifier but is not part of the essential externally-visible functionality of the classifier and can, therefore, be altered or deleted along with the internal implementation of the classifier and other properties that are considered part of its implementation. The default value for this attribute is *true*.
- isBehavior: Boolean
  Specifies whether requests arriving at this port are sent to the classifier behavior of this classifier (see "BehavioredClassifier (from BasicBehaviors, Communications)" on page 419). Such ports are referred to as *behavior port*. Any invocation of a behavioral feature targeted at a behavior port will be handled by the instance of the owning classifier itself, rather than by any instances that this classifier may contain. The default value is *false*.

Associations

- required: Interface
  References the interfaces specifying the set of operations and receptions that the classifier expects its environment to handle. This association is derived as the set of interfaces required by the type of the port or its supertypes.
- provided: Interface
  References the interfaces specifying the set of operations and receptions that the classifier offers to its environment, and which it will handle either directly or by forwarding it to a part of its internal structure. This association is derived from the interfaces realized by the type of the port or by the type of the port, if the port was typed by an interface.
- redefinedPort: Port
  A port may be redefined when its containing classifier is specialized. The redefining port may have additional interfaces to those that are associated with the redefined port or it may replace an interface by one of its subtypes. (Subsets *Element.redefinedElement*)

Constraints

[1] The required interfaces of a port must be provided by elements to which the port is connected.

[2] Port.aggregation must be *composite*.

[3] When a port is destroyed, all connectors attached to this port will be destroyed also.

[4] A defaultValue for port cannot be specified when the type of the Port is an Interface.

Semantics

A port represents an interaction point between a classifier instance and its environment or between a classifier instance and instances it may contain. A port by default has public visibility. However, a behavior port may be hidden but does not have to be.

The required interfaces characterize services that the owning classifier expects from its environment and that it may access through this interaction point: Instances of this classifier expect that the features owned by its required interfaces will be offered by one or more instances in its environment. The provided interfaces characterize the behavioral features that the owning classifier offers to its environment at this interaction point. The owning classifier must offer the features owned by the provided interfaces.

The provided and required interfaces completely characterize any interaction that may occur between a classifier and its environment at a port with respect to the data communicated at this port and the behaviors that may be invoked through this port. The interfaces do not necessarily establish the exact sequences of interactions across the port. When an instance of a classifier is created, instances corresponding to each of its ports are created and held in the slots specified by the ports, in accordance with its multiplicity. These instances are referred to as “interaction points” and provide unique references. A link from that instance to the instance of the owning classifier is created through which communication is forwarded to the instance of the owning classifier or through which the owning classifier communicates with its environment. It is, therefore, possible for an instance to differentiate between requests for the invocation of a behavioral feature targeted at its different ports. Similarly, it is possible to direct such requests at a port, and the requests will be routed as specified by the links corresponding to connectors attached to this port. (In the following, “requests arriving at a port” shall mean “request occurrences arriving at the interaction point of this instance corresponding to this port.”)

The interaction point object must be an instance of a classifier that realizes the provided interfaces of the port. If the port was typed by an interface, the classifier typing the interaction point object realizes that interface. If the port was typed by a class, the interaction point object will be an instance of that class. The latter case allows elaborate specification of the communication over a port. For example, it may describe that communication is filtered, modified in some way, or routed to other parts depending on its contents as specified by the classifier that types the port.

If connectors are attached to both the port when used on a property within the internal structure of a classifier and the port on the container of an internal structure, the instance of the owning classifier will forward any requests arriving at this port along the link specified by those connectors. If there is a connector attached to only one side of a port, any requests arriving at this port will terminate at this port.

For a behavior port, the instance of the owning classifier will handle requests arriving at this port (as specified in the behavior of the classifier, see Chapter 13, “Common Behaviors”), if this classifier has any behavior. If there is no behavior defined for this classifier, any communication arriving at a behavior port is lost.

Semantic Variation Points

If several connectors are attached on one side of a port, then any request arriving at this port on a link derived from a connector on the other side of the port will be forwarded on links corresponding to these connectors. It is a semantic variation point whether these requests will be forwarded on all links, or on only one of those links. In the latter case, one possibility is that the link at which this request will be forwarded will be arbitrarily selected among those links leading to an instance that had been specified as being able to handle this request (i.e., this request is specified in a provided interface of the part corresponding to this instance).

