The UML4SOA Profile

This document describes the UML4SOA profile for behavioural specifications of services.

Metadata

- Version: 2.1
- Date: 2009-07-24
- Authors: Philip Mayer, Nora Koch, Andreas Schroeder

Change Log

Version	Date	Notable Changes
1.0	September 30, 2007	Initial version.
1.1	July 08, 2008	Event handling.
1.2	September 15, 2008	Complete metamodel. Full transformation support.
1.3	December 1, 2008	Protocol state machines.
2.0	June 7, 2009	Data handling.
2.1	July 24, 2009	Revised metamodel.

License

Acknowledgements

This work has been partially supported by the EC project SENSORIA, IST-2005-016004.

Introduction to UML4SOA

UML4SOA is a profile for specifying behavioural aspects of service-oriented architectures (SOAs). In particular, we focus on service orchestrations, i.e. compositions of services, by means of an orchestration workflow. An orchestration, in turn, is another service to be used externally, or in other orchestrations.

We have selected UML2 activity diagrams as the base for modelling such workflows, and UML2 state machines for modelling their externally visible behaviour with regard to a certain partner. We extend both notations by SOA-specific stereotypes, thereby enabling developers to model SOA orchestrations in a high-level fashion. The extension is minimal, i.e. we use existing UML2 elements wherever possible, only extending the UML2 where we require additional semantics, or if it adds to the overall clarity of the diagrams.

The UML4SOA profile has been developed within the SENSORIA project, where it has been used as input to several case studies. There are also model transformation tools available for converting UML4SOA diagrams to BPEL/WSDL, Java, and Jolie. Finally, UML4SOA diagrams enjoy formal analysis support through the SENSORIA Development Environment (SDE) and integrated tools.

UML4SOA complements the SoaML profile that focuses on the structural aspects of SOAS and can be used in combination with other UML profiles, such as the MARTE profile, which has been used for performance analysis within the scope of the SENSORIA project.

Notes on Design Decisions

Diagrams

UML4SOA is a profile for specifying *behavioural* properties of services. We therefore focus on two diagram types of the UML which offer us the ability to specify behaviour:

- **Activity Diagrams**. We use activities and actions to create the platform-independent "implementation" of a service. In particular, this mechanism can be used for creating service orchestrations, i.e. a workflow which combines several services into a new one.
- **Protocol State Machine Diagrams**. We use PSMs for denoting the externally visible behaviour of services, as seen from a partner.

UML4SOA attempts to extend the UML2 in a lightweight way. We only define new metaclasses and stereotypes where necessary, allowing users to leverage their existing knowledge of the UML2 as far as possible.

Service Interactions

One of the most important points of behavioural service specification is the specification of interactions. We employ UML2 operations, defined in interfaces or classes, as the basic interaction mechanism, and re-use existing UML2 metaclasses for invoking and receiving functionality. In particular,

- a **service invocation** is a specialization of a CallOperationAction,
- a **service reply** is a specialization of a ReplyAction,
- a **service receive** is a specialization of an AcceptCallAction.

We do not assume a mode of communication between services – UML4SOA allows both message passing and RPC styles. However, we recommend message passing due to better maintainability of the resulting system.

Partners

Another important point of SOA-based architectures is the notion of service providers and requesters. Operations invoked on external services, or operation calls received from external services usually follow an abstract interface specification. This specification is useful for reaching an agreement on the common interface, and is used by UML through operations of classes and interfaces. In a SOA context, however, we need something more: The information about a service binding, i.e. the target or source we are talking to in a specific instance of a service implementation.

For this, we introduce the notion of a **Partner** of a service. We use specializations of UML2 ports for denoting a partner: **service point** and **request point**.

Structural Aspects

Focussing on behavioural aspects of services, we employ external profiles for structural properties and non-functional requirements. In particular, we employ stereotypes from the SoaML profile for structural properties of services:

- **MessageType** for denoting value types to be used as operation arguments and return types.
- **ServicePoint / RequestPoint** (specializations of Port) for denoting connection points through which service provider provides or service requester requests a service. In the text we use “partner” for either service point or request point.

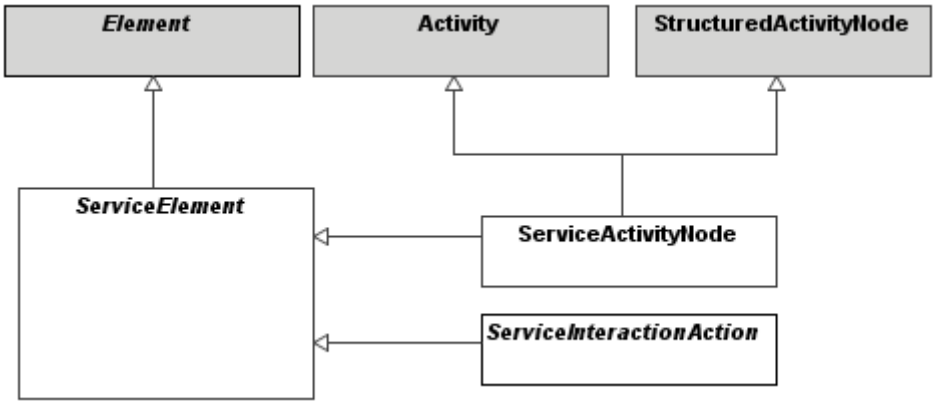
We recommend using SoaML and the above stereotypes for modelling structural parts of a SOA, however you are of course free to exchange these stereotypes with those of another profile.

Contents

The UML4SOA Metamodel

Core

Overview



Metaclass	Stereotype	UML Base Classes	Description	Used In
ServiceElement	None	Element	A ServiceElement serves the purpose of attaching compensation, event, and exception handlers to ServiceActivityNode and service interaction actions.	None
ServiceActivityNode	«serviceActivity»	StructuredActivityNode ServiceElement Activity	A ServiceActivityNode groups service-related actions for compensation, exception handling, and events.	Activity Diagram
ServiceInteractionAction	None	ServiceElement	ServiceInteractionAction is the common base class of all service interaction actions, which have associated with them a LinkPin.	None

ServiceElement (abstract)

Description

An ServiceElement is an abstraction of a ServiceActivityNode and a ServiceInteractionAction. It serves the purpose of attaching compensation, event, and exception handlers to ServiceActivityNodes and service interaction actions, such as send and reply actions.

Generalizes

- Element

Associations

- eventHandler : EventEdge[0...*]
  The event handlers attached to this ServiceElement.
- compensationHandler : CompensationEdge[1...1]
  The compensation handlers attached to this ServiceElement.

Semantics

None.

Constraints

a) If a compensation handler is specified, the target element MUST have this element as the compensatedElement.

b) If event handlers are specified, each of them MUST have this element as the eventBaseElement.

Notation

No visual representation.

