The VCL Model of the Barbados Crisis Management System

Nuno Amálio
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi
L-1359 Luxembourg

TR-LASSY-12-09

1 Introduction

This document presents a model of the bCMS (Barbados Crisis Management System) case study [1] expressed in the Visual Contract Language (VCL) [2, 3, 4]. VCL is a visual and formal language for the abstract description of software systems; its novelty lies in its capability to express predicates visually. All the diagrams of the model presented here have been built using VCL's tool, the Visual Contract Builder (VCB)[1][4]

2 Background: VCL

VCL [2, 4] is a formal language designed for the abstract description of software systems. Its modelling paradigms are set theory, object-orientation and design-by-contract (pre- and post-conditions). Currently, the semantics is defined thorough a translation into the formal language Z.

A VCL model comprises a collection of packages. Each package comprises one package diagram (PD), one structural diagram (SD), one behavioural diagram (BD) and several assertion and contract diagrams (ADs and CDs). PDs define VCL packages (coarse-grained modules) and their dependencies with other packages (e.g. Figs. 1a, 2a and 3a). SDs define structures and their relations that together make the state space of a package (e.g. Figs. 1b, 2b and 3b). BDs provide a map over the package's operations (e.g. Figs. 3c and 6c). ADs define predicates over a single state, which can be used to define invariants and queries (e.g. Figs. 4d, and 5b). Finally, CDs describe dynamics through a contract: a pair of predicates representing pre- and post-conditions (e.g. Figs. 4a and 4b).

3 The VCL Model of bCMS: overview

Package	**Description**
CommonTypes	Contains common types used across the various packages of the bCMS model.
CrisisCommon	Defines common types for the purpose of defining the problem domain of bCMS: crisis Management.
CrisisInfoPkg	Holds information about crisis that the bCMS has to manage.
VehicleManagement	Manages information of rescue vehicles assigned to crisis.
RoutePlanning	Deals with the planning of routes and associated information.
TimeOutPkg	Deals with the timeouts that can be triggered during crisis negotiation.
CrisisManagement	Supports the coordination of crisis.
Authentication	Deals with the authentication security concern.
bCMSSys	Representing the entire bCMS system with authentication.

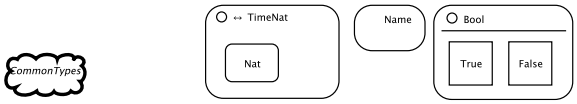
Table 1: The VCL packages of bCMS

This document presents a VCL model of the entire bCMS system with authentication. A VCL model is partitioned into packages. The packages that make the VCL model of bCMS are summarised in table 1.

[1] `http://vcl.gforge.uni.lu`

4 Package `CommonTypes`

Figure 1a presents the package diagram of the container package `CommonTypes`. This package's SD (Fig. 1b) introduces blobs `TimeNat` (time as natural numbers), `Name` (a set of names) and `Bool` (booleans with values `True` and `False`).

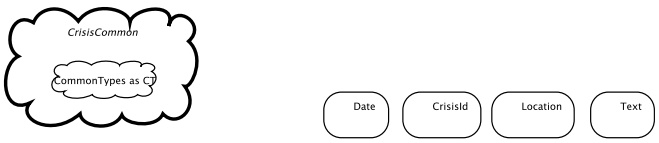


(a) PD of `CommonTypes` (b) SD of `CommonTypes`

Figure 1: The package `CommonTypes`

5 Package `CrisisCommon`

Figure 2a presents the package diagram of the container package `CrisisCommon`, which incorporates the package `CommonTypes`. Package `CrisisCommon` includes all structures defined in `CommonTypes` plus some structures of its own. This package's SD (Fig. 2b) introduces blobs `Date` (set of dates), `CrisisId` (set of crisis identifiers), and `Location` (set of GPS locations) and `Text` (text that is associated with descriptions).



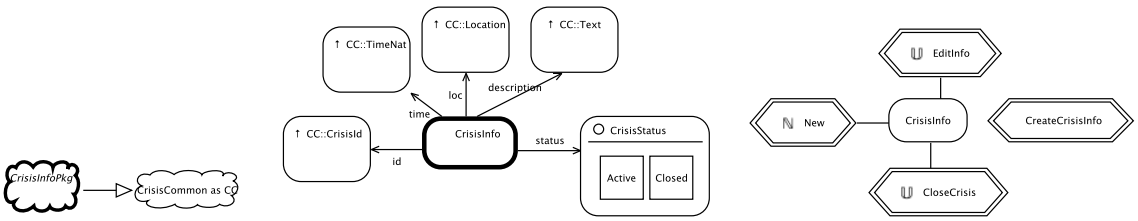
(a) PD of `CrisisCommon` (b) SD of `CrisisCommon`

Figure 2: The package `CrisisCommon`

6 Package `CrisisInfo`

Figure 3a presents the package diagram of the ensemble package `CrisisInfoPkg`, which sees (hollow-headed arrow) the package `CrisisCommon`. The package's SD (Fig. 3b) introduces the class (or domain blob) `CrisisInfo`, which holds the information associated with a crisis as defined by the entry *Crisis Details* of the data dictionary in [1]. `CrisisInfo` is mostly defined from types defined in package `CrisisCommon` (alias `CC`); a `CrisisInfo` holds a crisis identifier (`id`), a `time` when it was created, a crisis location (`loc`), a `description` and a `status` (`Active` or `Closed`).

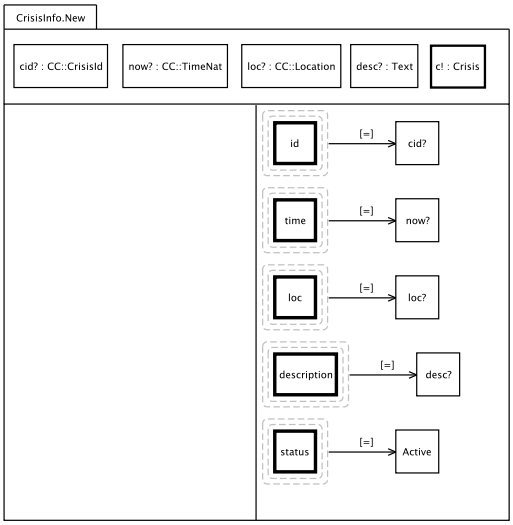
Figure 3c presents the BD of package `CrisisInfoPkg`. This introduces three operations for the class `CrisisInfo`: the constructor `New` and the update operations `EditInfo` and `CloseCrisis`. In addition, the BD introduces the global operation `CreateCrisisInfo`. These operations are defined as follows:



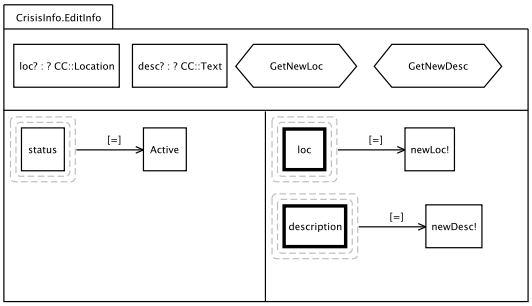
(a) PD of `CrisisInfo` (b) SD of `CrisisInfo` (c) BD of `CrisisInfo`

Figure 3: PD, SD and BD of package `CrisisInfoPkg`

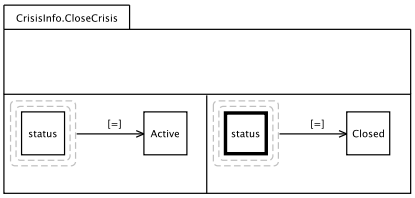
- The operation `New` (Fig. 4a) receives as inputs a crisis identifier (`cid?`), the current time (`now?`), a location (`loc?`) and a crisis description (`desc?`) and assigns them to newly created object (output `c!`). The object's status is set to `Active`.
- The operation `EditInfo` (Fig. 4b) is used during crisis negotiation while coordinators agree on the details of a crisis. It gives the possibility of updating the crisis location and description. The operation receives as inputs an optional crisis description and an optional location; the operations `GetNewDesc` (Fig. 4d) and `GetNewLoc` (Fig. 4e) then retrieve the new value to update crisis description and location depending on the values of the optional inputs. The operation's pre-condition requires that the `Crisis` status is `Active`.
- The operation `CloseCrisis` (Fig. 4c) is called when the coordinators agree to close a crisis. It requires that the crisis status is `Active` (pre-condition) and sets the crisis' status to `Closed`.
- The global operation `CreateCrisisInfo` (Fig. 5a) selects an unused crisis identifier to assign to the new crisis non-deterministically through operation `IsDistinctCrisisId` (Fig. 5b) and calls the constructor operation `New` of class `CrisisInfo`.



