**Date:** Month yyyy

Precise Semantics of UML Composite Structures

*Version (e.g., Alpha 1, Beta 1, 1.0)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**OMG Document Number: dtc or ptc/200x-xx-xx**

**Standard document URL: http://www.omg.org/spec/acronym/1.0/PDF**

**Associated File(s)\*:** [**http://www.omg.org/spec/acronym/200xxxxx**](http://www.omg.org/spec/ALMAS/2008-02-01)

[**http://www.omg.org/spec/acronym/200xxxxx**](http://www.omg.org/spec/EXPRESS/20080202)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Source document: Title (document number)

\* Original file(s): Title (document number)

Copyright © 2008, company name  
Copyright © 2008, Object Management Group, Inc.  
Copyright © 2008, company name

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

**OMG’s Issue Reporting Procedure**

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.)

Table of Contents

[1 Scope 1](#_Toc317878072)

[2 Conformance 1](#_Toc317878073)

[3 Normative References 1](#_Toc317878074)

[4 Terms and Definitions 1](#_Toc317878075)

[5 Symbols 1](#_Toc317878076)

[6 Additional Information 2](#_Toc317878077)

[6.1 Changes to Adopted OMG Specifications [optional] 2](#_Toc317878078)

[6.2 Acknowledgements 2](#_Toc317878079)

[7 Abstract Syntax 3](#_Toc317878080)

[7.1 Overview 3](#_Toc317878081)

[7.2 Class Descriptions 3](#_Toc317878082)

[7.2.1 Class *(from StructuredClasses, InternalStructures)* 3](#_Toc317878083)

[7.2.2 ConnectableElement *(from InternalStructures)* 3](#_Toc317878084)

[7.2.3 Connector *(from InternalStructures)* 4](#_Toc317878085)

[7.2.4 ConnectorEnd *(from InternalStructures, Ports)* 4](#_Toc317878086)

[7.2.5 EncapsulatedClassifier *(from Ports)* 4](#_Toc317878087)

[7.2.6 Interface *(from Interfaces, Communications)* 5](#_Toc317878088)

[7.2.7 InvocationAction *(from InvocationActions)* 5](#_Toc317878089)

[7.2.8 Port *(from Ports)* 5](#_Toc317878090)

[7.2.9 Property *(from InternalStructures)* 6](#_Toc317878091)

[7.2.10 StructuredClassifier *(from InternalStructures)* 6](#_Toc317878092)

[7.2.11 Trigger *(from InvocationActions)* 7](#_Toc317878093)

[8 Semantics 8](#_Toc317878094)

[8.1 Overview 8](#_Toc317878095)

[8.2 Class Descriptions 8](#_Toc317878096)

[9 Test Suite 9](#_Toc317878097)

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG’s specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG’s specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

**OMG Specifications**

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

*http://www.omg.org/technology/documents/spec\_catalog.htm*

Specifications within the Catalog are organized by the following categories:

**OMG Modeling Specifications**

* UML

MOF

XMI

CWM

Profile specifications

**OMG Middleware Specifications**

* CORBA/IIOP

IDL/Language Mappings

Specialized CORBA specifications

CORBA Component Model (CCM)

**Platform Specific Model and Interface Specifications**

* CORBAservices

CORBAfacilities

OMG Domain specifications

OMG Embedded Intelligence specifications

OMG Security specifications

All of OMG’s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters  
140 Kendrick Street  
Building A, Suite 300  
Needham, MA 02494  
USA  
Tel: +1-781-444-0404  
Fax: +1-781-444-0320  
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult [*http://www.iso.org*](http://www.iso.org/)

**Typographical Conventions**

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

**Helvetica/Arial - 10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

**Courier - 10 pt. Bold:** Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

# Scope

The Scope clause shall appear at the beginning of each specification and define, without ambiguity, the subject of the specification and the aspect(s) covered. It indicates the limits of applicability of the specification or particular parts of it. It shall not contain requirements.

The scope shall be succinct so that it can be used as a summary for bibliographic purposes.

It shall be worded as a series of statements of fact.

# Conformance

The Conformance clause identifies which clauses of the specification are mandatory (or conditionally mandatory) and which are optional in order for an implementation to claim conformance to the specification.

Note: For conditionally mandatory clauses, the conditions must, of course, be specified.

# Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

List of normative references.

# Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Term

Definition

Term

Definition

Term

Definition

# Symbols

List of symbols/abbreviations.

# Additional Information

## Changes to Adopted OMG Specifications [optional]

TBD.

## Acknowledgements

TBD

# Abstract Syntax

TBD.

## Overview

TBD.