ServiceActivityNode

Description

A ServiceActivityNode represents either

a) a special activity for service behaviour or

b) a grouping element for actions and ServiceActivityNode (nesting)

A ServiceActivityNode may have control edges connected to it, and pins when merged with CompleteActivities or on specializations in CompleteStructuredActivities. The execution of any embedded actions may not begin until the ServiceActivityNode has received its object and control tokens. The availability of output tokens from the structured activity node does not occur until all embedded actions (note that this does not include handlers) have completed execution.

In addition to both Activity and StructuredActivityNode, a ServiceActivityNode node may have attached compensation and event handlers.

Generalizes

- StructuredActivityNode
- ServiceElement
- Activity

Associations

None.

Semantics

The semantics follow the ones defined in StructuredActivityNode. In addition, we define the following semantics for compensation and event handlers. When the ServiceActivityNode has a compensation handler attached, this handler is installed when the ServiceActivityNode completes successfully. An event handler may be executed at any time during the execution of the ServiceActivityNode, running in parallel to the ServiceActivityNode. Event handlers may be invoked multiple times, and run at the same time.

A top-level service activity is attached to a class which represents the service or group of services for which this is the behaviour.

Constraints

No additional constraints.

Notation

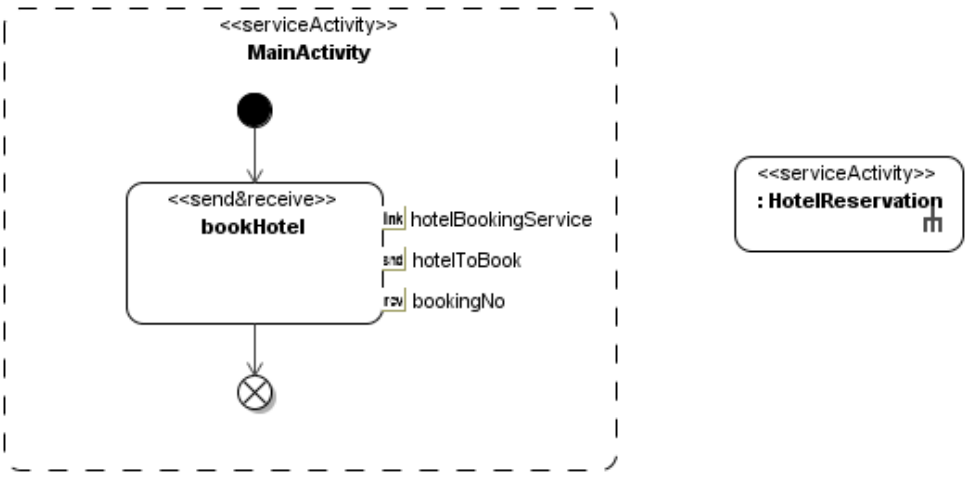
As a ServiceActivityNode comes in two versions, there are two notations.

First, the "StructuredActivityNode" notation: the ServiceActivityNode is notated with a dashed round cornered rectangle enclosing its nodes and edges, with the keyword «serviceActivity» at the top. Also see children of StructuredActivityNode.

Second, the "Activity" notation: Same notation as for activities apply, however the «serviceActivity» stereotype must be used.

Examples

The following examples show the use of a service activity. The activity on the left contains one action with the stereotype <>, which in turn contains three pins. The serviceActivity is annotated with the <> stereotype, and carries a name ("MainActivity").



ServiceInteractionAction (abstract)

Description

ServiceInteractionAction is the common base class of all service interaction actions, which have associated with them a LinkPin.

Generalizes

- ServiceElement

Associations

- partner : LinkPin[1...1]
  Specifies the partner of this ServiceInteractionAction. In case of a ServiceSendAction, this link subsets target.

Semantics

The interaction is linked to a partner of the behaviour via the link pin. The operation is specified in the actions themselves.

Constraints

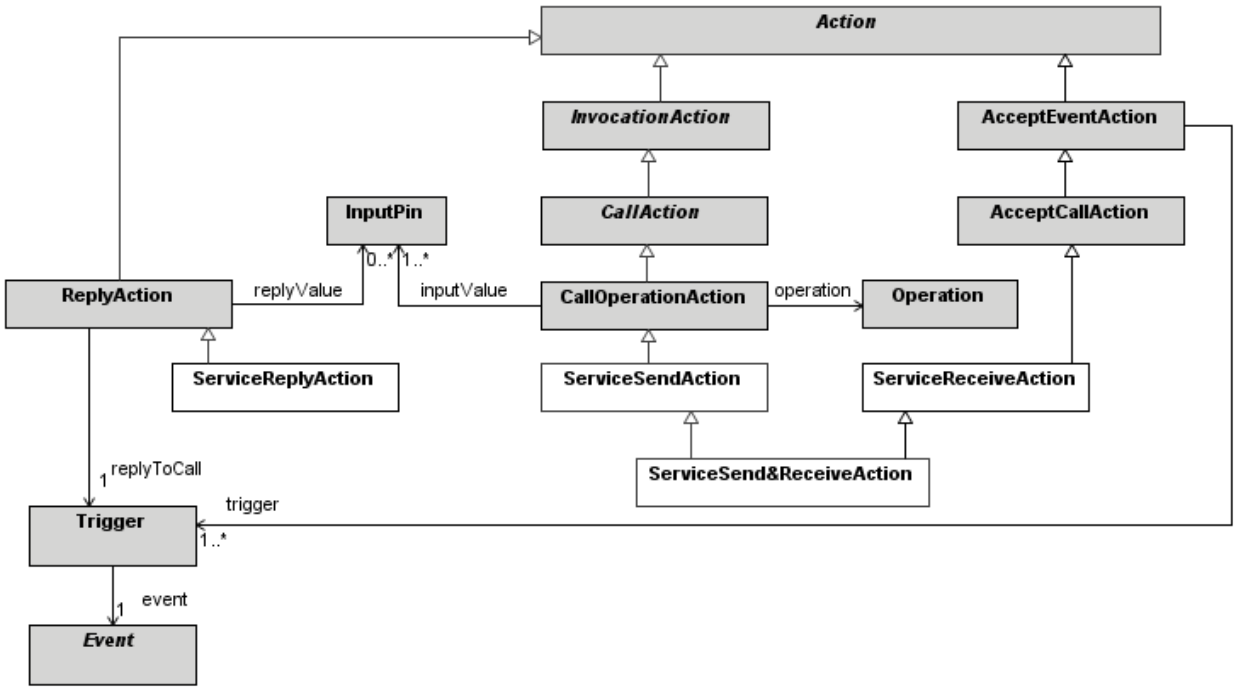
No additional constraints.

Notation

No notation.