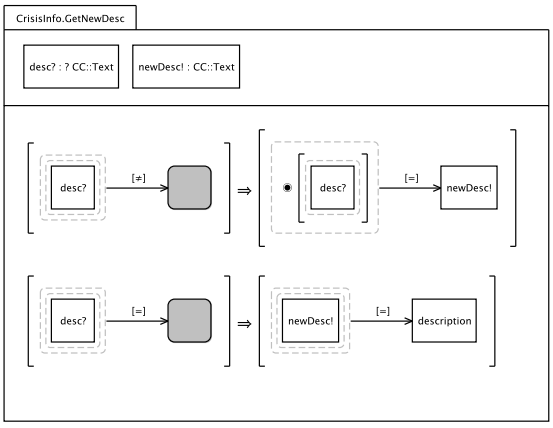
(a) Operation `CrisisInfo.New`



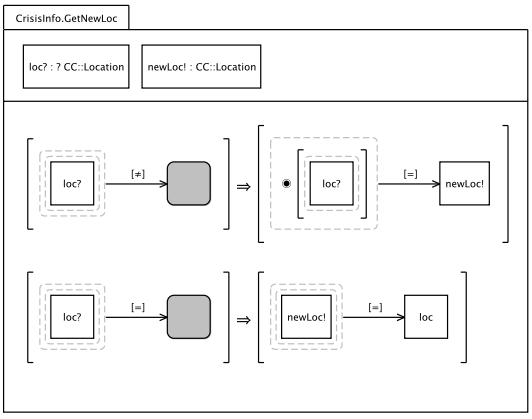
(b) Operation `CrisisInfo.EditInfo`



(c) Operation `CrisisInfo.CloseCrisis`

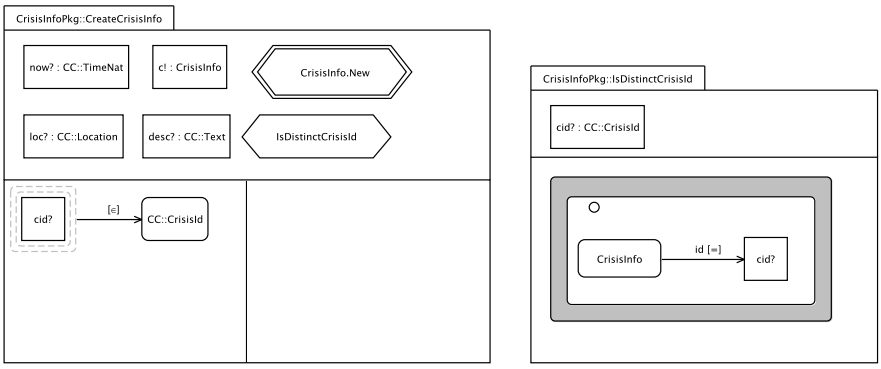


(d) Operation `CrisisInfo.GetNewDesc`



(e) Operation `CrisisInfo.GetNewLoc`

Figure 4: Local operations of blob `CrisisInfo`

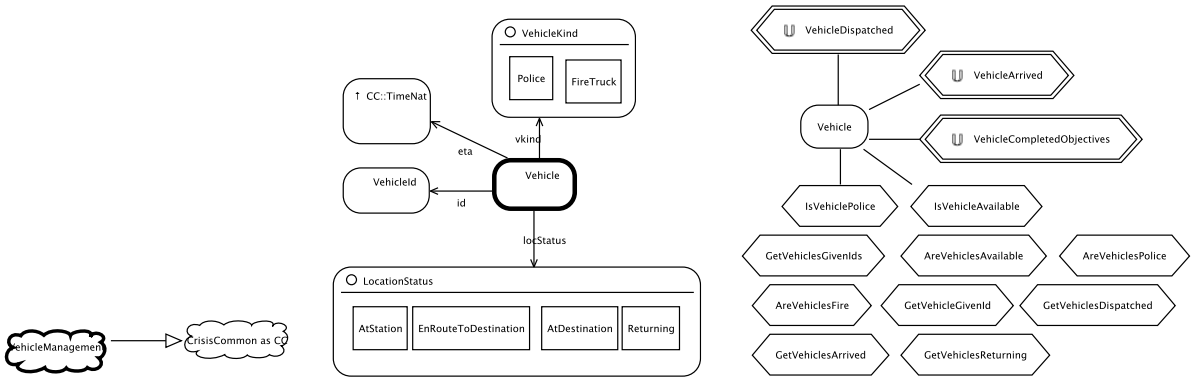


(a) Operation `CreateCrisisInfo` (b) Operation `IsDistinctCrisisId`

Figure 5: Global operations of package `CrisisInfoPkg`

7 Package `VehicleMangement`

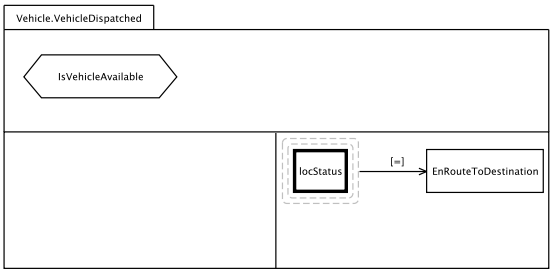
This package is responsible for managing the information concerning the vehicles that are assigned to crisis locations. Fig. 6a presents this package's PD, which sees the package `CommonCrisis` (identified with alias `CC`). The SD (Fig 6b) defines the class `Vehicle`, which has as properties an identifier (`id`), an expected time of arrival (`eta`), a vehicle kind (`vkind`) and a location status (`AtStation`, `EnRouteToDestination`, `AtDestination` and `Returning`).



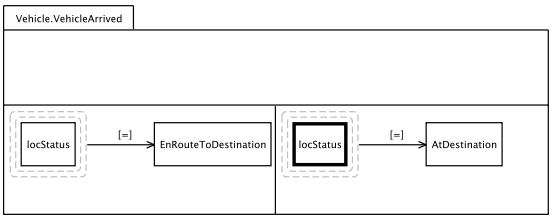
(a) PD of `VehicleMangement` (b) SD of `VehicleMangement` (c) BD of `VehicleMangement`

Figure 6: PD, SD and BD of package `VehicleMangement`

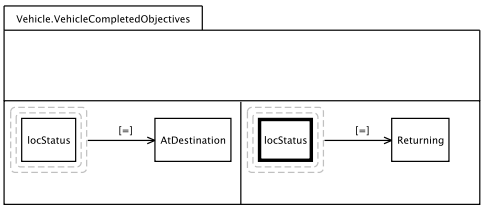
The BD of `VehicleMangement` (Fig. 6c) introduces several local operations for the class `Vehicle` and several global package operations.



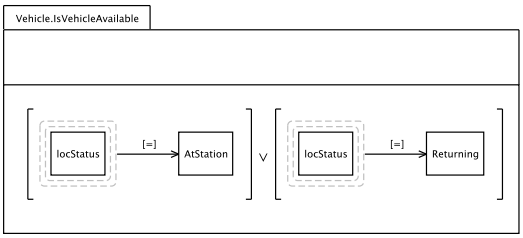
(a) Operation `Vehicle.VehicleDispatched`



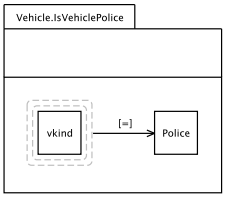
(b) Operation `Vehicle.VehicleArrived`



(c) Operation `Vehicle.VehicleCompletedObjectives`

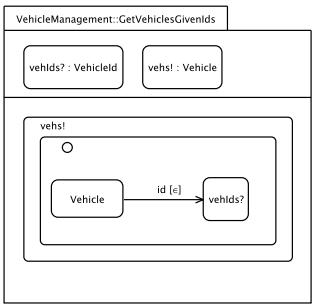


(d) Operation `Vehicle.IsVehicleAvailable`

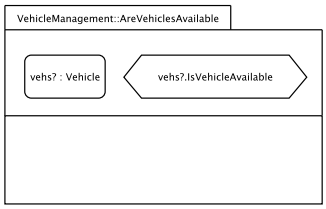


(e) Operation `Vehicle.IsVehiclePolice`

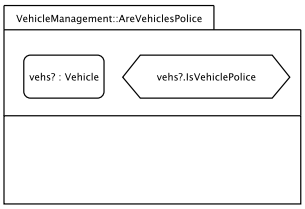
Figure 7: Local operations of blob `Vehicle`