### Abstract Syntax

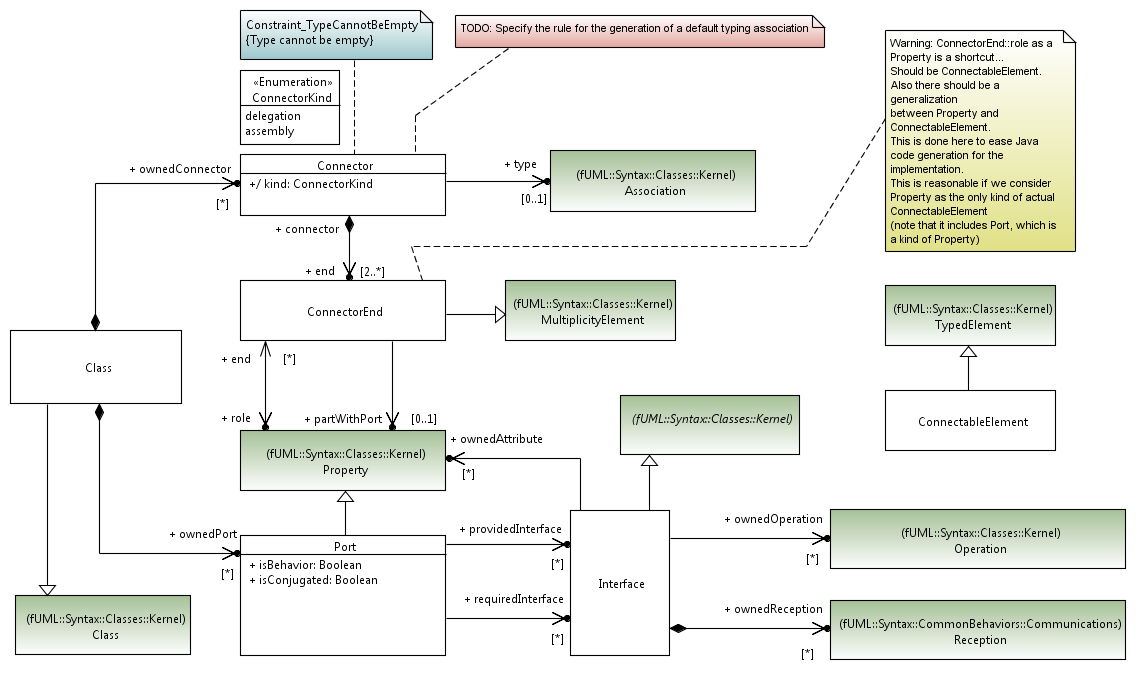


Figure 1. Composite Structures

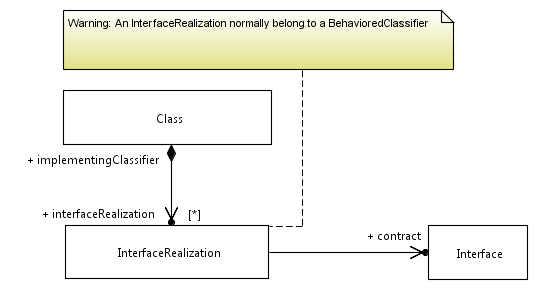


Figure 2. Interface Realization

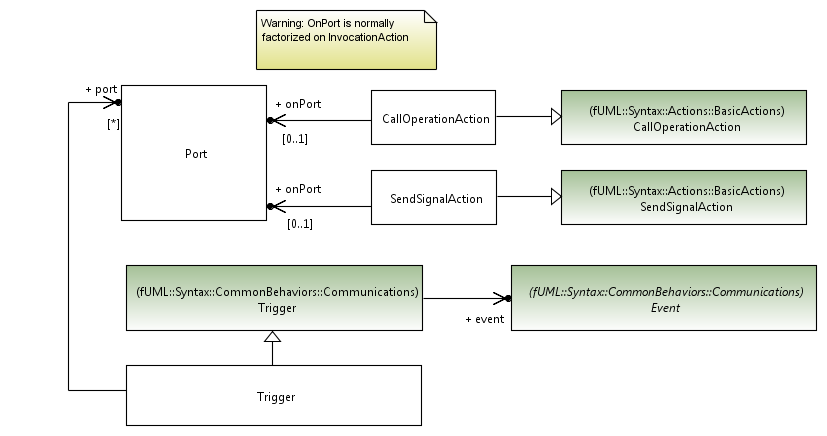


Figure 3. Communication

### Class Descriptions

### CallOperationAction

#### Generalizations

 fUML::Syntax::Actions::BasicActions::CallOperationAction

#### Properties

 onPort : Port [0..1]

### Class

#### Generalizations

 fUML::Syntax::Classes::Kernel::Class

#### Properties

 interfaceRealization : InterfaceRealization [0..\*]

 ownedConnector : Connector [0..\*]

 ownedPort : Port [0..\*]

### ConnectableElement

#### Generalizations

 fUML::Syntax::Classes::Kernel::TypedElement

### Connector

#### Generalizations

#### Properties

 end : ConnectorEnd [2..\*]

 kind : ConnectorKind [1]

 type : Association [0..1]

### ConnectorEnd

#### Generalizations

 fUML::Syntax::Classes::Kernel::MultiplicityElement

#### Properties

 partWithPort : Property [0..1]

 role : Property [1]

### Interface

#### Generalizations

 fUML::Syntax::Classes::Kernel::Classifier

#### Properties

 ownedAttribute : Property [0..\*]

 ownedOperation : Operation [0..\*]

 ownedReception : Reception [0..\*]

### InterfaceRealization

#### Generalizations

#### Properties

 contract : Interface [1]

### Port

#### Generalizations

 fUML::Syntax::Classes::Kernel::Property

#### Properties

 isBehavior : Boolean [1]

 isConjugated : Boolean [1]

 providedInterface : Interface [0..\*]

 requiredInterface : Interface [0..\*]

### SendSignalAction

#### Generalizations

 fUML::Syntax::Actions::BasicActions::SendSignalAction

#### Properties

 onPort : Port [0..1]

### Trigger

#### Generalizations

 fUML::Syntax::CommonBehaviors::Communications::Trigger

#### Properties

 port : Port [0..\*]

# Execution Model

Note from the Reston meeting:

We need to introduce the following classes in the ExecutionModel:

* **PortObject** : A particular kind of Object, which should embed a behavior describing request propagations across ports (which may be a propagation following attached connector instances, or insertion of the corresponding signal instance in the event pool of the Object owning this PortObject). This class probably needs to maintain a list of delegation connector instances and a list of assembly connector instances, in order to be able to determine paths for propagation.
  + **=> Maybe, not necessary to have port objects**.
* **ConnectorInstance (extends fuml::StructuredValue)** : In the case where a Connector is typed by an association, fUML::Link is probably sufficient (ends of the connector instance can be determined using featureValue). However, in the case where the connector is not typed, we cannot use Link, because it extends ExtensionalValue, and because we need something else that FeatureValues to identify ends.
  + **=> Unclassified links may be introduced in a revision of fUML**