Actions

Overview



Metaclass	Stereotype	UML Base Classes	Description	Used In
ServiceSendAction	«send»	CallOperationAction ServiceInteractionAction	A ServiceSendAction sends invokes an operation of a partner asynchronously.	Activity Diagram
ServiceReceiveAction	«receive»	ServiceInteractionAction AcceptCallAction	ServiceReceiveAction is used to receive an operation call from an external partner.	Activity Diagram
ServiceReplyAction	«reply»	ReplyAction ServiceInteractionAction	The ServiceReplyAction allows specification of data to be sent out in reply to a ServiceReceiveAction.	Activity Diagram
ServiceSend&ReceiveAction	«send&receive»	ServiceSendAction ServiceReceiveAction	A ServiceSend&ReceiveAction action is a complete, synchronous operation call execution with a partner.	Activity Diagram

ServiceSendAction

Description

ServiceSendAction is an action that invokes an operation of a target service asynchronously, i.e. without waiting for a reply. The argument values are data to be transmitted as parameters of the operation call. There is no return value.

ServiceSendAction inherits argument from InvocationAction. We restrict this to SendPins which contain the data to be sent.

Generalizes

- CallOperationAction
- ServiceInteractionAction

Associations

- (inherited association from supertype) : SendPin[0...*]
  {subsets argument}

Semantics

The ServiceSendAction sends out the data without waiting for a response. CallOperationAction contains the Operation directly.

Constraints

a) ServiceSendAction constrains argument (inherited from InvocationAction) to pins of type SendPin.

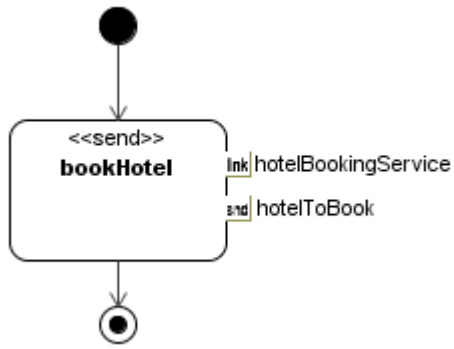
b) target is constrained to instances of LinkPin.

Notation

A ServiceSendAction is stereotyped with «send». The operation name is given inside the action body.

Examples

This example shows a send (without receive). An operation call is sent to the partner hotelBookingService (specified in the lnk pin). The data usage itself is stored in the variable hotelToBook (specified in the send pin). There is no return value.



ServiceReceiveAction

Description

ServiceReceiveAction is an accept call action representing the receipt of an operation call from an external partner. No answer is given to the external partner.

Generalizes

- ServiceInteractionAction
- AcceptCallAction

Associations

- (inherited association from supertype) : ReceivePin[0...*]
  {subsets result}

Semantics

ServiceReceiveAction blocks until the message is received. It requires a trigger (with a CallEvent event), which contains the operation. ReceivePins must be given, which contain the variables in which the incoming data is stored.

Constraints

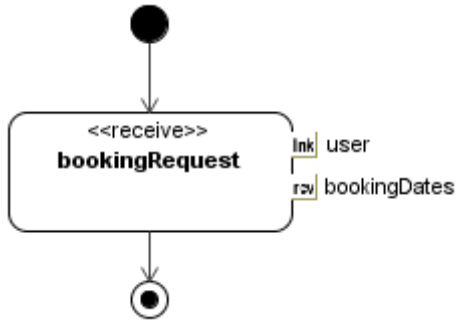
The result pins MUST be ReceivePins. This ensures that the data received has value types. The trigger must be a CallEvent.

Notation

A ServiceReceiveAction is stereotyped with «receive». The operation name (from trigger->CallEvent) is given inside the action body.

Examples

This example shows a receive. A message is received from a partner (called "user", specified in the lnk pin). The message itself is stored in the variable bookingDates (specified in the rcv pin). The message type called bookingRequest.



ServiceReplyAction

Description

ServiceReplyAction is an action that accepts a return value and a value containing return information produced by a previous ServiceReceiveAction. The reply action returns the values to the request point of the previous call, completing execution of the call.

ServiceReplyAction is a specialized version of ReplyAction for the service-oriented context. The inherited attribute replyValue is subset to point to instances of SendPin, instead of a generic input pin, thereby ensuring the data can be interpreted as value data. Thus, a ServiceReplyAction sends back data to a request point for which previous data was received.

Generalizes

- ReplyAction
- ServiceInteractionAction

Associations

- (inherited association from supertype) : SendPin[0...*]
  {subsets replyValue}

Semantics

The ServiceReplyAction completes a call from an external source.

Constraints

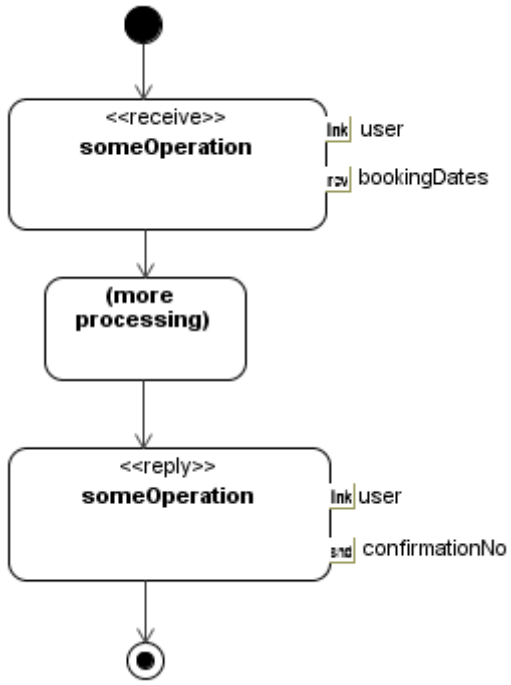
The replyValue pins MUST be of type SendPin.

Notation

A ServiceReplyAction is stereotyped with «reply». The operation name is given inside the action body (corresponding to the operation inside the attached Trigger).

Examples

This example shows a reply. A reply is always an answer to a previous receive, and carries the same partner and operation name as the receive. In this example, a bookingRequest is received from partner "user", and the message is stored in the variable bookingDates. Now, some processing takes place. After that is finished, the data in the variable bookingInformation is sent as a reply to the "user" partner.



ServiceSend&ReceiveAction

Description

A ServiceSend&ReceiveAction action is a complete synchronous operation call execution with a partner. Some data (stored in the SendPins) is sent, then the action waits for data to be sent back, which is stored in the ReceivePins.

Generalizes

- ServiceSendAction
- ServiceReceiveAction

Associations

None.

Semantics

See ServiceSendAction and ServiceReceiveAction.

Constraints

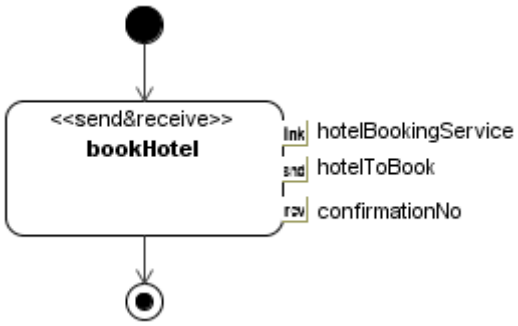
No additional constraints.