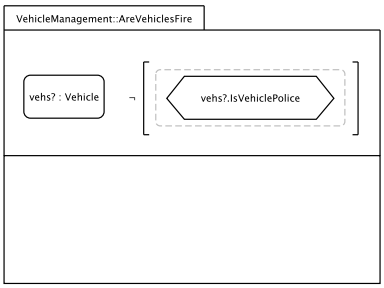
(a) Operation `GetVehiclesGivenIds`



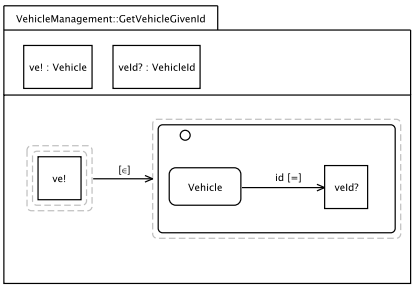
(b) Operation `AreVehiclesAvailable`



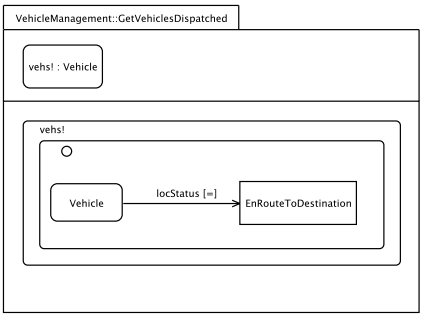
(c) Operation `AreVehiclesPolice`



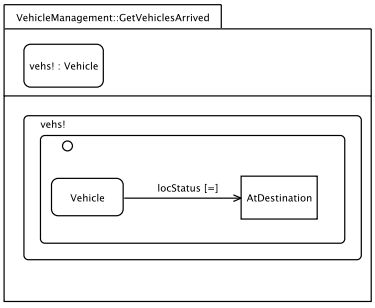
(d) Operation `AreVehiclesFire`



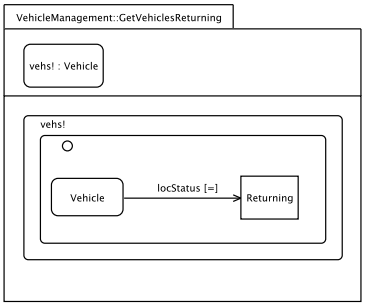
(e) Operation `GetVehicleGivenId`



(f) Operation `GetVehiclesDispatched`



(g) Operation `GetVehiclesArrived`

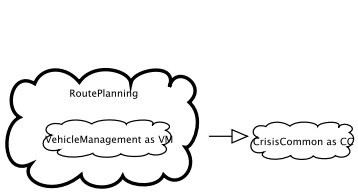


(h) Operation `GetVehiclesReturning`

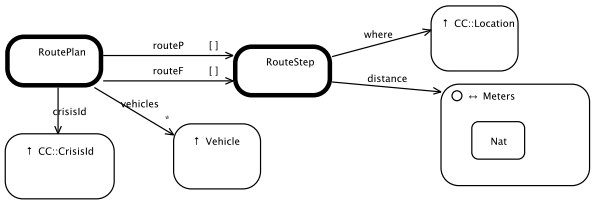
Figure 8: Global operations of package `VehicleManagement`

8 Package `RoutePlanning`

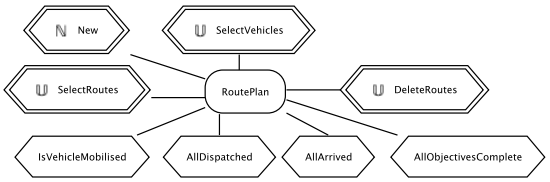
This package is responsible for managing the routes that rescue vehicles need to follow to arrive at crisis locations. Fig. 9a presents this package's PD, which sees the package `CommonCrisis` (identified with alias `CC`) and incorporates package `VehicleManagement` (identified with alias `VM`). The SD (Fig 9b) defines the classes (or domain blobs) `RoutePlan` and `RouteStep`. A `RoutePlan` holds a crisis identifier (`crisisId`), a set of `vehicles` and two sequence of route steps (symbol `[]`): one for police vehicles (`routeP`) and one for fire vehicles (`routeF`). A `RouteStep` represents a step in the route; it comprises a location (`where`) and a `distance`.



(a) PD of `RoutePlanning`



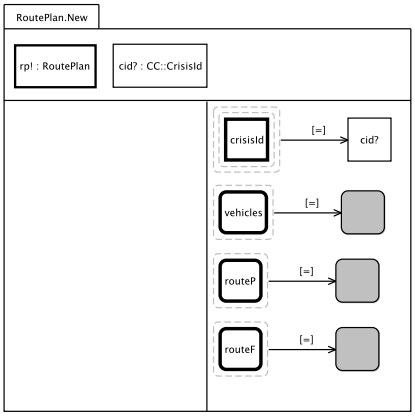
(b) SD of `RoutePlanning`



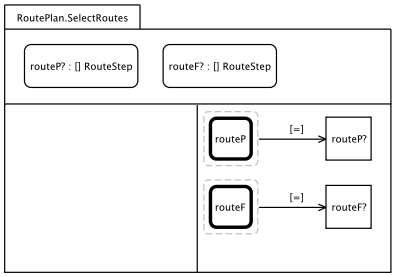
(c) BD of `RoutePlanning`

Figure 9: PD, SD and BD of package `RoutePlanning`

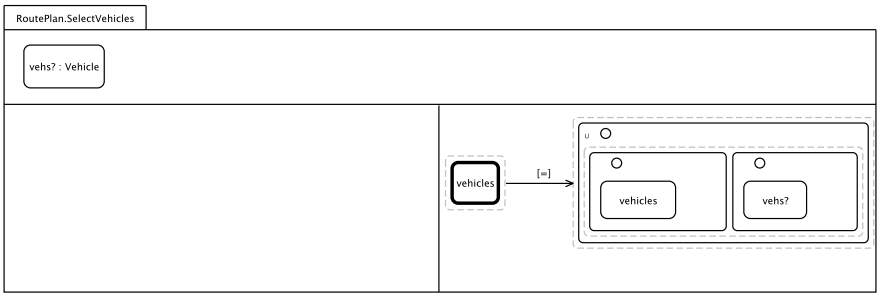
The package's BD (Fig:RoutePlanningBD) introduces several operations for the class `RoutePlan` to operate on route plans.



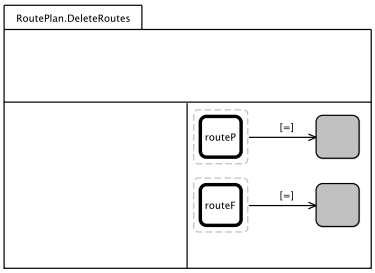
(a) Operation `RoutePlan.New`



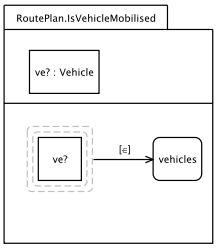
(b) Operation `RoutePlan.SelectRoutes`



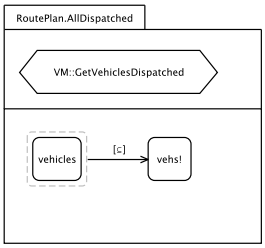
(c) Operation `RoutePlan.SelectVehicles`



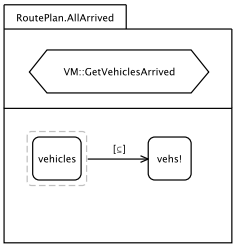
(d) Operation `RoutePlan.DeleteRoutes`



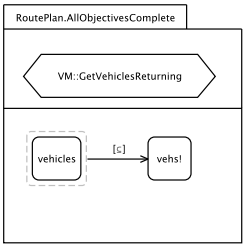
(e) Operation `RoutePlan.IsVehicleMobilised`