**Alternative?**

**. 1. Dynamically create an association to type this connector**

**. 2. Through a transformation, generate the association typing the connector => The execution model can be written with the hypothesis that the connector is typed by an association.**

We need to extend the following classes from the fUML execution model:

* Structural feature actions: **Extend ClearStructuralFeatureValueActionActivation and RemoveStrucutralFeatureValueActionActivation** to account for the fact that when the corresponding actions are invoked for a structural feature which is a port, their execution may result in removing ConnectorInstances (i.e., not only links)
  + => If we make link, this is already covered by fuml
* Link actions: NOTE: I was about to write something on create and destroy link actions for addressing the case of ConnectorInstances described above (i.e. creating / destroying links which are not typed by associations), but it seems that writeLinkActions requires LinkEndDatas, which as far as I understand it prevents from creating a link which is not instance of an association.
* **Locus: overload *instantiate*** in order to enable instantiations resulting in “non-empty” objects (i.e., exploit the kind of structural configuration described by the composite structure)
  + Fig 9.27 => Issue of constructors. Note: We need a mechanism to manage stereotypes in fUML. Could we do that with some kinds of profile APIs? (e.g., for SysML, *applyBlock(Class), isBlock(Class)* )
* **Extend InvocationActionActivation (and SendSignalActionActivation and CallOperationActionActivation)** to account for the fact that the invocation targets a particular PortObject
  + Problem in UML:
    - An InvocationAction has a 1..1 target input pin (which is typed by the classifier of the target object), and references an operation (which should be an operation of the target object classifier). In the case of the port, the operation should be an operation of the required interface. => The onPort must be a port of the opposite, not a port of the context classifier.
    - Hope it will be fixed in UML 2.5
* **Extend SignalInstance** so that it can carry the information that it occurred on a specific PortObject. It may also be necessary to have an information capturing the last link crossed by the SignalInstance during the propagation process, so that it is possible for a PortObject to determine if it must propagate through delegation or assembly links (e.g., in the case where the PortObject represents a Port providing and requiring a same given Interface).
* **Extend AcceptEventActionEventAccepter** so that matching takes into account the port on which the SignalInstance occured

## Overview

TBD.

## Abstract syntax

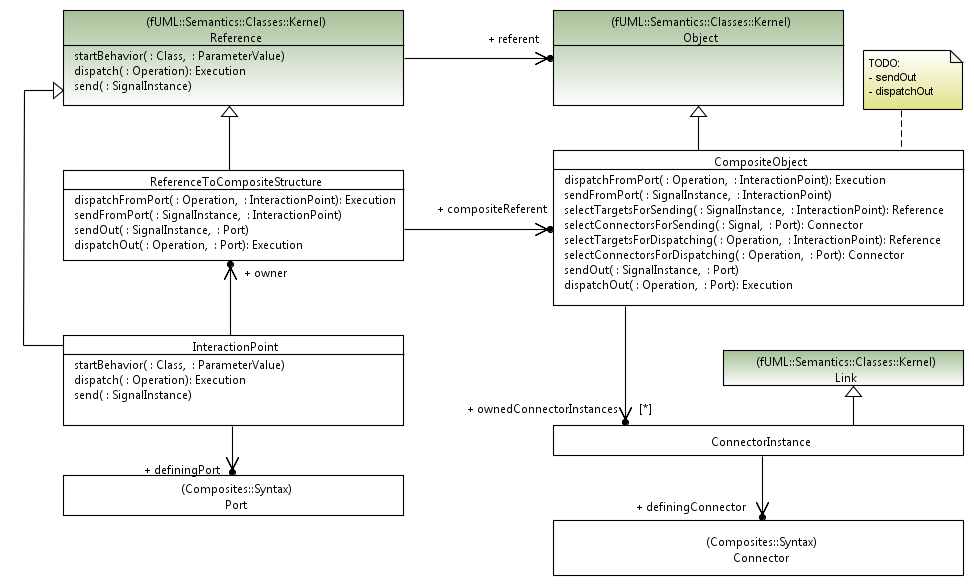


Figure 4. Composite Objects

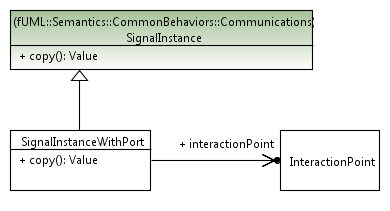


Figure 5. Signal

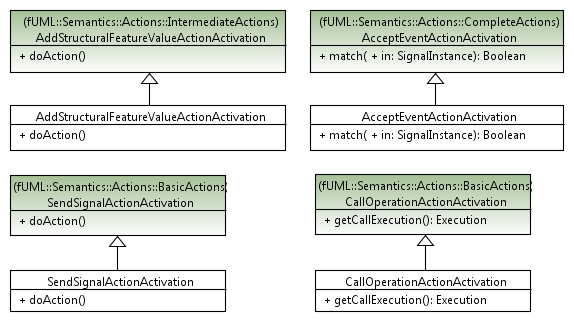


Figure 6. Actions

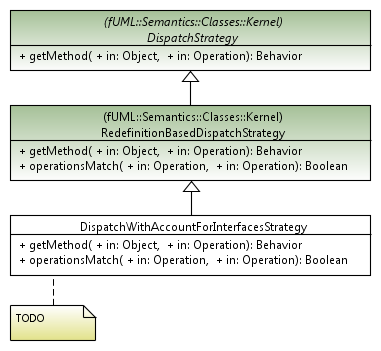


Figure 7. Strategies

## Class Descriptions

### AcceptEventActionActivation

The behavior of fUML CallOperationActionActivation::match() is overriden, in order to account for the fact that a given signal instance may need to be matched with triggers where a list of ports is given. (NOTE: Addresses requirement R9 "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.")