Notation

A port of a classifier is shown as a small square symbol. The name of the port is placed near the square symbol. If the port symbol is placed overlapping the boundary of the rectangle symbol denoting that classifier this port is exposed (i.e., its visibility is *public*). If the port is shown inside the rectangle symbol, then the port is hidden and its visibility is as specified (it is *protected* by default).

A port of a classifier may also be shown as a small square symbol overlapping the boundary of the rectangle symbol denoting a part typed by that classifier (see Figure 9.17). The name of the port is shown near the port; the multiplicity follows the name surrounded by brackets. Name and multiplicity may be elided.

The type of a port may be shown following the port name, separated by colon (“:”). A provided interface may be shown using the “lollipop” notation (see “Interface (from Interfaces)” on page 82) attached to the port. A required interface may be shown by the “socket” notation attached to the port. The presentation options shown there are also applicable to interfaces of ports. Figure 9.17 shows the notation for ports: *p* is a port on the *Engine* class. The provided interface (also its type) of port *p* is *powertrain*. The multiplicity of *p* is “1.” In addition, a required interface, *power*, is shown also. The figure on the left shows the provided interface using the “lollipop” notation, while the figure on the right shows the interface as the type of the port.

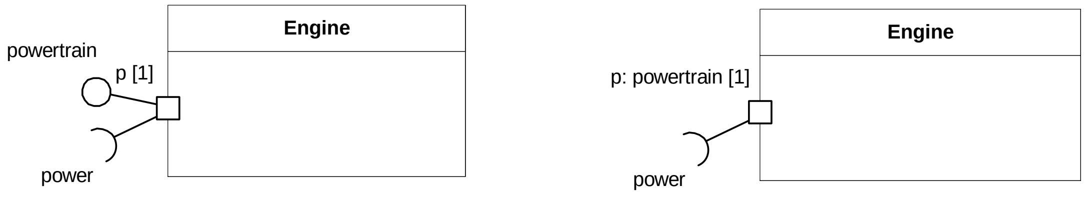


Figure 9.17 - Port notation

A behavior port is indicated by a port being connected through a line to a small state symbol drawn inside the symbol representing the containing classifier. (The small state symbol indicates the behavior of the containing classifier.) Figure 9.18 shows the behavior port *p*, as indicated by its connection to the state symbol representing the behavior of the *Engine* class. Its provided interface is *powertrain*. In addition, a required interface, *power*, is shown also.

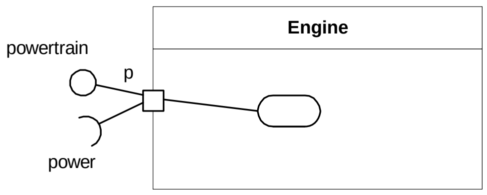


Figure 9.18 - Behavior port notation

Presentation Options

The name of a port may be suppressed. Every depiction of an unnamed port denotes a different port from any other port.

If there are multiple interfaces associated with a port, these interfaces may be listed with the interface icon, separated by commas. Figure 9.19 below shows a port *OnlineServices* on the *OrderProcess* class with two provided interfaces, *OrderEntry* and *Tracking*, as well as a required interface *Payment*.