(f) Operation `RoutePlan.AllDispatched`



(g) Operation `RoutePlan.AllArrived`

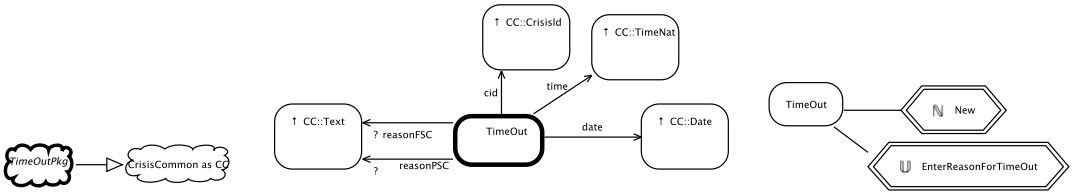


(h) Operation `RoutePlan.AllObjectivesComplete`

Figure 10: Local operations of blob `RoutePlan`

9 Package `TimeOutPkg`

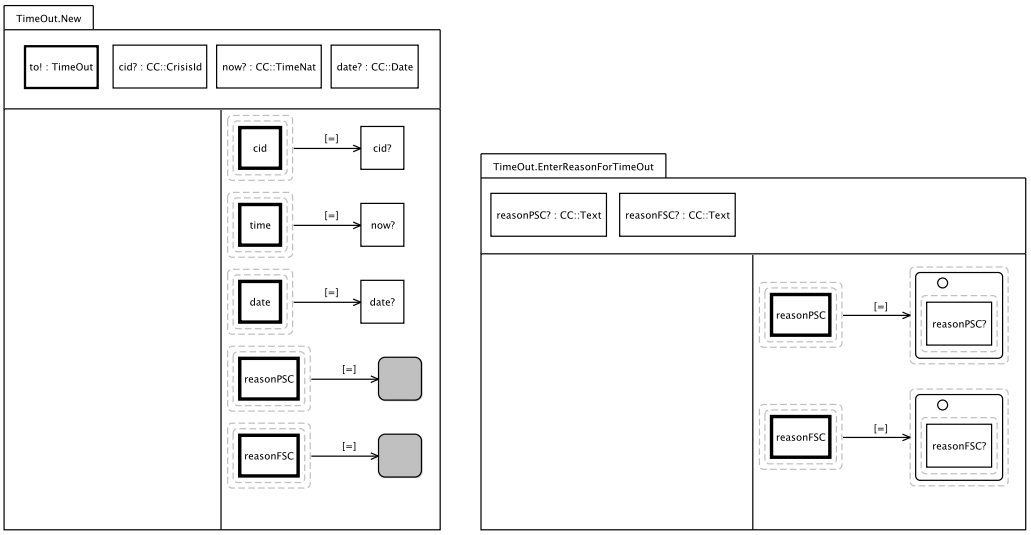
This package is responsible for managing the timeouts, which occur when the coordinator negotiate the details and the response to a crisis. A timeout is issued whenever the negotiation delay expires. Fig. 11a presents this package's PD, which sees the package `CommonCrisis` (identified with alias `CC`). The SD (Fig 11b) defines the class (or domain blob) `TimeOut`, which holds the information associated with a crisis as defined by the entry *Timeout Log* of the data dictionary in [1]. A `TimeOut` holds a crisis identifier (`cid`), a `time`, a `date` and two justifications for the reason for the timeout (`reasonPSC` and `reasonFSC`).



(a) PD of `TimeOutPkg` (b) SD of `TimeOutPkg` (c) BD of `TimeOutPkg`

Figure 11: PD, SD and BD of package `TimeOutPkg`

The package's BD (Fig. 11c) introduces operations for the class `TimeOut`.

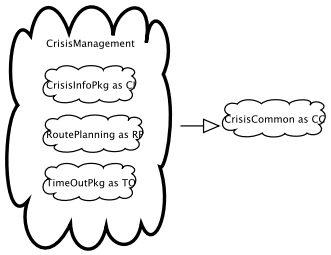


(a) Operation `TimeOut.New` (b) Operation `TimeOut.EnterReasonForTimeOut`

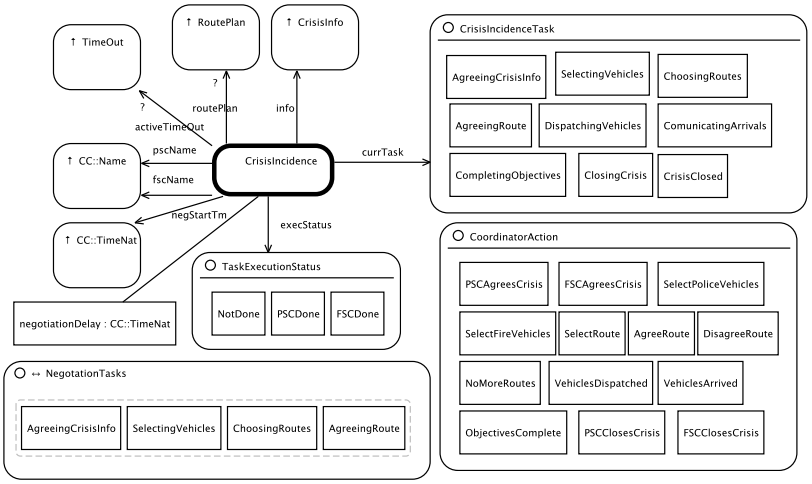
Figure 12: Local operations of blob `TimeOut`

10 Package `CrisisManagement`

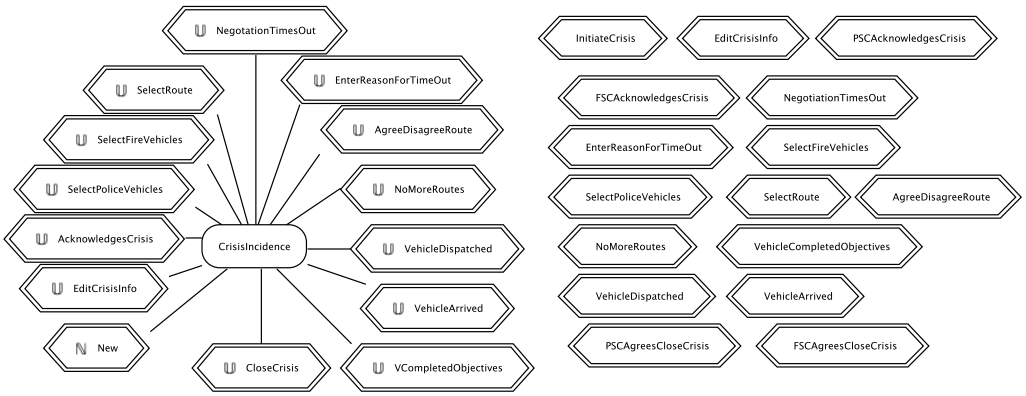
This package manages the coordination of the response to crisis as described by the main scenario of [1]. It is the package factoring the main problem-domain functionality of the bCMS. Fig. 13a presents this package's PD, which sees the package `CommonCrisis` (identified with alias `CC`) and incorporates the packages `CrisisInfoPkg` (alias `CI`), `RoutePlanning` (alias `RP`) and `TimeOutOkg` (alias `TO`). The SD (Fig 13b) defines the class (or domain blob) `CrisisIncidence`, which holds the information that is associated with a crisis response, and several other sets, such as `CrisisIncidenceTask` (describing different tasks of crisis coordination as described in the main scenario of [1]), and CoordinatorAction (describing the different actions that coordinators take for the resolution of a crisis). The package's BD (Fig. 13c) introduces several operations for the class `CrisisIncidence` and several other global operations. These operations represent actions that coordinator actors can take during the resolution of a crisis.



(a) PD of `CrisisManagement`

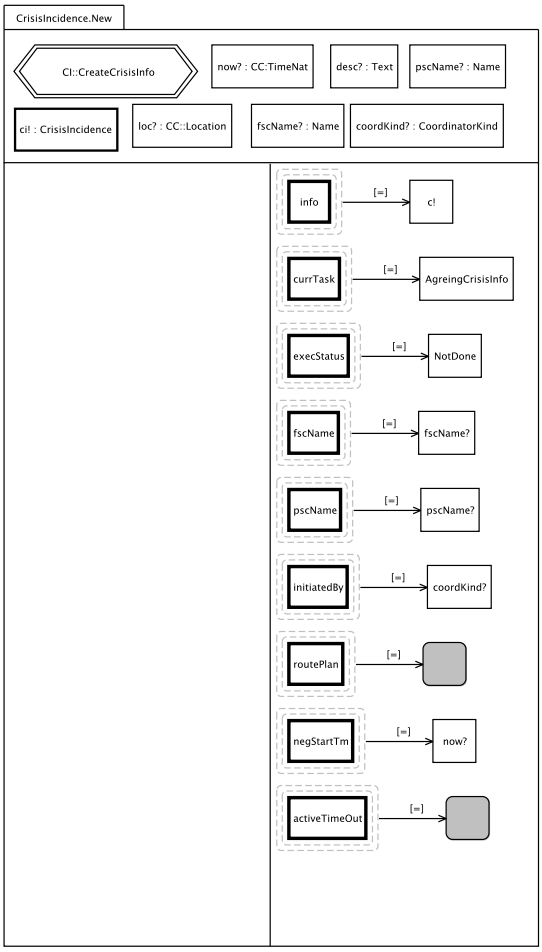


(b) SD of `CrisisManagement`

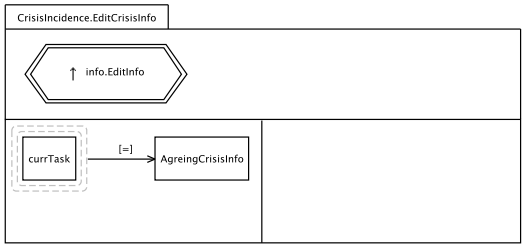


(c) BD of `CrisisManagement`

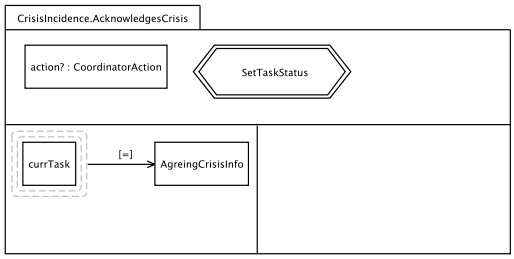
Figure 13: PD, SD and BD of package `CrisisManagement`