Notation

A ServiceSend&ReceiveAction is stereotyped with «send&receive». The operation name is given inside the action body.

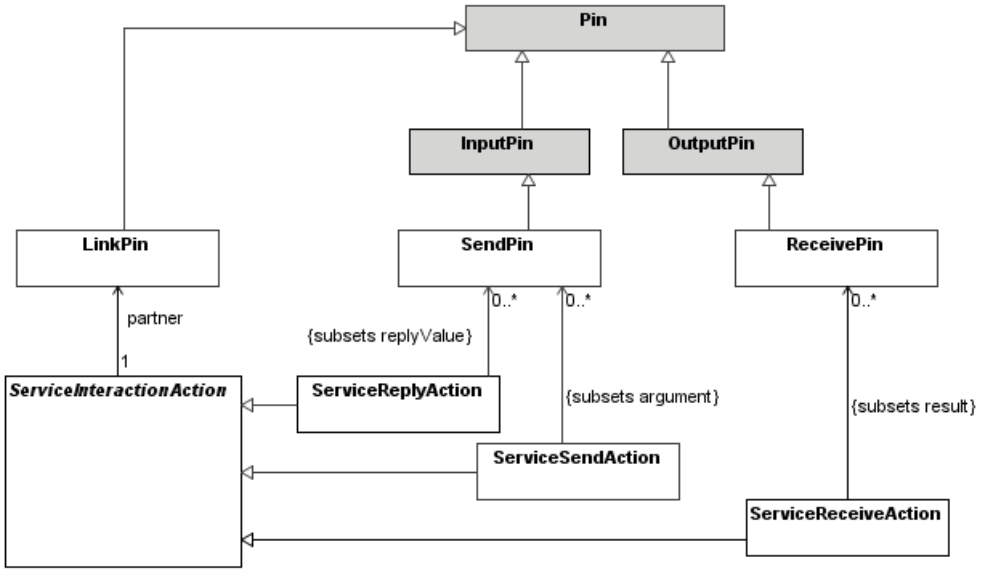
Examples

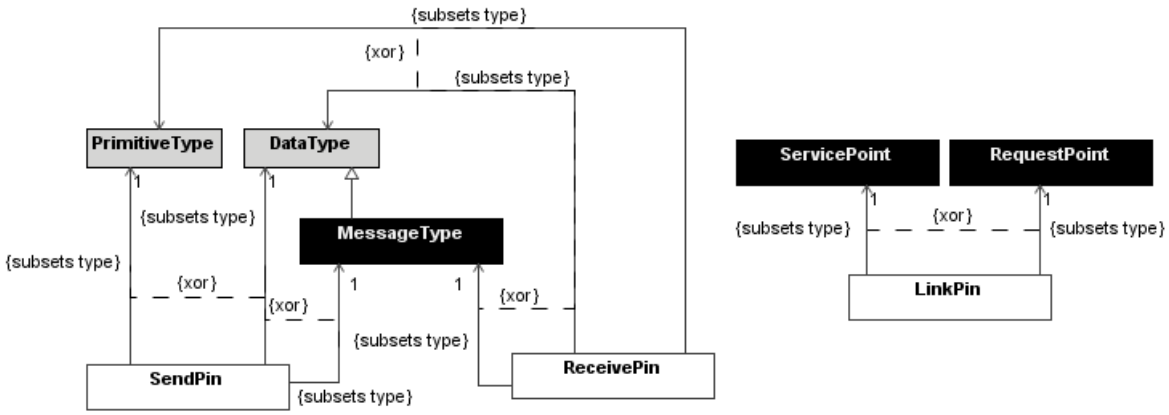
This example shows a send&receive. An operation is invoked on the partner hotelBookingService (specified in the lnk pin). The data itself is stored in the variable hotelToBook (specified in the snd pin) and must be initialized before the action. The return value from the service is stored in the variable confirmationNo (specified in the rcv pin).



Pins

Overview





Metaclass	Stereotype	UML Base Classes	Description	Used In
LinkPin	«lnk»	Pin	A LinkPin is used to indicate the partner service for the service interaction.	Activity Diagram
SendPin	«snd»	InputPin	A SendPin is used in send actions to denote the data to be sent to an external service.	Activity Diagram
ReceivePin	«rcv»	OutputPin	A ReceivePin is used in receive actions to denote the data to be received from an external service.	Activity Diagram

LinkPin

Description

A LinkPin is used to indicate the partner service for the service interaction.

Generalizes

- Pin

Associations

- (inherited association from supertype) : ServicePoint[1...1] {subsets type}
- (inherited association from supertype) : RequestPoint[1...1] {subsets type}

Semantics

The partner must be bound before execution. Note that a partner can be a service requester, or a service provider.

Constraints

a) The type of this pin is either RequestPoint or ServicePoint (not both).

b) The RequestPoint or ServicePoint must be a port attached to the class which the root ServiceActivityNode of this behavioural specification belongs to.

Notation

The pin is stereotyped with «link», or with the corresponding icon ("lnk"). The parter name is specified along with the pin.

Examples

This example shows the use of a LinkPin. In all partner-related actions, for example in this send&receive, the service partner must be specified. In this case, the partner is called "bookingService". Note that a partner can be a service requester, or a service provider to this activity. In this case, as this is a send&receive, bookingService is a provider. In case of a receive or receive&send, it is a requester.

SendPin

Description

A SendPin is used in send actions to denote the data to be sent to an external service.

Generalizes

- InputPin

Associations

- (inherited association from supertype) : MessageType[1...1]
  {subsets type}
- (inherited association from supertype) : PrimitiveType[1...1]
  {subsets type}
- (inherited association from supertype) : DataType[1...1]
  {subsets type}

Semantics

A SendPin must either a) specify a UML variable holding the data, or b) a constant value.

Constraints

The type must be a subtype of either MessageType, PrimitiveType, or DataType. Also, the SendPin must have the correct type for the operation and partner invoked.

Notation

The SendPin must be stereotyped with «snd», or with the corresponding icon ("snd"). Furthermore, it needs to be annotated with the information about data to be sent (either variables, or constants).

Examples

This example shows the use of a SendPin. The send pin contains the message to be sent to a partner. In most cases, this is a variable, but a constant value may be specified as well.

ReceivePin

Description

A ReceivePin is used in receive actions to denote the data to be received from an external service.

Generalizes

- OutputPin

Associations

- (inherited association from supertype) : MessageType[1...1]
  {subsets type}
- (inherited association from supertype) : PrimitiveType[1...1]
  {subsets type}
- (inherited association from supertype) : DataType[1...1]
  {subsets type}

Semantics

A ReceivePin must specify a UML Variable holding the data. The variable must be typed with either a) a MessageType or b) a ServiceValueType.