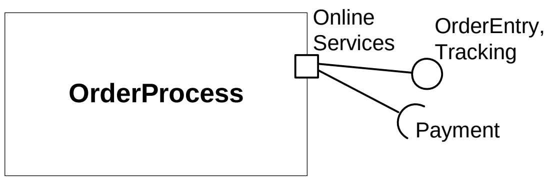


Figure 9.19 - Port notation showing multiple provided interfaces

Examples

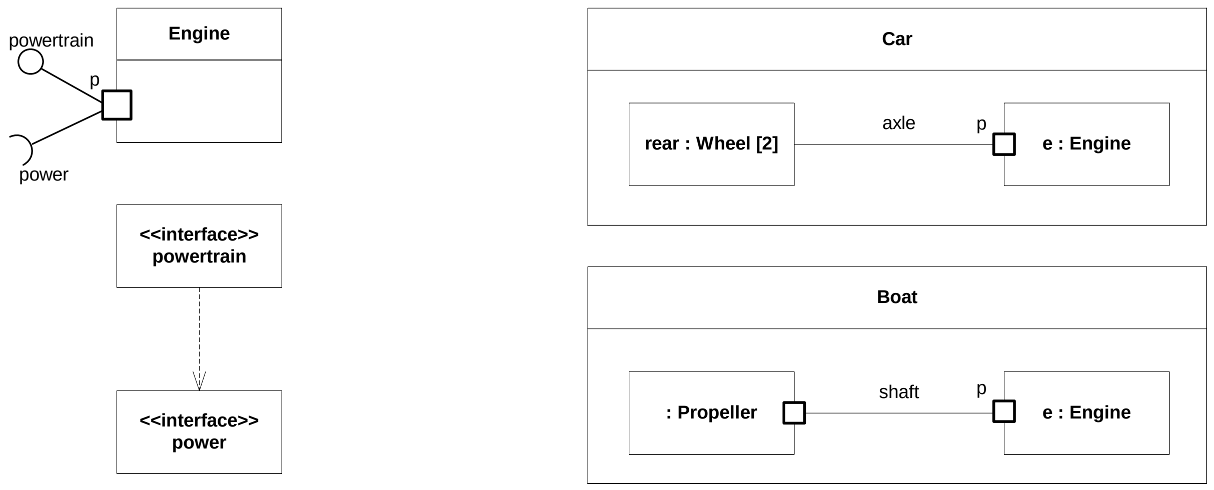


Figure 9.20 - Port examples

Figure 9.20 shows a class *Engine* with a port *p* with a provided interface *powertrain*. This interface specifies the services that the engine offers at this port (i.e., the operations and receptions that are accessible by communication arriving at this port). The interface *power* is the required interface of the engine. The required interface specifies the services that the engine expects its environment to provide. At port *p*, the *Engine* class is completely encapsulated; it can be specified without any knowledge of the environment the engine will be embedded in. As long as the environment obeys the constraints expressed by the provided and required interfaces of the engine, the engine will function properly.

Two uses of the *Engine* class are depicted: Both a boat and a car contain a part that is an engine. The *Car* class connects port *p* of the engine to a set of wheels via the *axle*. The *Boat* class connects port *p* of the engine to a propeller via the *shaft*. As long as the interaction between the *Engine* and the part linked to its port *p* obeys the constraints specified by the provided and required interfaces, the engine will function as specified, whether it is an engine of a car or an engine of a boat. (This example also shows that connectors need not necessarily attach to parts via ports (as shown in the *Car* class.)

Rationale

The required and provided interfaces of a port specify everything that is necessary for interactions through that interaction point. If all interactions of a classifier with its environment are achieved through ports, then the internals of the classifier are fully isolated from the environment. This allows such a classifier to be used in any context that satisfies the constraints specified by its ports.

Changes from previous UML

This metaclass has been added to UML.

9.3.12 Property (from InternalStructures)

Generalizations

- "Property (from Kernel, AssociationClasses)" on page 118 *(merge increment)*

Description

A property represents a set of instances that are owned by a containing classifier instance.

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

Semantics

When an instance of the containing classifier is created, a set of instances corresponding to its properties may be created either immediately or at some later time. These instances are instances of the classifier typing the property. A property specifies that a set of instances may exist; this set of instances is a subset of the total set of instances specified by the classifier typing the property.

A part declares that an instance of this classifier may contain a set of instances by composition. All such instances are destroyed when the containing classifier instance is destroyed. Figure 9.21 shows two possible views of the *Car* class. In subfigure (i), *Car* is shown as having a composition association with role name *rear* to a class *Wheel* and an association with role name *e* to a class *Engine*. In subfigure (ii), the same is specified. However, in addition, in subfigure (ii) it is specified that *rear* and *e* belong to the internal structure of the class *Car.* This allows specification of detail that holds only for instances of the *Wheel* and *Engine* classes within the context of the class *Car*, but which will not hold for wheels and engines in general. For example, subfigure (i) specifies that any instance of class *Engine* can be linked to an arbitrary number of instances of class *Wheel*. Subfigure (ii), however, specifies that within the context of class *Car*, the instance playing the role of *e* may only be connected to two instances playing the role of *rear.* In addition, the instances playing the *e* and *rear* roles may only be linked if they are roles of the same instance of class *Car.*

In other words, subfigure (ii) asserts additional constraints on the instances of the classes *Wheel* and *Engine*, when they are playing the respective roles within an instance of class *Car.* These constraints are not true for instances of *Wheel* and *Engine* in general. Other wheels and engines may be arbitrarily linked as specified in subfigure (i).