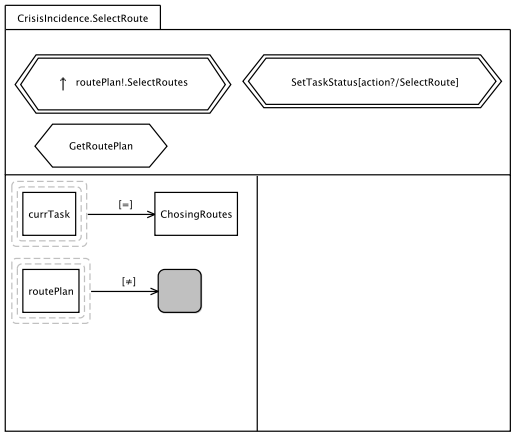
(a) Operation `CrisisIncidence.New`



(b) Operation `CrisisIncidence.EditCrisisInfo`

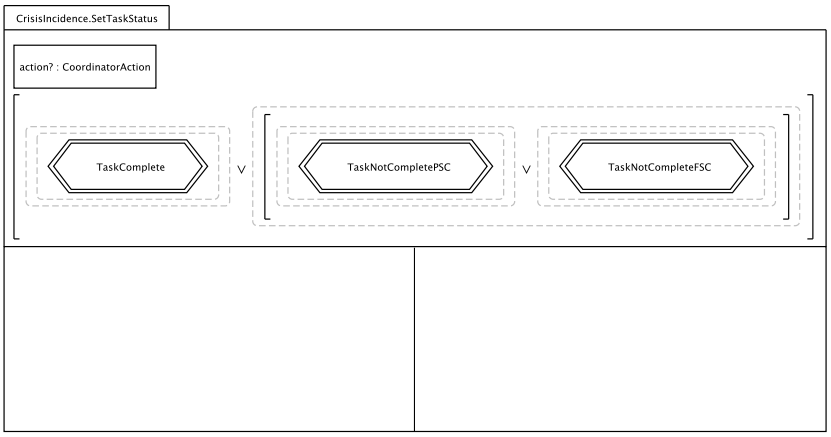


(c) Operation `CrisisIncidence.AcknowledgesCrisis`

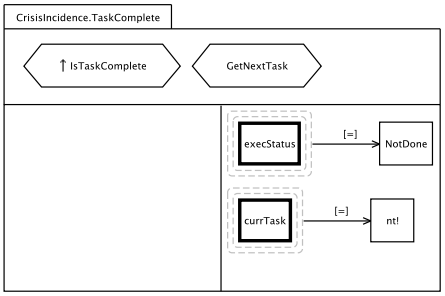


(d) Operation `CrisisIncidence.SelectRoute`

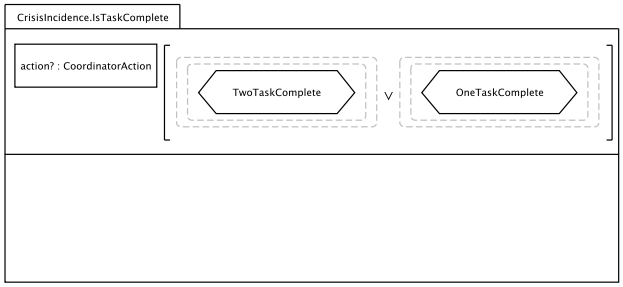
Figure 14: Local operations of blob `CrisisIncidence` (I)



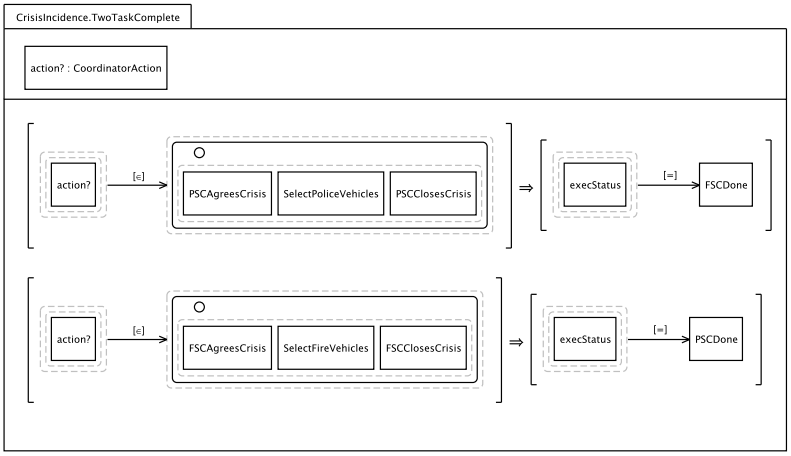
(a) Operation `CrisisIncidence.SetTaskStatus`



(b) Operation `CrisisIncidence.TaskComplete`

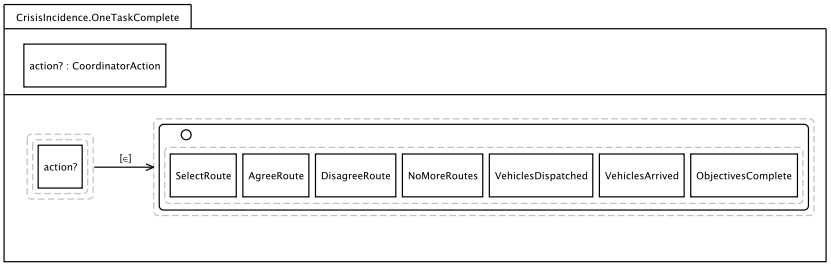


(c) Operation `CrisisIncidence.IsTaskComplete`

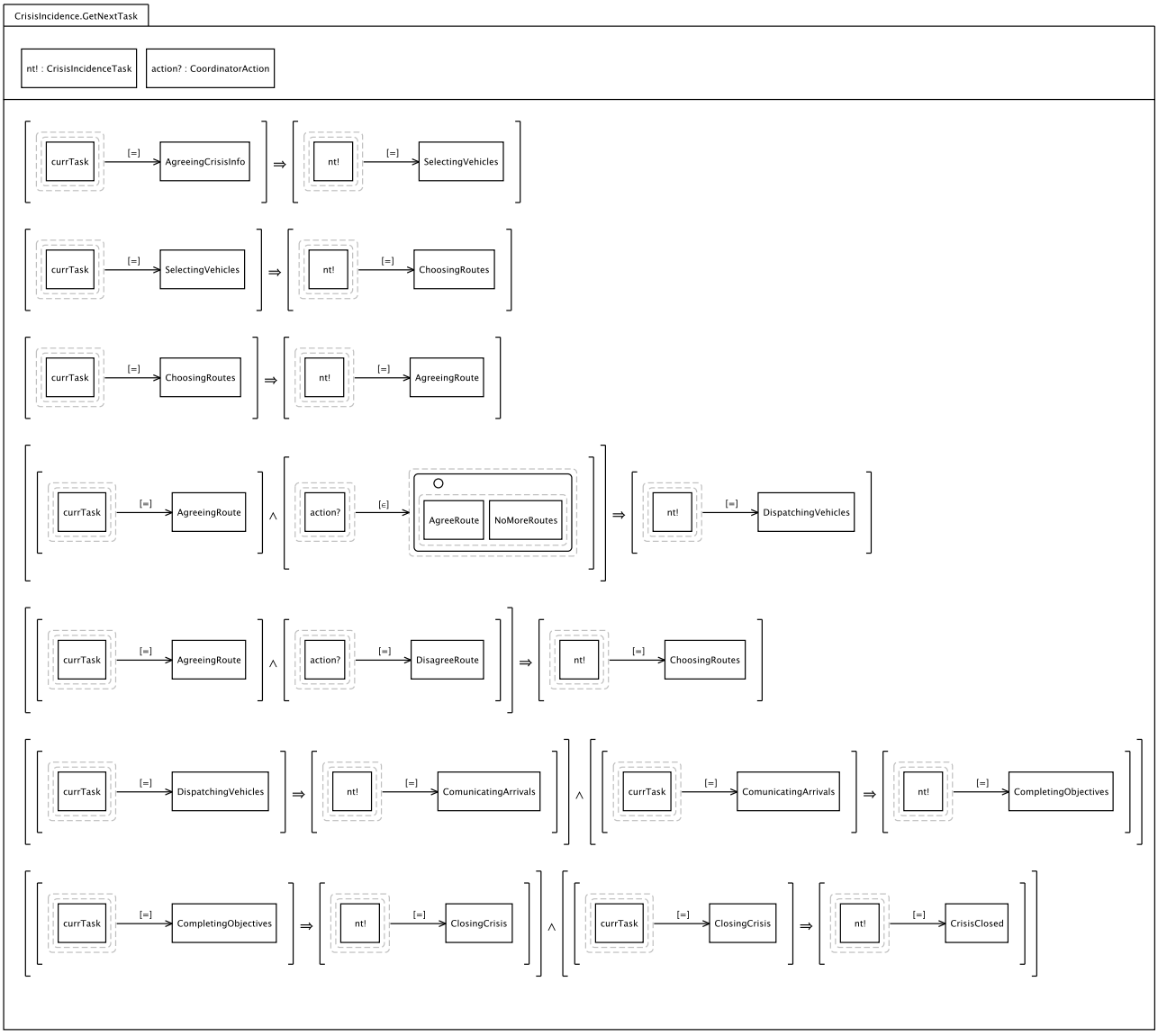


(d) Operation `CrisisIncidence.TwoTaskComplete`

Figure 15: Local operation `SetTaskStatus` of blob `CrisisIncidence` and sub-operations (I)