#### Generalizations

 fUML::Semantics::Actions::CompleteActions::AcceptEventActionActivation

#### Operations

[1] public match (in signalInstance : SignalInstance) : Boolean {

// Return true if the given signal instance matches a trigger of the accept

// action of this activation.

// Matching implies that the type of the signalInstance matches the Signal

// of one of the triggers.

// When the type matches with the Signal, and if the trigger specifies a

// list of ports,

// the signalInstance matches the trigger only if it occured on a port

// identified in the list.

AcceptEventAction action = (AcceptEventAction)(this.node) ;

TriggerList triggers = action.trigger ;

Signal signal = signalInstance.type ;

Boolean matches = false;

Integer i = 1;

while (!matches & i <= triggers.size()) {

Trigger t = triggers.getValue(i-1) ;

matches = ((SignalEvent)t.event).signal == signal ;

if (matches && (t instanceof Composites::Syntax::Trigger)) {

if (! (signalInstance instanceof SignalInstanceWithPort)) {

matches = false ;

}

else {

PortList portsOfTrigger = ((Composites::Syntax::Trigger)t).port ;

Port onPort =

((SignalInstanceWithPort)signalInstance).interactionPoint

.definingPort ;

Boolean portMatches = false ;

Integer j = 1 ;

while (! portMatches & j <= portsOfTrigger.size() ) {

portMatches = onPort == portsOfTrigger.getValue(j-1) ;

j = j + 1 ;

}

matches = portMatches ;

}

}

i = i + 1;

}

return matches;

}

### AddStructuralFeatureValueActionActivation

The behavior of fUML AddStructuralFeatureActionActivation::doAction() is overriden. In the case where the targeted structural feature is a port and the value to be added is a Reference, an InteractionPoint is created on the basis of the given Reference. It then behaves like in fUML, except that the execution continues using the created InteractionPoint instead of the given Reference.

#### Generalizations

 fUML::Semantics::Actions::IntermediateActions::AddStructuralFeatureValueActionActivation

#### Operations

[1] public doAction () {

// If the feature is a port and the input value to be added is a Reference,

// Replaces this Reference by an InteractionPoint, and then behaves

// as usual.

// If the feature is not a port, behaves as usual

AddStructuralFeatureValueAction action =

(AddStructuralFeatureValueAction)(this.node);

StructuralFeature feature = action.structuralFeature;

if (! (feature instanceof Port)) {

// Behaves as usual

super.doAction() ;

}

else {

ValueList inputValues = this.takeTokens(action.value);

// NOTE: Multiplicity of the value input pin is required to be 1..1.

Value inputValue = inputValues.getValue(0);

if (inputValue instanceof Reference) {

// First constructs an InteractionPoint from the inputValue

Reference reference = (Reference)inputValue ;

InteractionPoint interactionPoint = new InteractionPoint() ;

interactionPoint.referent = reference.referent ;

interactionPoint.definingPort = (Port)feature ;

// The value on action.object is necessarily instanceof

// ReferenceToCompositeStructure (otherwise, the feature cannot

// be a port)

ReferenceToCompositeStructure owner =

(ReferenceToCompositeStructure)this

.takeTokens(action.object).getValue(0);

interactionPoint.owner = owner ;

// Then replaces the Reference by an InteractionPoint

// in the inputValues

inputValues.remove(0) ;

inputValues.addValue(0,interactionPoint) ;

// Finally concludes with usual fUML behavior of

// AddStructuralFeatureValueAction (i.e., the usual behavior when

// the value on action.object pin is a StructuredValue)

Integer insertAt = 0;

if (action.insertAt != null) {

insertAt = ((UnlimitedNaturalValue)this

.takeTokens(action.insertAt).getValue(0))

.value.naturalValue;

}

if (action.isReplaceAll) {

owner.setFeatureValue(feature, inputValues, 0);

} else {

FeatureValue featureValue = owner.getFeatureValue(feature);

if (featureValue.values.size() > 0 & insertAt == 0 ) {

// If there is no insertAt pin, then the structural feature must

// be unordered, and the insertion position is immaterial.

insertAt = ((ChoiceStrategy)this.getExecutionLocus()

.factory.getStrategy("choice"))

.choose(featureValue.values.size());

}

if (feature.multiplicityElement.'isUnique') {

// Remove any existing value that duplicates the input value

Integer j = position(inputValue, featureValue.values, 1);

if (j > 0) {

featureValue.values.remove(j-1);

if (insertAt > 0 & j < insertAt) {

insertAt = insertAt - 1;

}

}

}

if (insertAt <= 0) {

// Note: insertAt = -1 indicates an unlimited value of "\*"

featureValue.values.addValue(inputValue);

} else {

featureValue.values.addValue(insertAt - 1, inputValue);

}

}

}

else {

// behaves as usual

super.doAction();

}

}

}

### CallOperationActionActivation

Extends fUML CallOperationActionActivation::getCallExecution(). If onPort is specified, instead of dispatching directly to target reference by calling operation dispatch, dispatchOut (cf. ReferenceToCompositeStructure) is called, so that the operation call will be finally dispatched to the environment (from where the execution will be taken). (Note: Adresses requirement R2 "Invocation actions may also be sent to a target via a given port, either on the sending object or on another object.")