Constraints

The type must be a subtype of either MessageType, PrimitiveType, or DataType. Also, the ReceivePin must have the correct type for the operation and partner invoked.

Notation

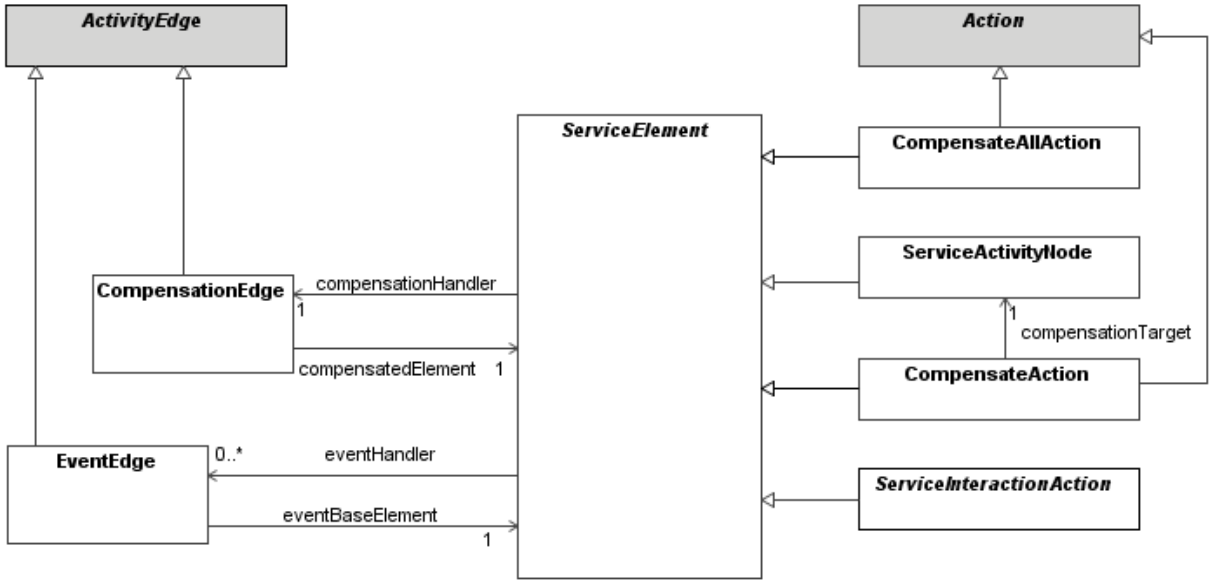
The ReceivePin must be stereotyped with «rcv», or with the corresponding icon ("rcv"). Furthermore, it needs to be annotated with the information about data to be received (variable name).

Examples

This example shows the use of a ReceivePin. The receive pin contains the target where the message received will be stored, i.e. the variable name.

Compensation&Events

Overview



Metaclass	Stereotype	UML Base Classes	Description	Used In
CompensationEdge	«compensation»	ActivityEdge	CompensationEdge is an edge connecting a ServiceElement to be compensated with the one specifying a compensation.	Activity Diagram
EventEdge	«event»	ActivityEdge	EventEdge is an edge connecting event handlers with a ServiceElement during which the event may occur.	Activity Diagram
CompensateAction	«compensate»	Action ServiceElement	The CompensateAction invokes the compensation handler for a particular ServiceActivityNode.	Activity Diagram
CompensateAllAction	«compensateAll»	Action ServiceElement	The CompensateAllAction invokes all installed compensation handlers which are nested in the current ServiceActivityNode.	Activity Diagram

CompensationEdge

Description

CompensationEdge is an edge connecting a ServiceElement to be compensated with the one specifying a compensation.

Generalizes

- ActivityEdge

Associations

- compensatedElement : ServiceElement[1...1]
  The element to be compensated.

Semantics

Execution of a compensation handler is triggered with a CompensateAction or a CompensateAllAction. Exceptions thrown during a compensation handler must be handled in the invoking ServiceActivityNode, or in a handler attached to the compensation handler.

Constraints

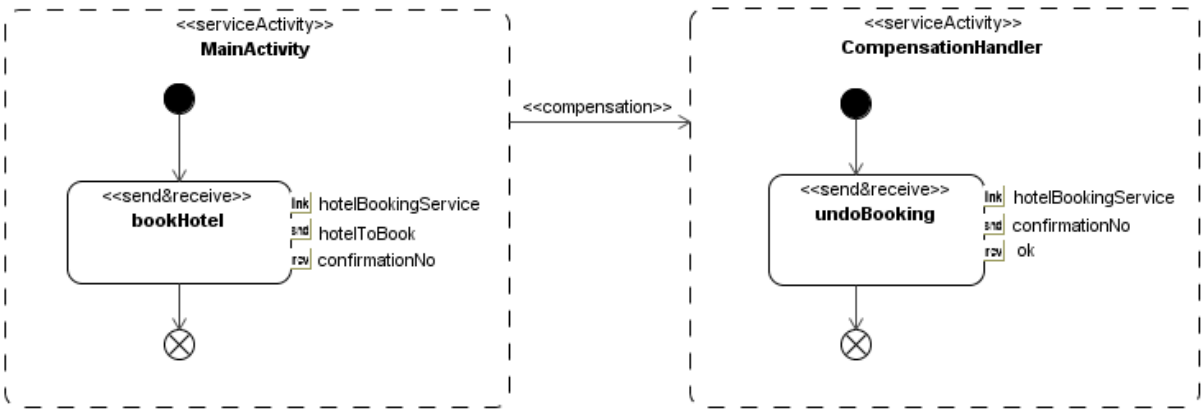
The compensatedElement MUST have this element as a compensationHandler.

Notation

The edge is annotated with the stereotype «compensation».

Examples

This example shows the use of a <>-typed compensation handler. A (normal) <> performs a booking. Later on, if the booking needs to be compensated, the compensation handler is invoked to undo the work.



EventEdge

Description

EventEdge is an edge connecting event handlers with a ServiceElement during which the event may occur.

Generalizes

- ActivityEdge

Associations

- eventBaseElement : ServiceElement[1...1]
  The element during which this event may occur.

Semantics

Execution of a event handler is triggered externally by means of a message, or a timed event.