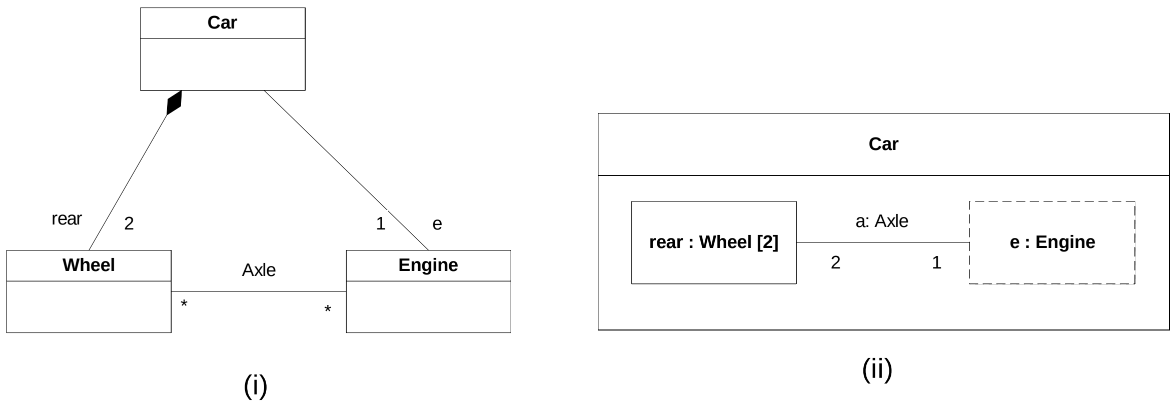


Figure 9.21 - Properties

Notation

A part is shown by graphical nesting of a box symbol with a solid outline representing the part within the symbol representing the containing classifier in a separate compartment. A property specifying an instance that is not owned by composition by the instance of the containing classifier is shown by graphical nesting of a box symbol with a dashed outline.

The contained box symbol has only a name compartment, which contains a string according to the syntax defined in the Notation subsection of “Property (from Kernel, AssociationClasses)” on page 118. Detail may be shown within the box symbol indicating specific values for properties of the type classifier when instances corresponding to the property symbol are created.

Presentation Options

The multiplicity for a property may also be shown as a multiplicity mark in the top right corner of the part box.

A property symbol may be shown containing just a single name (without the colon) in its name string. This implies the definition of an anonymously named class nested within the namespace of the containing class. The part has this anonymous class as its type. Every occurrence of an anonymous class is different from any other occurrence. The anonymously defined class has the properties specified with the part symbol. It is allowed to show compartments defining attributes and operations of the anonymously named class.

Examples

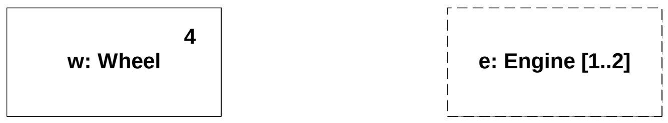


Figure 9.22 - Property examples

Figure 9.22 shows examples of properties. On the left, the property denotes that the containing instance will own four instances of the *Wheel* class by composition. The multiplicity is shown using the presentation option discussed above. The property on the right denotes that the containing instance will reference one or two instances of the *Engine* class. For additional examples, see 9.3.13, “StructuredClassifier (from InternalStructures),” on page 182.

Changes from previous UML

A connectable element used in a collaboration subsumes the concept of ClassifierRole.

9.3.13 StructuredClassifier (from InternalStructures)

Generalizations

- “Classifier (from Kernel, Dependencies, PowerTypes)” on page 48

Description

A structured classifier is an abstract metaclass that represents any classifier whose behavior can be fully or partly described by the collaboration of owned or referenced instances.

Attributes

No additional attributes

Associations

• role: ConnectableElement	References the roles that instances may play in this classifier. (Abstract union; subsets *Classifier.feature*)
• ownedAttribute: Property	References the properties owned by the classifier. (Subsets *StructuredClassifier.role, Classifier.attribute,* and *Namespace.ownedMember*)
• part: Property	References the properties specifying instances that the classifier owns by composition. This association is derived, selecting those owned properties where *isComposite* is *true*.
• ownedConnector: Connector	References the connectors owned by the classifier. (Subsets *Classifier.feature* and *Namespace.ownedMember*)

Constraints

[1] The multiplicities on connected elements must be consistent.