#### Generalizations

 fUML::Semantics::Actions::BasicActions::CallOperationActionActivation

#### Operations

[1] public getCallExecution() : Execution {

// If onPort is not specified, behaves like in fUML

// If onPort is specified, and if the value on the target input pin is a

// reference, dispatch the operation

// to it and return the resulting execution object.

// As compared to fUML, instead of dispatching directly to target reference

// by calling operation dispatch,

// dispatchOut is called, so that the operation call will be finally

// dispatched to the environment (from where the execution will be taken).

CallOperationAction action = (CallOperationAction)(this.node);

Execution execution = null ;

if (action.onPort == null ) {

execution = super.getCallExecution() ;

}

else {

Value target = this.takeTokens(action.target).getValue(0);

if (target instanceof ReferenceToCompositeStructure) {

execution = ((ReferenceToCompositeStructure)target)

.dispatchOut(action.operation, action.onPort);

}

}

return execution;

}

### CompositeObject

CompositeObject extends fUML Object with specific operations for managing propagations of requests through ports, from the environment to the internals of this object, or from the objet to its environment. NOTE, this class addresses the following requirements: - R4: 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. - R5: If there is a connector attached to only one side of a port, any requests arriving at this port will terminante at this port [Non-behavior port] - R6: 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), if this classifier has any behavior. - R7: If there is no behavior defined for this classifier, any communication arriving at a behavior port is lost. - R8: 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 fowarded 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.

#### Generalizations

 fUML::Semantics::Classes::Kernel::Object

#### Properties

 ownedConnectorInstances : ConnectorInstance [0..\*]. The collection of ConnectorInstance owned by this CompositeObject. For each ConnectorInstance, definingConnector is a Connector belonging to one of the Class typing this CompositeObject

#### Operations

[1] public dispatchFromPort (in operation : Operation,

in interactionPoint : InteractionPoint)

: Execution {

// If the interaction is a behavior port, does nothing [for the moment... ?],

// since the only kind of event supported in fUML is SignalEvent

// If this is not a behavior port, select appropriate delegation links

// from interactionPoint, and propagates the operation call through

// these links

Execution execution = null ;

if (interactionPoint.definingPort.isBehavior) {

// Do nothing

}

else {

ReferenceList targets =

this.selectTargetsForDispatching(operation, interactionPoint) ;

// If targets is empty, no delegation target have been found,

// and the operation call will be lost

if (! (targets.size()==0)) {

// Choose one target non-deterministically

Integer index =

((ChoiceStrategy)this.locus.factory.getStrategy("choice"))

.choose(targets.size()) ;

Reference target = targets.getValue(index - 1) ;

execution = target.dispatch(operation) ;

}

}

return execution ;

}

[2] public dispatchOut (in operation : Operation, in onPort : Port) : Execution {

// TODO:

// Propagate the operation call through interaction points corresponding

// to onPorts, following appropriate links,

// This will result in calling dispatch(operation) on oppositeEnd found

// from the links

return null ;

}

[3] public selectConnectorsForDispatching(in operation : Operation,

in port : Port)

: Connector [\*] {

// From the given signal and port, retrieves potential connectors through

// which request can be delegated. These connectors are delegation

// connectors attached to Port interactionPoint.definingPort, and whose

// target provides the requested operation

ConnectorList connectors = new ConnectorList() ;

Integer i = 1 ;

// Iterates on types of this CompositeObject

while (i <= this.types.size()) {

Type t =this.types.getValue(i-1) ;

if (t instanceof Class\_) {

Class\_ class\_ = (Class\_) t ;

Integer j = 1 ;

// Iterates on Connectors of the current type

while (j <= class\_.ownedConnector.size()) {

Connector cddConnector = class\_.ownedConnector.getValue(j-1) ;

if (cddConnector.kind == ConnectorKind.delegation) {

Integer k = 1 ;

Boolean matches = false ;

// Iterates on ConnectorEnds of the current Connector

while (k <= cddConnector.end.size() && !matches) {

ConnectorEnd cddEnd = cddConnector.end.getValue(k-1) ;

if (!(cddEnd.role == port && cddEnd.partWithPort == null)) {

if (cddEnd.role instanceof Port) {

if (operation.owner instanceof Interface) {

// We have to look in provided interfaces of the port if

// they define directly or indirectly the Operation

Integer l = 1 ;

// Iterates on provided interfaces of the current end

InterfaceList providedInterfaces =

((Port)cddEnd.role).providedInterface ;

while (l <= providedInterfaces.size() && !matches) {

Interface interface\_ = ((Port)cddEnd.role)

.providedInterface

.getValue(l-1) ;

// Iterates on members of the current Interface

Integer m = 1 ;

while (m <= interface\_.member.size() && !matches) {

NamedElement cddOperation =

interface\_.member

.getValue(m-1) ;

if (cddOperation instanceof Operation) {

DispatchWithAccountForInterfacesStrategy strategy =

new DispatchWithAccountForInterfacesStrategy() ;

matches =

strategy.operationsMatch(

(Operation)cddOperation,

operation) ;

if (matches) {

connectors.addValue(cddConnector) ;

}

}

m = m + 1 ;

}

l = l + 1 ;

}

}

}

else {

// The connector does not target a Port.