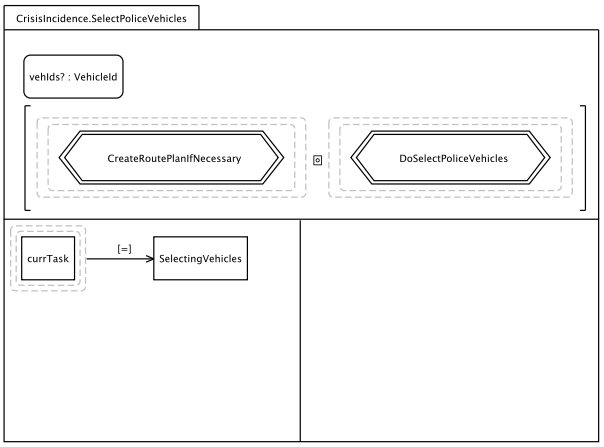


(a) Operation `CrisisIncidence.OneTaskComplete`

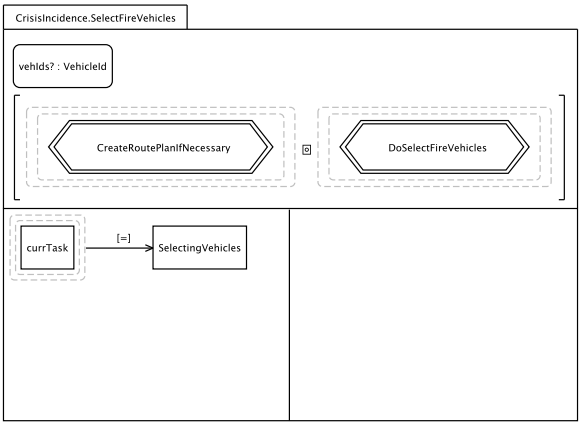


(b) Operation `CrisisIncidence.GetNextTask`

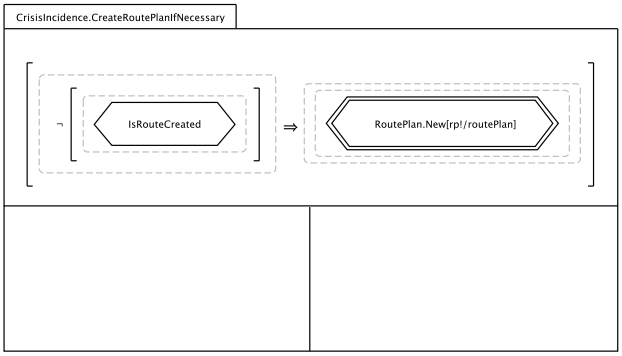
Figure 16: Local operation `SetTaskStatus` of blob `CrisisIncidence` and sub-operations (II)



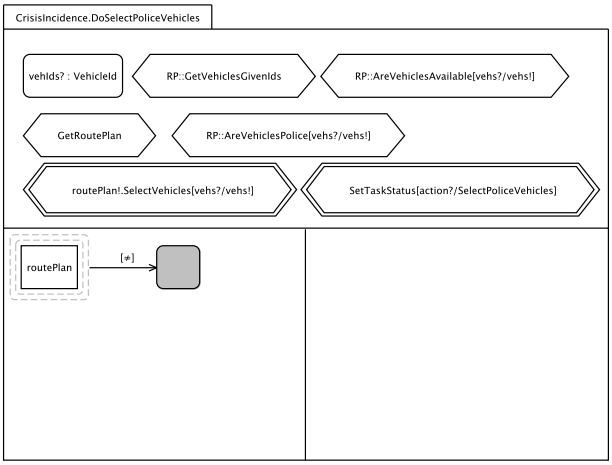
(a) Operation `CrisisIncidence.SelectPoliceVehicles`



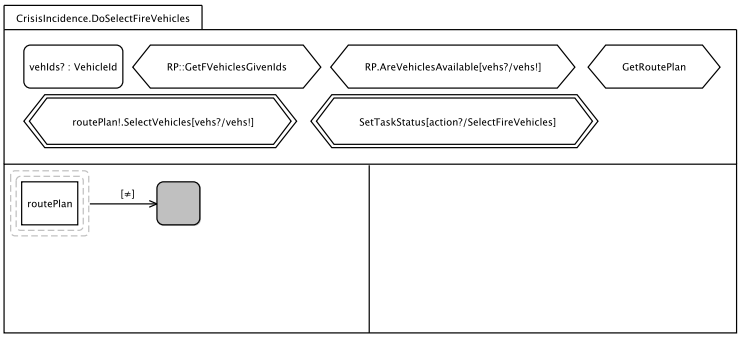
(b) Operation `CrisisIncidence.SelectFireVehicles`



(c) Operation `CrisisIncidence.CreateRoutePlanIfNecessary`

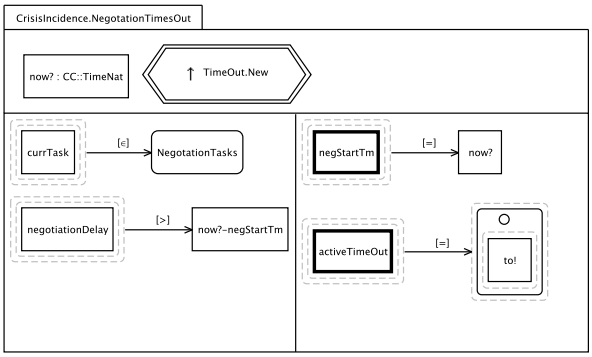


(d) Operation `CrisisIncidence.DoSelectPoliceVehicles`

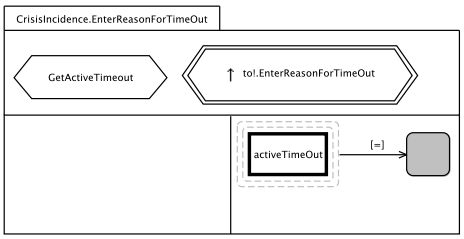


(e) Operation `CrisisIncidence.DoSelectFireVehicles`

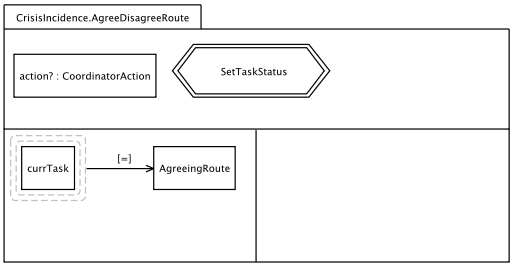
Figure 17: Local operations of blob `CrisisIncidence` (II)



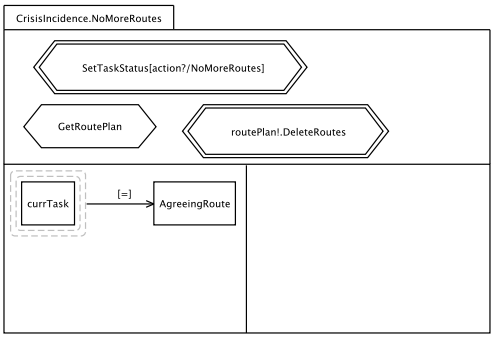
(a) Operation `CrisisIncidence.NegotationTimesOut`



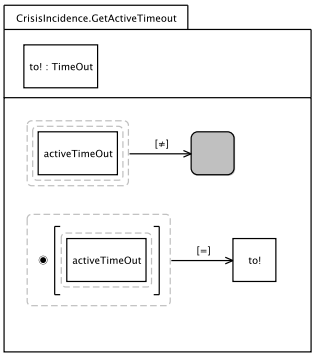
(b) Operation `CrisisIncidence.EnterReasonForTimeOut`



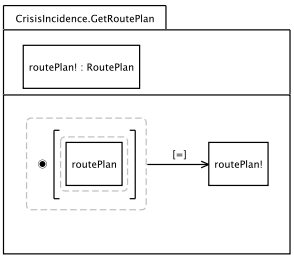
(c) Operation `CrisisIncidence.AgreeDisagreeRoute`