Semantics

The multiplicities on the structural features and connector ends indicate the number of instances (objects and links) that may be created within an instance of the containing classifier, either when the instance of the containing classifier is created, or at a later time. The lower bound of the multiplicity range indicates the number of instances that are created (unless indicated differently by an associated instance specification or an invoked constructor function); the upper bound of the multiplicity range indicates the maximum number of instances that may be created. The slots corresponding to the structural features are initialized with these instances.

The manner of creation of the containing classifier may override the default instantiation. When an instance specification is used to specify the initial instance to be created for a classifier (see “Class” on page 162), the multiplicities of its parts determine the number of initial instances that will be created within that classifier. Initially, there will be as many instances held in slots as indicated by the corresponding multiplicity. Multiplicity ranges on such instance specifications may not contain upper bounds.

All instances corresponding to parts of a structured classifier are destroyed recursively, when an instance of that structured classifier is deleted. The instance is removed from the extent of its classifier, and is itself destroyed.

Semantic Variation Points

The rules for matching the multiplicities of connector ends and those of parts and ports they interconnect are a semantic variation point. Also, the specific topology that results from such multi-connectors will differ from system to system. One possible approach to this is illustrated in Figure 9.23and Figure 9.24.

For each instance playing a role in an internal structure, there will initially be as many links as indicated by the multiplicity of the opposite ends of connectors attached to that role (see “ConnectorEnd” on page 172 for the semantics where no multiplicities are given for an end). If the multiplicities of the ends match the multiplicities of the roles they are attached to (see Figure 9.23 i), the initial configuration that will be created when an instance of the containing classifier is created consists of the set of instances corresponding to the roles (as specified by the multiplicities on the roles) fully connected by links (see the resultant instance, Figure 9.23 ii).

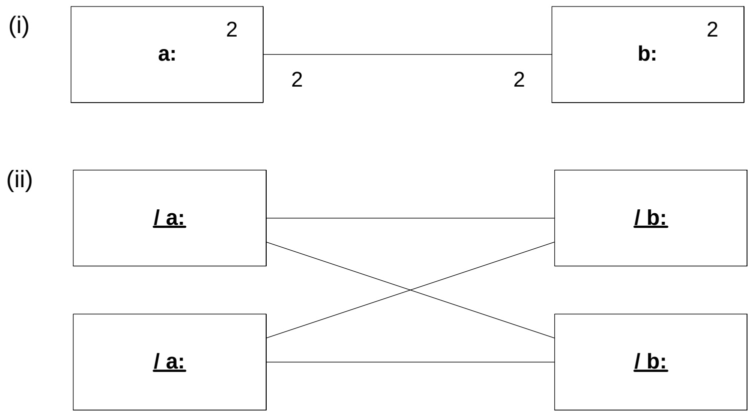


Figure 9.23 - “Star” connector pattern

Multiplicities on connector ends serve to restrict the number of initial links created. Links will be created for each instance playing the connected roles according to their ordering until the minimum connector end multiplicity is reached for both ends of the connector (see the resultant instance, Figure 9.24 ii). In this example, only two links are created, resulting in an array pattern.

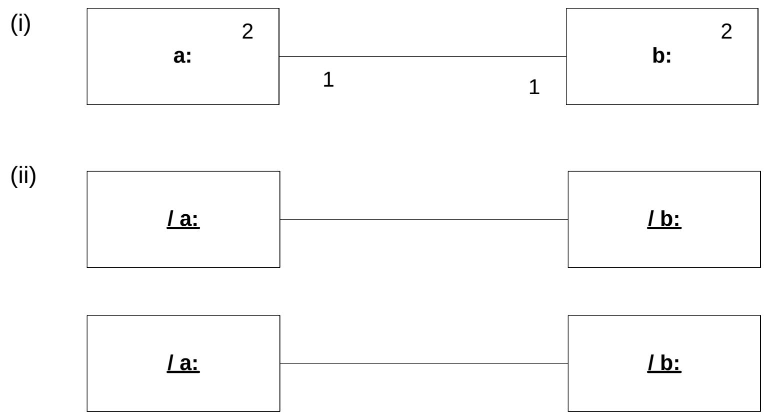


Figure 9.24 - “Array” connector pattern

Notation

The namestring of a role in an instance specification obeys the following syntax:

{<name> [‘/’ <rolename>] | ‘/’ <rolename>} [‘:’ <classifiername> [‘,’ <classifiername>]]*

The name of the instance specification may be followed by the name of the part that the instance plays. The name of the part may only be present if the instance plays a role.