Constraints

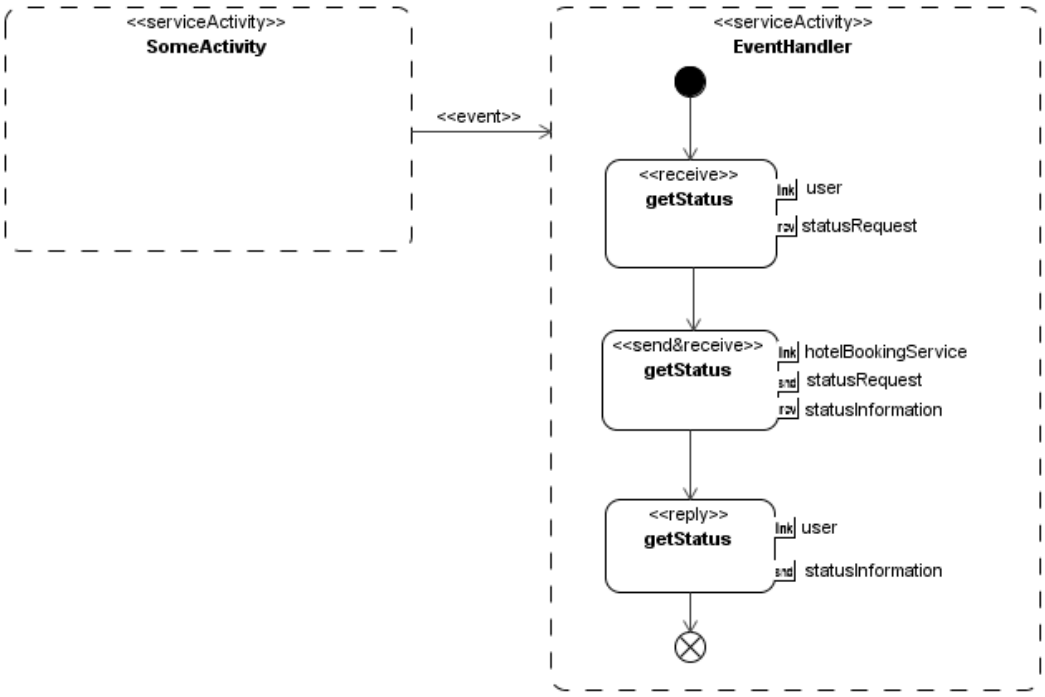
The eventBaseElement MUST have this element as an eventHandler.

Notation

The edge is annotated with the stereotype «event».

Examples

This example shows the use of a <>-typed event handler. The event handler is installed in parallel to SomeActivity. If a message is received from a client (getStatus()), a partner is invoked to retrieve the statusInformation, which is then returned to the client. This happens in parallel to SomeActivity.



CompensateAction

Description

The CompensateAction invokes the compensation handler for a particular ServiceActivityNode.

Generalizes

- Action
- ServiceElement

Associations

- compensationTarget : ServiceActivityNode[1...1]
  The ServiceActivityNode to be compensated.

Semantics

A CompensateAction may only be invoked from an exception or compensation handler. The referenced ServiceActivityNode must be successfully completed. After the compensation handler has been executed, the execution resumes normally.

Constraints

a) The CompensateAction may ONLY be used within a compensation or exception handler.

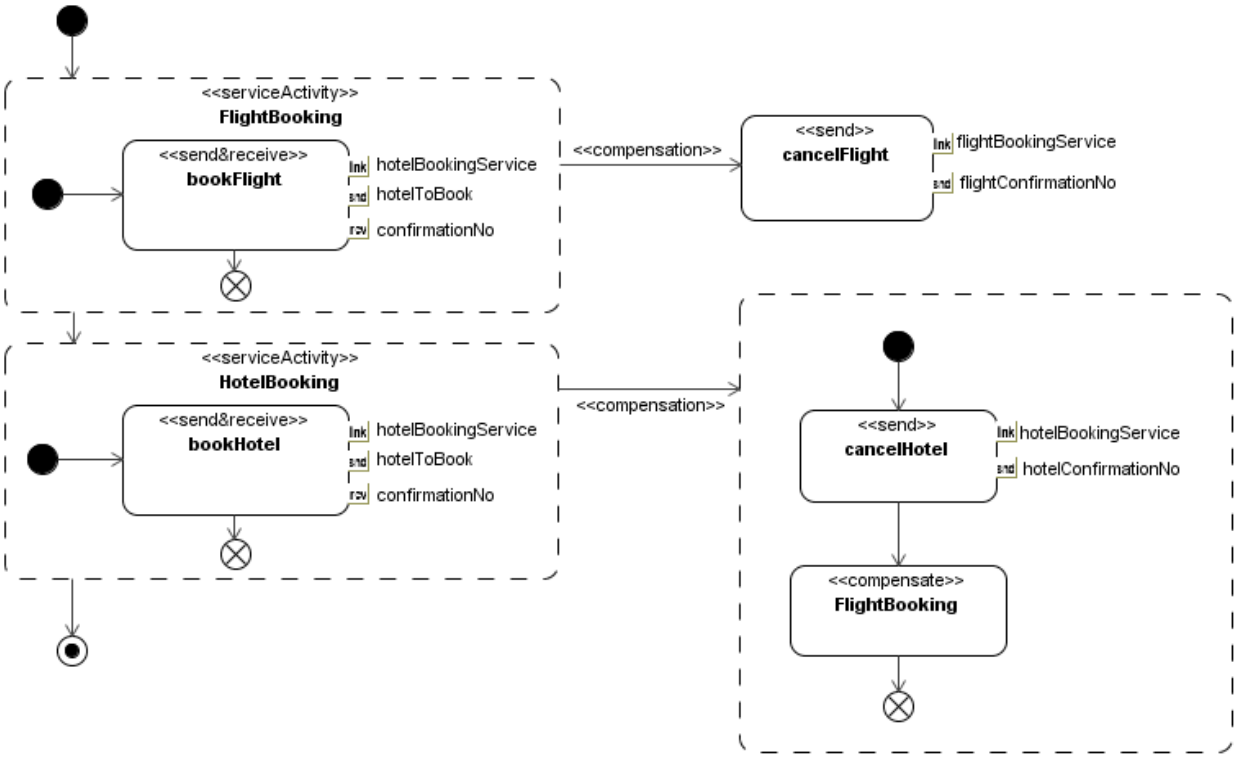
b) The compensationTarget MUST be a ServiceActivityNode which has a compensation handler, and that ServiceActivityNode MUST be nested within the ServiceActivityNode in which the compensation action is invoked.

Notation

Annotation with stereotype «compensate». The target name is given inside the action.

Examples

This example shows the use of the compensate action. In this example, we assume that the compensation handler of FlightBooking is not called from the outside, but instead the compensation handler of HotelBooking also compensates the FlightBooking.



CompensateAllAction

Description

The CompensateAllAction invokes all installed compensation handlers which are nested in the current ServiceActivityNode.

Generalizes

- Action
- ServiceElement

Associations

None.

Semantics

A CompensateAllAction may only be invoked from an exception or compensation handler. After the compensation handlers have been executed, the execution resumes normally.