(d) Operation `CrisisIncidence.NoMoreRoutes`

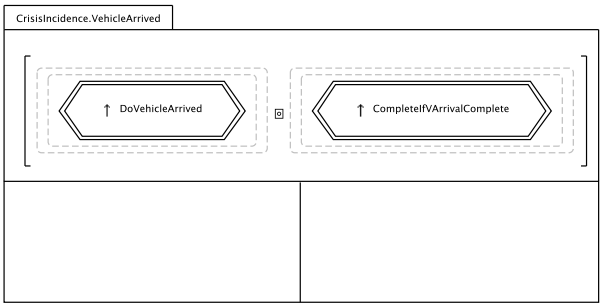


(e) Operation `CrisisIncidence.GetActiveTimeOut`

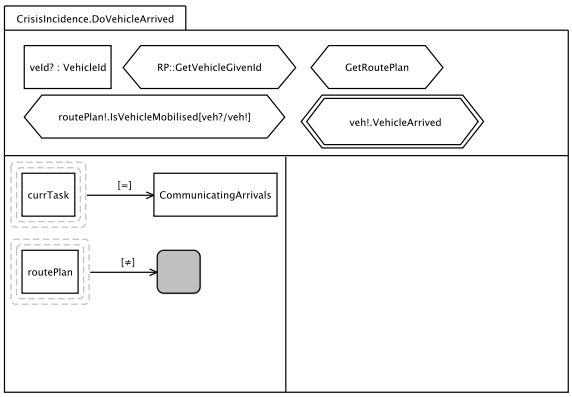


(f) Operation `CrisisIncidence.GetRoutePlan`

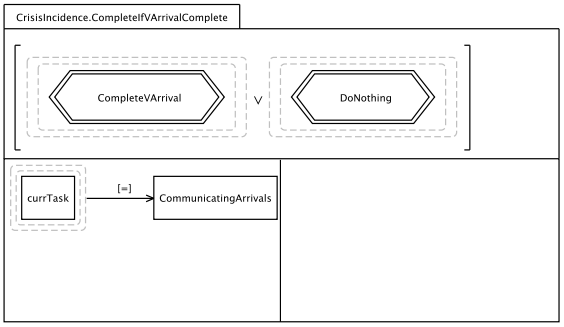
Figure 18: Local operations of blob `CrisisIncidence` (III)



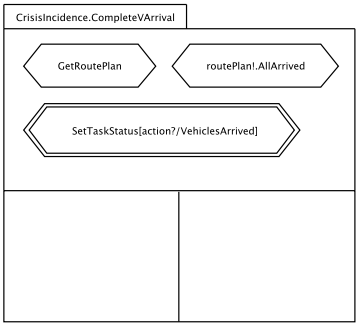
(a) Operation `CrisisIncidence.VehicleArrived`



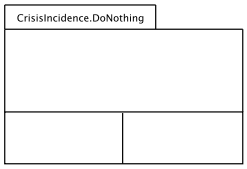
(b) Operation `CrisisIncidence.DoVehicleArrived`



(c) Operation `CrisisIncidence.CompleteIfVArrivalComplete`

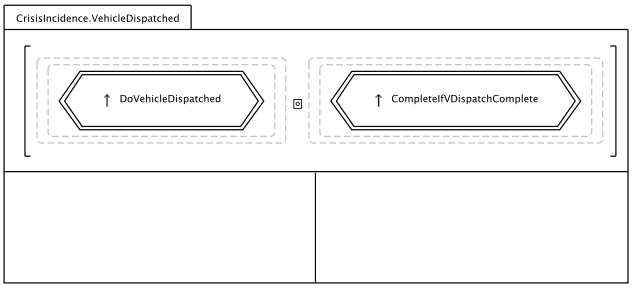


(d) Operation `CrisisIncidence.CompleteVArrival`

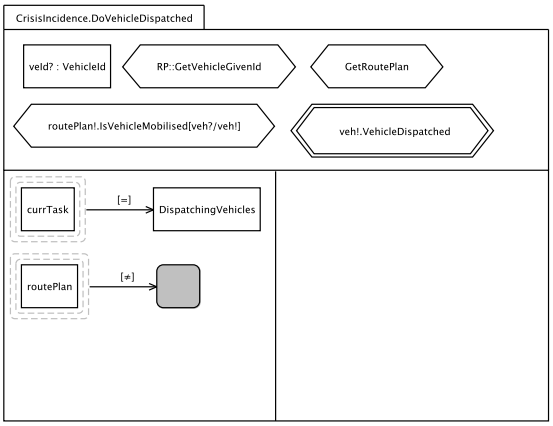


(e) Operation `CrisisIncidence.DoNothing`

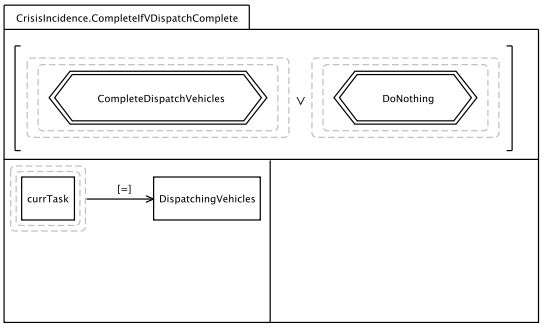
Figure 19: Local operations of blob `CrisisIncidence` (IV)



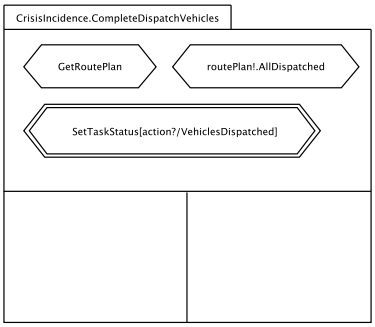
(a) Operation `CrisisIncidence.VehicleDispatched`



(b) Operation `CrisisIncidence.DoVehicleDispatched`

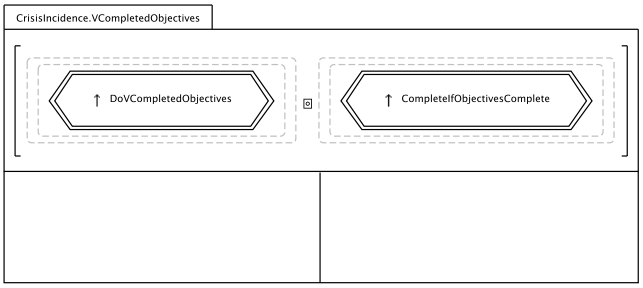


(c) Operation `CrisisIncidence.CompleteIfVDispatchComplete`

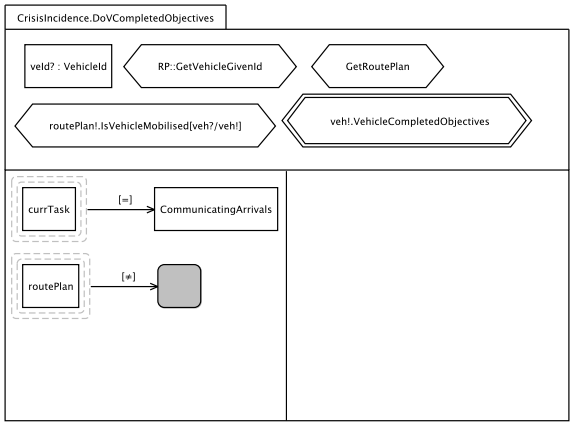


(d) Operation `CrisisIncidence.CompleteDispatchVehicles`

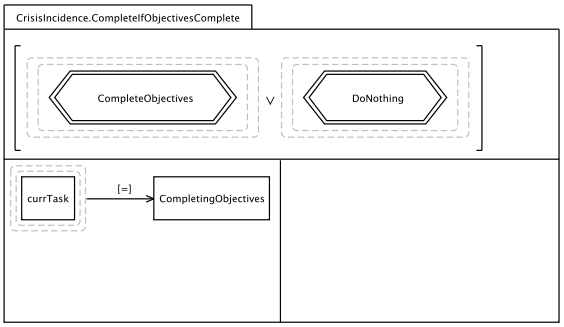
Figure 20: Local operations of blob `CrisisIncidence` (V)



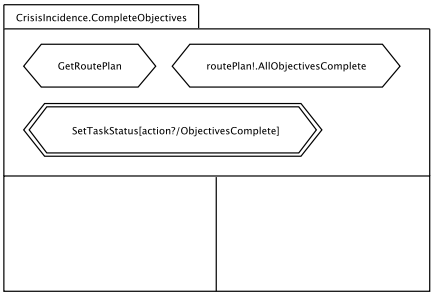
(a) Operation `CrisisIncidence.VCompletedObjectives`



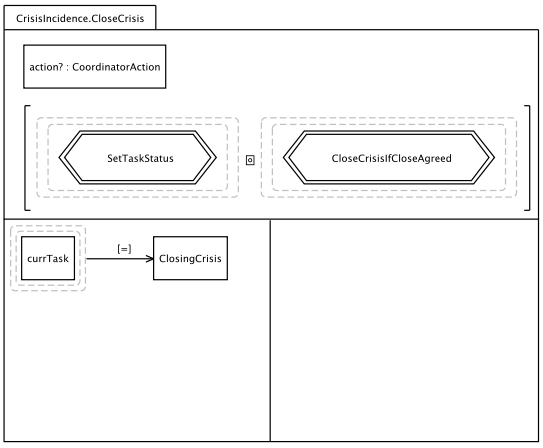
(b) Operation `CrisisIncidence.DoVCompletedObjectives`



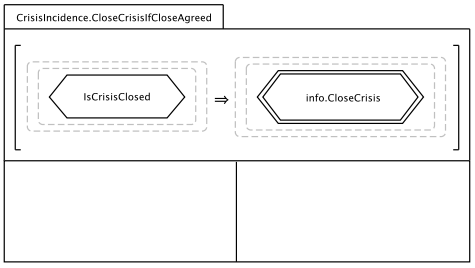
(c) Operation `CrisisIncidence.CompleteIfObjectivesComplete`