// We have to look if the Classifier typing this property

// directly or indirectly provides a Reception for signal

Integer l = 1 ;

if (cddEnd.role.typedElement.type instanceof Class\_) {

while (l <= ((Class\_)cddEnd.role.typedElement.type)

.member.size()

&& !matches) {

NamedElement cddOperation =

((Class\_)cddEnd.role.typedElement.type)

.member.getValue(l-1) ;

if (cddOperation instanceof Operation) {

DispatchWithAccountForInterfacesStrategy strategy =

new DispatchWithAccountForInterfacesStrategy() ;

matches =

strategy.operationsMatch(

(Operation)cddOperation,

operation) ;

if (matches) {

connectors.addValue(cddConnector) ;

}

}

l = l + 1 ;

}

}

}

}

k = k + 1 ;

}

}

j = j + 1 ;

}

i = i + 1 ;

}

}

return connectors ;

}

[4] public selectConnectorsForSending(in signal : Signal,

in port : Port)

: Connector [\*] {

// From the given signal and port, retrieves potential connectors

// through which request can be delegated

// These connectors are delegation connectors attached to Port

// interactionPoint.definingPort,

// and whose target provide a reception for Signal signalInstance.type

ConnectorList connectors = new ConnectorList() ;

Integer i = 1 ;

// Iterates on types of this CompositeObject

while (i <= this.types.size()) {

Type t =this.types.getValue(i-1) ;

if (t instanceof Class\_) {

Class\_ class\_ = (Class\_) t ;

Integer j = 1 ;

// Iterates on Connectors of the current type

while (j <= class\_.ownedConnector.size()) {

Connector cddConnector =

class\_.ownedConnector.getValue(j-1) ;

if (cddConnector.kind == ConnectorKind::delegation) {

Integer k = 1 ;

Boolean matches = false ;

// Iterates on ConnectorEnds of the current Connector

while (k <= cddConnector.end.size() && !matches) {

ConnectorEnd cddEnd = cddConnector.end.getValue(k-1) ;

if (!(cddEnd.role == port && cddEnd.partWithPort == null)) {

if (cddEnd.role instanceof Port) {

// We have to look in provided interfaces of the port

// if there is a reception defined for signal

Integer l = 1 ;

// Iterates on provided interfaces of the current end

InterfaceList providedInterfaces =

((Port)cddEnd.role).providedInterface ;

while (l <= providedInterfaces.size() && !matches) {

Interface interface\_ =

((Port)cddEnd.role).providedInterface.getValue(l-1) ;

// Iterates on Receptions of the current interface

Integer m = 1 ;

while (m <= interface\_.ownedReception.size()

&& !matches) {

if (interface\_.ownedReception.getValue(m-1).signal

== signal) {

matches = true ;

connectors.addValue(cddConnector) ;

}

m = m + 1 ;

}

l = l + 1 ;

}

if (matches == false) {

// No matching reception has been found in reception

// directly owned by provided interfaces

// Need to check in inherited members of these

//provided interfaces.

l = 1 ;

// Iterates again on provided interfaces

while (l <= providedInterfaces.size() && ! matches) {

Interface interface\_ =

providedInterfaces.getValue (l -1) ;

// Iterates on inherited members of the current Interface

Integer m = 1 ;

while (m <= interface\_.inheritedMember.size()

&& !matches) {

NamedElement cddReception =

interface\_.inheritedMember.getValue(m-1) ;

if ((cddReception instanceof Reception)

&&

(((Reception)cddReception).signal == signal)) {

matches = true ;

connectors.addValue(cddConnector) ;

}

m = m + 1 ;

}

l = l + 1 ;

}

}

}

else {

// The connector does not target a Port.

// We have to look if the Classifier typing this property

// directly or indirectly provides a Reception for signal

Integer l = 1 ;

if (cddEnd.role.typedElement.type instanceof Classifier) {

while (l <= ((Classifier)cddEnd.role

.typedElement.type).member.size()

&& !matches) {

NamedElement cddReception =

((Classifier)cddEnd.role.typedElement.type)

.member.getValue(l-1) ;

if ((cddReception instanceof Reception)

&&

(((Reception)cddReception).signal == signal)) {

matches = true ;

connectors.addValue(cddConnector) ;

}

l = l + 1 ;

}

}

}

}

k = k + 1 ;

}

}

j = j + 1 ;

}

i = i + 1 ;

}

}

return connectors ;

}

[5] public selectTargetsForDispatching (in operation : Operation,

in interactionPoint : InteractionPoint)

: Reference [\*] {

// From the given operation and interactionPoint, retrieves potential

// connectors through which request can be delegated

// These connectors are delegation connectors attached to Port

// interactionPoint.definingPort,

// and whose target provides or realize operation

ConnectorList connectors =

this.selectConnectorsForDispatching(operation,

interactionPoint.definingPort) ;

// Select links owned by the context object for which the

// definingConnector is included in the list of matching connectors.

Integer i = 1 ;

ConnectorInstanceList connectorInstances =

new ConnectorInstanceList() ;

while (i <= connectors.size()) {

Integer j = 1 ;

Connector connector = connectors.getValue(i-1) ;

while (j <= this.ownedConnectorInstances.size()) {

ConnectorInstance connectorInstance =

this.ownedConnectorInstances.getValue(j-1) ;

if (connectorInstance.definingConnector == connector) {

connectorInstances.addValue(connectorInstance) ;

}

j=j+1 ;

}

i = i+1 ;

}

// For each matching link, retrieves the end value opposite

// to interactionPoint.

// If this value is a reference (which means that it is possible to dispatch

// operation to it), it is added in the list of potential targets.

ReferenceList targets = new ReferenceList() ;

i = 1 ;

while (i <= connectorInstances.size()) {

ConnectorInstance link = connectorInstances.getValue(i-1) ;

Association association = link.type ;

Property oppositeEnd = association.memberEnd.getValue(0);

if (oppositeEnd == interactionPoint.definingPort) {

oppositeEnd = association.memberEnd.getValue(1);

}

Value value = link.getFeatureValue(oppositeEnd).values.getValue(0) ;

if (value instanceof Reference) {

targets.addValue((Reference)value) ;

}

i = i + 1;

}

// if targets is empty, no matching targets have been found,

// and the operation call will be lost

return targets ;

}

[6] public selectTargetsForSending (in signalInstance : SignalInstance,

in interactionPoint : InteractionPoint)

: Reference[\*] {

// From the given signalInstance and interactionPoint,

// retrieves potential connectors through which request can be delegated

// These connectors are delegation connectors attached to

// Port interactionPoint.definingPort, and whose target provide a

// reception for Signal signalInstance.type

ConnectorList connectors =

this.selectConnectorsForSending(signalInstance.type,

interactionPoint.definingPort) ;

// Select links owned by the context object for which the

// definingConnector is included in the list of matching connectors.