Constraints

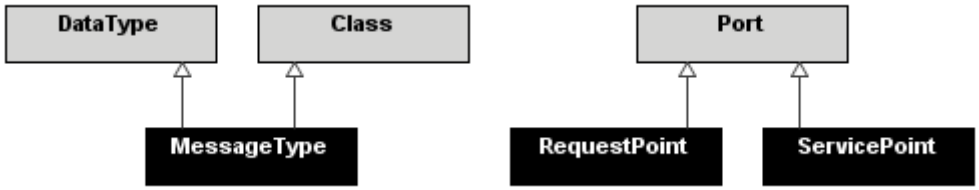
The CompensateAllAction may ONLY be used within a compensation or exception handler.

Notation

Annotation with stereotype «compensateAll». The target name is given inside the action.

Extensions

Overview



Metaclass	Stereotype	UML Base Classes	Description	Used In
MessageType	None	Class DataType	A message type is a class (without behaviour) or a datatype whose instances are intended to be transmitted as a messages between services.	Class Diagram
RequestPoint	None	Port	A request point is a port which is used by a participant to request services through interfaces.	Class Diagram
ServicePoint	None	Port	A service point is a port which is used by a participant to provide services through interfaces.	Class Diagram

MessageType

Description

A message type is a class (without behaviour) or a datatype whose instances are intended to be transmitted as a messages between services.

Generalizes

- Class
- DataType

Associations

None.

Semantics

This class is a placeholder for a message type. In a concrete behavioural specification, use a stereotype from a profile for structural service specifications.

Constraints

No additional constraints.

Notation

None.

RequestPoint

Description

A request point is a port which is used by a participant to request services through interfaces.

Generalizes

- Port

Associations

None.

Semantics

None.

Constraints

No additional constraints.

Notation

None.

ServicePoint

Description

A service point is a port which is used by a participant to provide services through interfaces.

Generalizes

- Port

Associations

None.

Semantics

None.

Constraints

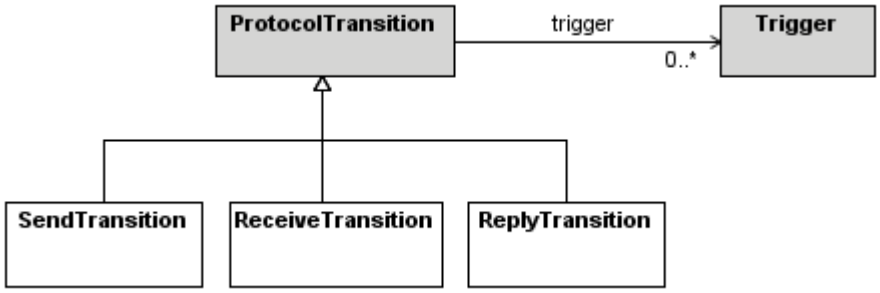
No additional constraints.

Notation

None.

Protocol State Machine

Overview



Metaclass	Stereotype	UML Base Classes	Description	Used In
ReceiveTransition	None	ProtocolTransition	A specialized transition indicating a message receive.	Protocol State Machine Diagram
ReplyTransition	None	ProtocolTransition	A specialized transition indicating a message send.	Protocol State Machine Diagram
SendTransition	None	ProtocolTransition	A specialized transition indicating a message send.	Protocol State Machine Diagram

ReceiveTransition

Description

A specialized transition indicating a message receive.

Generalizes

- ProtocolTransition

Associations

None.

Semantics

No additional semantics.

Constraints

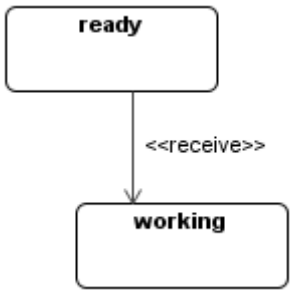
The trigger of this transition must be a ReceiveOperationEvent.

Notation

Annotation with stereotype «receive».

Examples

This is an example for using a receive transition.



ReplyTransition

Description

A specialized transition indicating a message send.

Generalizes

- ProtocolTransition

Associations

None.

Semantics

No additional semantics.

Constraints

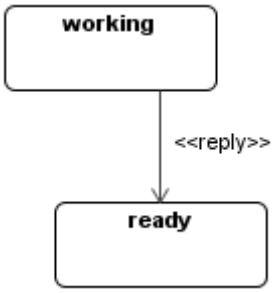
The trigger of this transition must be a SendOperationEvent.

Notation

Annotation with stereotype «reply».

Examples

This is an example for using a reply transition.



SendTransition

Description

A specialized transition indicating a message send.

Generalizes

- ProtocolTransition

Associations

None.

Semantics

No additional semantics.

Constraints

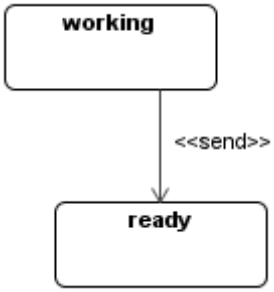
The trigger of this transition must be a SendOperationEvent.