Examples

The following example shows two classes, *Car* and *Wheel*. The *Car* class has four parts, all of type *Wheel*, representing the four wheels of the car. The front wheels and the rear wheels are linked via a connector representing the front and rear axle, respectively. An implicit association is defined as the type of each axle with each end typed by the *Wheel* class. Figure 9.25 specifies that whenever an instance of the *Car* class is created, four instances of the *Wheel* class are created and held by composition within the car instance. In addition, one link each is created between the front wheel instances and the rear wheel instances.

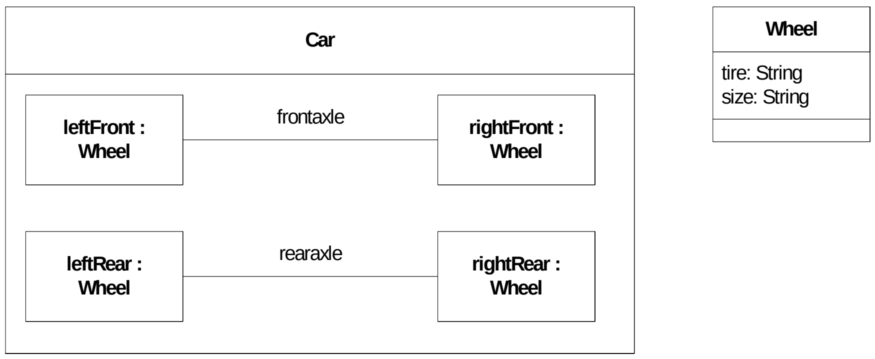


Figure 9.25 - Connectors and parts in a structure diagram

Figure 9.26 specifies an equivalent system, but relies on multiplicities to show the replication of the wheel and axle arrangement. This diagram specifies that there will be two instances of the left wheel and two instances of the right wheel, with each matching instance connected by a link deriving from the connector representing the axle. As specified by the multiplicities, no additional instances of the *Wheel* class can be added as left or right parts for a *Car* instance.

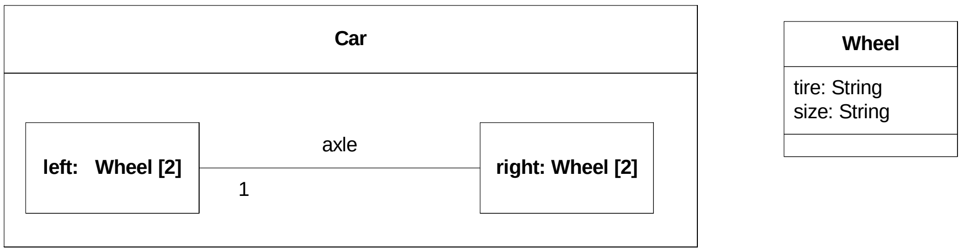


Figure 9.26 - Connectors and parts in a structure diagram using multiplicities

Figure 9.27 shows an instance of the *Car* class (as specified in Figure 9.25). It describes the internal structure of the *Car* that it creates and how the four contained instances of *Wheel* will be initialized. In this case, every instance of *Wheel* will have the predefined size and use the brand of tire as specified. The left wheel instances are given names, and all wheel instances are shown as playing the respective roles. The types of the wheel instances have been suppressed.

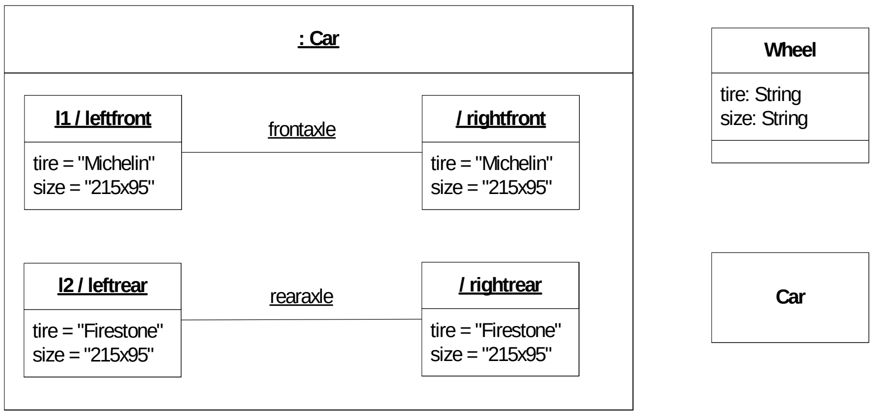


Figure 9.27 - A instance of the Car class

Finally, Figure 9.28 shows a constructor for the *Car* class (see “Class” on page 162). This constructor takes a parameter *brand* of type *String*. It describes the internal structure of the *Car* that it creates and how the four contained instances of *Wheel* will be initialized. In this case, every instance of *Wheel* will have the predefined size and use the brand of tire passed as parameter. The left wheel instances are given names, and all wheel instances are shown as playing the parts. The types of the wheel instances have been suppressed.