Integer i = 1 ;

ConnectorInstanceList connectorInstances =

new ConnectorInstanceList() ;

while (i <= connectors.size()) {

Integer j = 1 ;

Connector connector = connectors.getValue(i-1) ;

while (j <= this.ownedConnectorInstances.size()) {

ConnectorInstance connectorInstance =

this.ownedConnectorInstances.getValue(j-1) ;

if (connectorInstance.definingConnector == connector) {

connectorInstances.addValue(connectorInstance) ;

}

j=j+1 ;

}

i = i+1 ;

}

// For each matching link, retrieves the end value opposite

// to interactionPoint.

// If this value is a reference (which means that it is possible to send it

// a signal), it is added in the list of potential targets.

ReferenceList targets = new ReferenceList() ;

i = 1 ;

while (i <= connectorInstances.size()) {

ConnectorInstance link = connectorInstances.getValue(i-1) ;

Association association = link.type ;

Property oppositeEnd = association.memberEnd.getValue(0);

if (oppositeEnd == interactionPoint.definingPort) {

oppositeEnd = association.memberEnd.getValue(1);

}

Value value = link.getFeatureValue(oppositeEnd).values.getValue(0) ;

if (value instanceof Reference) {

targets.addValue((Reference)value) ;

}

i = i + 1;

}

// if targets is empty, no matching targets have been found,

// and the signal instance will be lost

return targets ;

}

[7] public sendFromPort (in signalInstance : SignalInstance,

in interactionPoint : InteractionPoint) {

// If the interaction is a behavior port,

// creates a SignalInstanceWithPort from the signal instance,

// and sends it as usual using operation send

// If this is not a behavior port,

// select appropriate delegation targets from interactionPoint,

// and propagates the signal to these targets

if (interactionPoint.definingPort.isBehavior) {

SignalInstanceWithPort newSignalInstance =

new SignalInstanceWithPort() ;

SignalInstance copy = (SignalInstance)signalInstance.copy() ;

newSignalInstance.featureValues = copy.featureValues ;

newSignalInstance.type = copy.type ;

newSignalInstance.interactionPoint = interactionPoint ;

this.send(newSignalInstance) ;

}

else {

ReferenceList targets =

this.selectTargetsForSending(signalInstance,

interactionPoint) ;

// If targets is empty, no delegation target have been found,

// and the signal is lost

Integer i = 1 ;

// Do the following concurrently

while (i <= targets.size()) {

Reference target = targets.getValue(i-1) ;

target.send(signalInstance) ;

i = i + 1 ;

}

}

}

[8] public sendOut(in signalInstance : SignalInstance, in onPort : Port) {

// TODO:

// Propagate the signal instance through interaction points

// corresponding to onPorts, following appropriate links.

// This will result in calling send(signalInstance) on

// oppositeEnd found from the links

}

### ConnectorInstance

ConnectorInstance extends Link with the ability to specify that this association instance plays a particular Connector. NOTE: The execution model described in this specification makes the hypothesis that connectors are necessarily typed by an Association.

#### Generalizations

 fUML::Semantics::Classes::Kernel::Link

#### Properties

 definingConnector : Connector [1]. The Connector played by this ConnectorInstance

### DispatchWithAccountForInterfacesStrategy

Extends fUML RedefinitionBasedDispatchStrategy to account for the fact that the invoked operation may belong to an interface, and not to one of the classifiers of the target object (NOTE: Not mandatory to have it defined as an extension. Could be defined as direct specialization of DispatchStrategy)

#### Generalizations

 fUML::Semantics::Classes::Kernel::RedefinitionBasedDispatchStrategy

#### Operations

[1] public getMethod (in object : Object,

in operation : Operation )

: Behavior {

// Override getMethod so that it accounts for the fact that the Operation

// may belong to an Interface (realized by one of the classifier of object),

// and not directly to one of the classifier of object.

return null ;

}

[2] public operationsMatch (in ownedOperation : Operation ,

in baseOperation : Operation )

: Boolean {

// Override operationsMatch, in the case where baseOperation belongs

// to an Interface

return false ;

}

### InteractionPoint

An InteractionPoint represents the runtime manifestation of a Reference to an Object playing the role of a Port. More specifically, it overrides operation dispatching and signal receptions in order to capture the specific propagation semantics of requests targeting a port. NOTE: This class is related to the following requirements: - R1. 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" clause

#### Generalizations

 fUML::Semantics::Classes::Kernel::Reference

#### Properties

 definingPort : Port [1]. The Port for which this InteractionPoint is a runtime manifestation

 owner : ReferenceToCompositeStructure [1]. Represents the Reference to the CompositeObject owning this InteractionPort. NOTE: This is introduced to address requirement R3 (It represents the "link from that instance to the instance of the owning classifier [...] through which communication is forwarded to the instance of the owning classifier or through which the owning classifier communicates)

#### Operations

[1] public dispatch (in operation : Operation ) : Execution {

// Delegates dispatching to the owning object

return this.owner.dispatchFromPort(operation, this) ;

}

[2] public send (in signalInstance : SignalInstance ) {

// Delegates sending to the owning object

this.owner.sendFromPort(signalInstance, this) ;

}

[3] public startBehavior (in classifier : Class [0..1],

in inputs : ParameterValue [\*]) {

// Overriden to do nothing

}

### ReferenceToCompositeStructure

This class extends fuml Reference with specific operations for managing request propagation through ports, from the environment to the internals of the referent object, or from the referent objet to its environment. (NOTE: Addresses requirement R1 "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" clause", and R2 "Invocation actions may also be sent to a target via a given port, either on the sending object or on another object.")