Notation

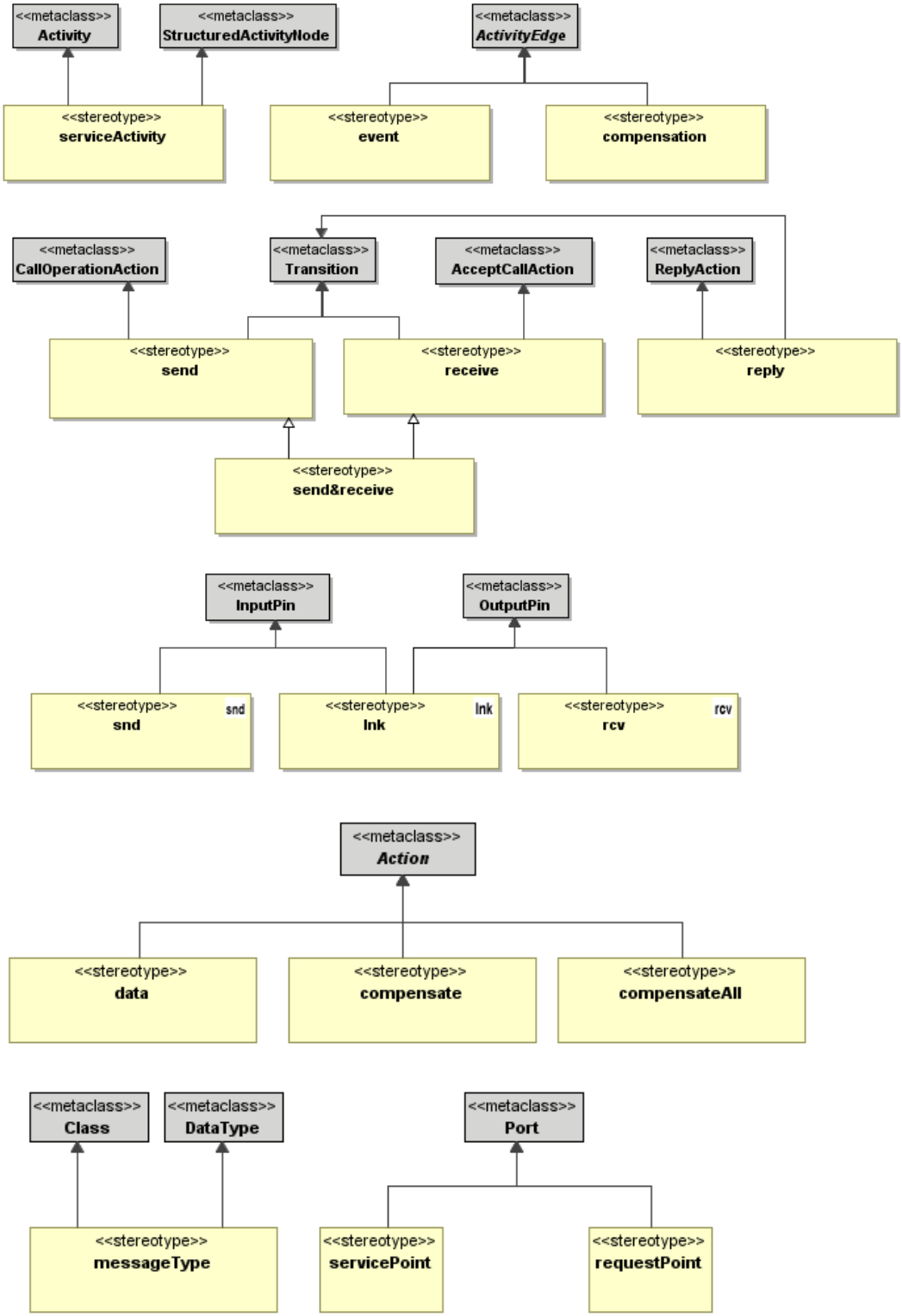
Annotation with stereotype «send».

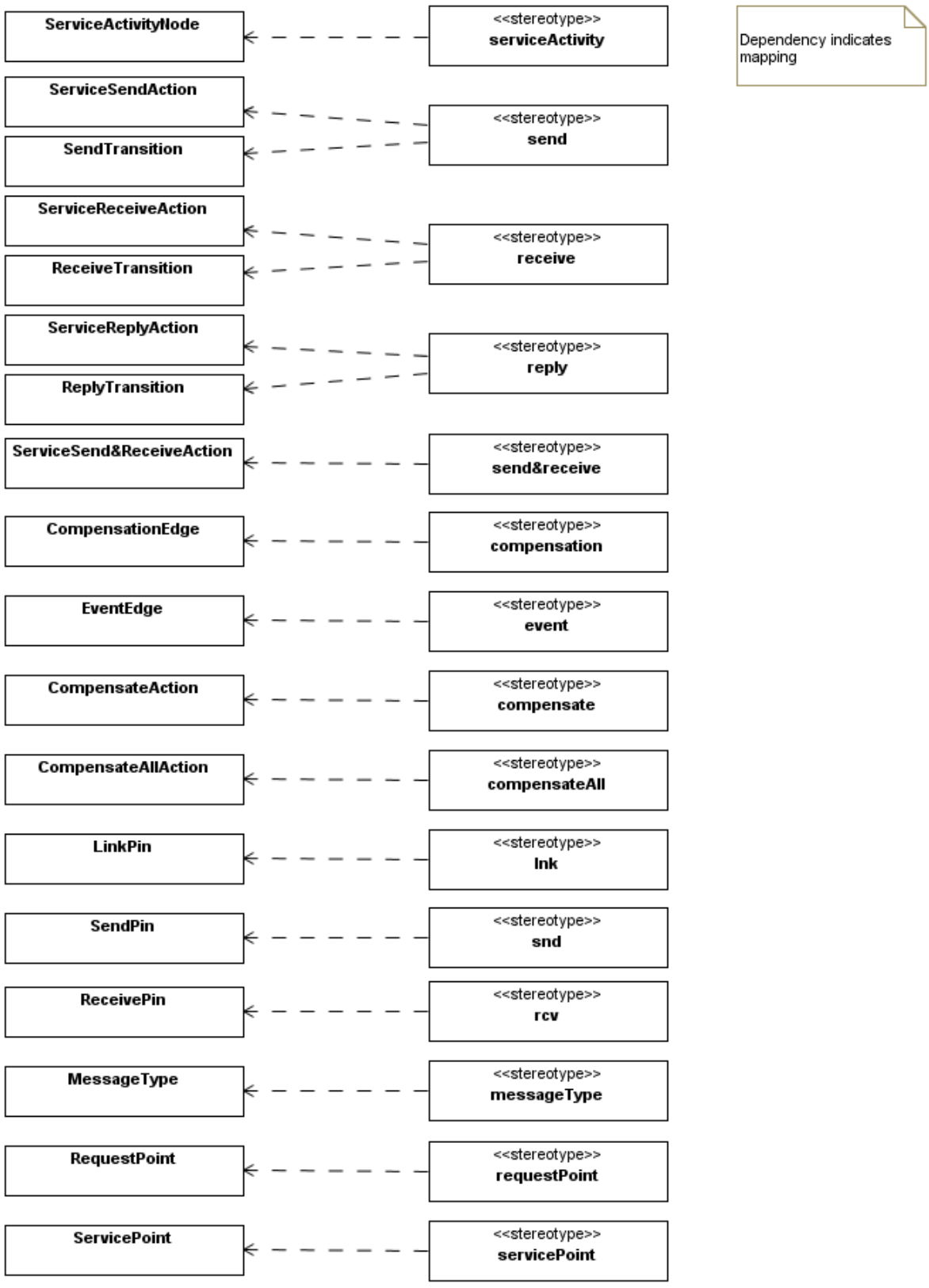
Examples

This is an example for using a send transition.



Stereotypes




ServiceActivityNode
<<stereotype>>
serviceActivity
Dependency indicates mapping
ServiceSendAction
SendTransition
<<stereotype>>
send
ServiceReceiveAction
ReceiveTransition
<<stereotype>>
receive
ServiceReplyAction
ReplyTransition
<<stereotype>>
reply
ServiceSend&ReceiveAction
<<stereotype>>
send&receive
CompensationEdge
<<stereotype>>
compensation
EventEdge
<<stereotype>>
event
CompensateAction
<<stereotype>>
compensate
CompensateAllAction
<<stereotype>>
compensateAll
LinkPin
<<stereotype>>
lnk
SendPin
<<stereotype>>
snd
ReceivePin
<<stereotype>>
rcv
MessageType
<<stereotype>>
messageType
RequestPoint
<<stereotype>>
requestPoint
ServicePoint
<<stereotype>>
servicePoint