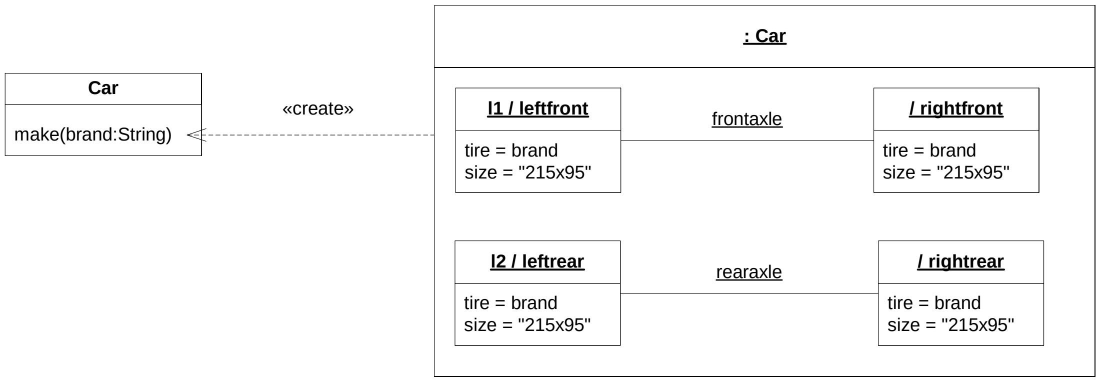


Figure 9.28 - A constructor for the Car class

9.3.14 Trigger (from InvocationActions)

Generalizations

- "Trigger (from Communications)" on page 441 *(merge increment)*

Description

A trigger specification may be qualified by the port on which the event occurred.

Associations

• port: Port [*]	Specifies the ports at which a communication that caused an event may have arrived.

Semantics

Specifying one or more ports for an event implies that the event triggers the execution of an associated behavior only if the event was received via one of the specified ports.

Notation

The ports of a trigger are specified following a trigger signature by a list of port names separated by comma, preceded by the keyword «from»:

*'«from»' <port-name> [',' <port-name>]**

9.3.15 Variable (from StructuredActivities)

Generalizations

- "Variable (from StructuredActivities)" on page 401 *(merge increment)*

Description

A variable is considered a connectable element.

Semantics

Extends variable to specialize connectable element.

9.4 Diagrams

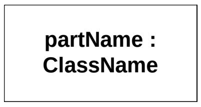
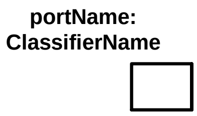
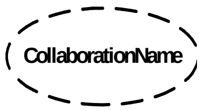
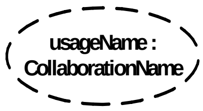
Composite structure diagram

A composite structure diagram depicts the internal structure of a classifier, as well as the use of a collaboration in a collaboration use.

Graphical nodes

Additional graphical nodes that can be included in composite structure diagrams are shown in Table 9.1.

Table 9.1 - Graphic nodes included in composite structure diagrams

Node Type	Notation	Reference
Part		See “Property (from InternalStructures)” on page 179.
Port		See “Ports” on page 175. A port may appear either on a contained part representing a port on that part, or on the boundary of the class diagram, representing a port on the represented classifier itself. The optional *ClassifierName* is only used if it is desired to specify a class of an object that implements the port.
Collaboration		See “Collaboration” on page 164.
CollaborationUse		See “CollaborationUse (from Collaborations)” on page 166.

Graphical paths

Additional graphical paths that can be included in composite structure diagrams are shown in Table 9.2.

Table 9.2 - Graphic nodes included in composite structure diagrams

Path Type	**Notation**	**Reference**
Connector	————————	See “Connector” on page 170.
Role binding	- - - - - - - - - - - -	See “CollaborationUse (from Collaborations)” on page 166.

Structure diagram

All graphical nodes and paths shown on composite structure diagrams can also be shown on other structure diagrams.