#### Generalizations

 fUML::Semantics::Classes::Kernel::Reference

#### Properties

 compositeReferent : CompositeObject [1]. The composite object referenced by this ReferenceToCompositeStructure. This property subsets Reference::referent.

#### Operations

[1] public dispatchFromPort (in operation : Operation,

in interactionPoint : InteractionPoint)

: Execution {

//Delegates dispatching to composite referent

return this.compositeReferent.dispatchFromPort(operation, interactionPoint) ;

}

[2] public dispatchOut (in operation : Operation, in onPort : Port) : Execution {

// Delegates dispatching (through the port, to the environment)

// to compositeReferent

return this.compositeReferent.dispatchOut(operation, onPort) ;

}

[3] public sendFromPort (in signalInstance : SignalInstance,

in interactionPoint : InteractionPoint) {

// delegates sending to composite referent

this.compositeReferent.sendFromPort(signalInstance, interactionPoint) ;

}

[4] public sendOut(in signalInstance : SignalInstance, in onPort : Port) {

// Delegates sending (through the port, to the environment)

// to compositeReferent

this.compositeReferent.sendOut(signalInstance, onPort) ;

}

### SendSignalActionActivation

Extends behavior of fUML SendSignalActionActivation::doAction(). If onPort is specified, instead of sending directly to target reference by calling operation send, sendOut (cf. ReferenceToCompositeStructure) is called, so that the constructed signal instance will be finally sent to the environment. (Note: Addresses requirement R2 "Invocation actions may also be sent to a target via a given port, either on the sending object or on another object.")

#### Generalizations

 fUML::Semantics::Actions::BasicActions::SendSignalActionActivation

#### Operations

[1] public doAction () {

// If onPort is not specified, behaves like in fUML

// If onPort is specified,

// Get the value from the target pin. If the value is not a reference,

// then do nothing.

// Otherwise, construct a signal using the values from the argument pins

// As compared to fUML, instead of sending directly to target reference

// by calling operation send,

// sendOut is called, so that the constructed signal will be finally sent

// to the environment.

SendSignalAction action = (SendSignalAction)(this.node);

if (action.onPort == null) {

super.doAction() ;

}

else {

Value target = this.takeTokens(action.target).getValue(0) ;

if (target instanceof ReferenceToCompositeStructure) {

Signal signal = action.signal;

SignalInstance signalInstance = new SignalInstance();

signalInstance.type = signal;

PropertyList attributes = signal.ownedAttribute;

InputPinList argumentPins = action.argument;

Integer i = 0 ;

while ( i < attributes.size()) {

Property attribute = attributes.getValue(i);

InputPin argumentPin = argumentPins.getValue(i);

ValueList values = this.takeTokens(argumentPin);

signalInstance.setFeatureValue(attribute, values, 0);

}

ReferenceToCompositeStructure targetReference =

(ReferenceToCompositeStructure)target ;

Port onPort = action.onPort ;

targetReference.sendOut(signalInstance, onPort) ;

}

}

}

### SignalInstanceWithPort

SignalInstanceWithPort extends fUML SignalInstance with the ability to reference the specific InteractionPoint on which it occured. This is introduced to address the requirement R9 ("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.").

#### Generalizations

 fUML::Semantics::CommonBehaviors::Communications::SignalInstance

#### Properties

 interactionPoint : InteractionPoint [1]. The InteractionPoint on which this signal instance occured.

#### Operations

[1] public copy () : Value {

// Create a new signal instance with the same type, interaction point and

// feature values as this signal instance.

SignalInstanceWithPort newValue = (SignalInstanceWithPort) super.copy();

newValue.type = this.type ;

newValue.interactionPoint = this.interactionPoint ;

return newValue;

}

# Test Suite

TBD.

**Annex A: Title**

(normative)

A.1  Clause heading

Text

**Normative annexes** are integral parts of the standard. Their presence is optional. An annex’s normative status (as opposed to informative) shall be made clear by the way in which it is referred to in the text and under the heading of the annex.

**Informative annexes** give additional information intended to assist the understanding or use of the standard and shall not contain provisions to which it is necessary to conform in order to be able to claim compliance with the standard. Their presence is optional. An annex’s informative status (as opposed to normative) shall be made clear by the way in which it is referred to in the text and under the heading of the annex.

A.2  Clause heading

Text

**Annex B: Title**

(normative)

B.1 Sample IDL

#pragma prefix “http\_//example.com"

**module stockquote\_wsdl {**

interface StockQuotePortType {

typedef sequence<float> ArrayOfFloat;

typedef struct TimePeriod {

wstring startTime;

wstring endTime;

};

ArrayOfFloat GetTradePrices(

in wstring tickerSymbol,

in TimePeriod timePeriod,

out float frequency);

};

};

B.2 Sample Code

<?xml version="1.0"?>

<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"

xmlns:tns="http://example.com/stockquote.wsdl"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsd1="http://example.com/stockquote/schema"

xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>

<schema targetNamespace="http://example.com/stockquote/schema"

xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns="http://www.w3.org/2001/XMLSchema">

<complexType name="TimePeriod">

<all>

<element name="startTime" type="xsd:string"/>

<element name="endTime" type="xsd:string"/>

</all>

</complexType>

<complexType name="ArrayOfFloat">

<complexContent>

<restriction base="soapenc:Array">