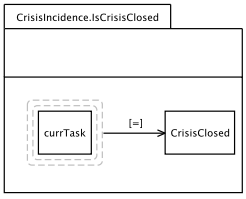
(d) Operation `CrisisIncidence.CompleteObjectives`



(e) Operation `CrisisIncidence.CloseCrisis`

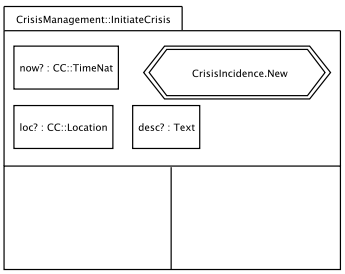


(f) Operation `CrisisIncidence.CloseCrisisIfCloseAgreed`

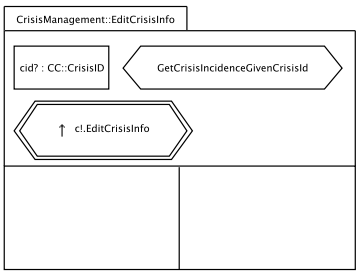


(g) Operation `CrisisIncidence.IsCrisisClosed`

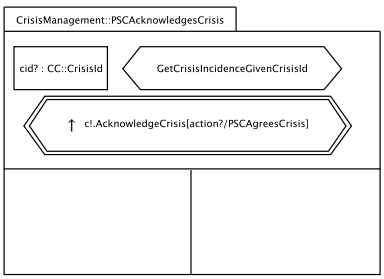
Figure 21: Local operations of blob `CrisisIncidence` (VI)



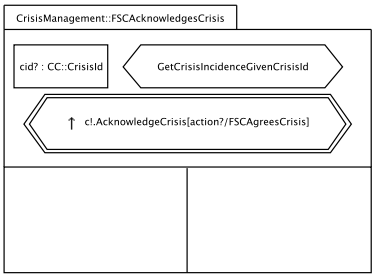
(a) Operation `InitiateCrisis`



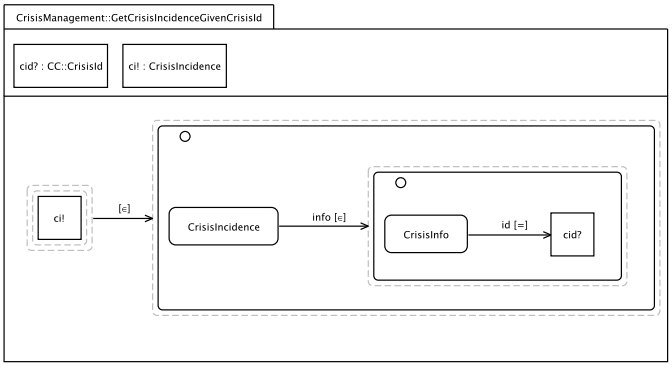
(b) Operation `EditCrisisInfo`



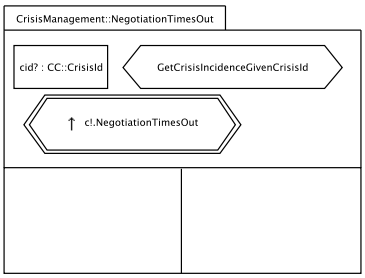
(c) Operation `PSCAcknowledgesCrisis`



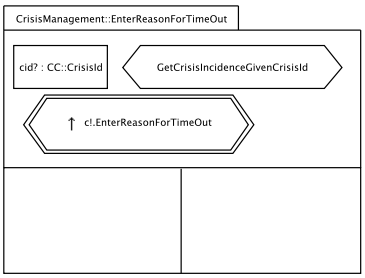
(d) Operation `FSCAcknowledgesCrisis`



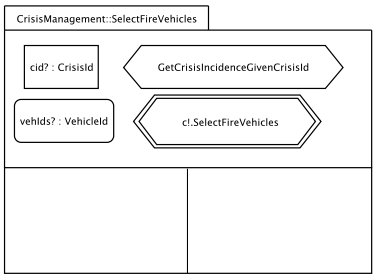
(e) Operation `GetCrisisIncidenceGivenCrisisId`



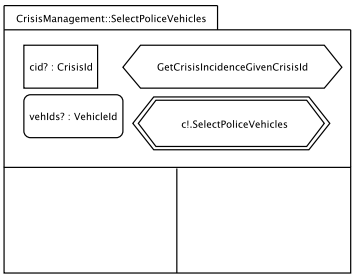
(f) Operation `NegotiationTimesOut`



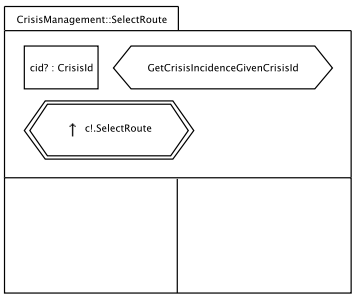
(g) Operation `EnterReasonForTimeOut`



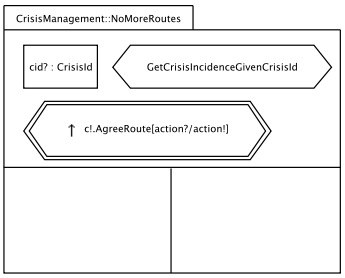
(h) Operation `SelectFireVehicles`



(i) Operation `SelectPoliceVehicles`

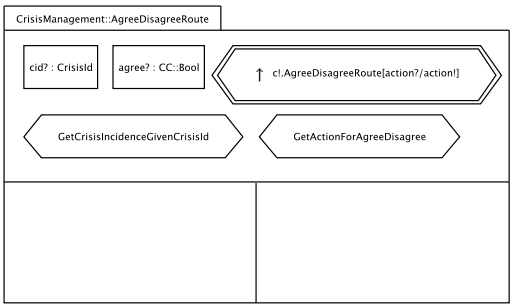


(j) Operation `SelectRoute`

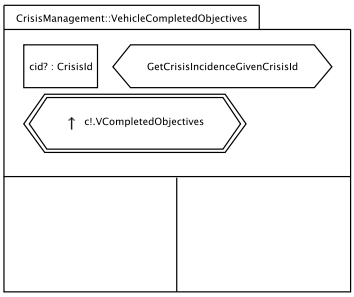


(k) Operation `NoMoreRoutes`

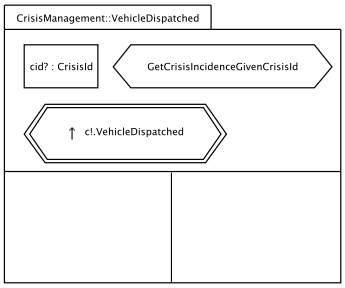
Figure 22: Global operations of package `CrisisManagement` (I)



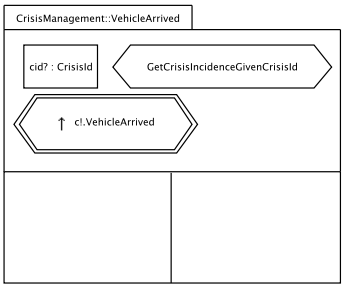
(a) Operation `AgreeDisagreeRoute`



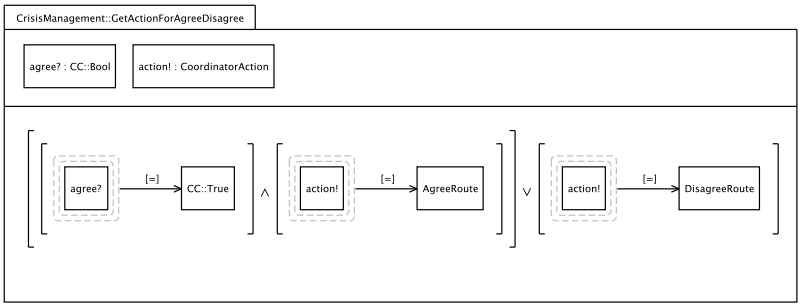
(b) Operation `VehicleCompletedObjectives`



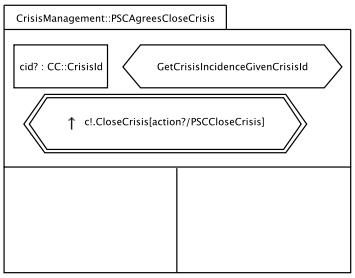
(c) Operation `VehicleDispatched`



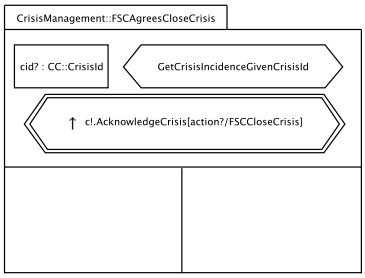
(d) Operation `VehicleArrived`



(e) Operation `GetActionForAgreeDisagree`



(f) Operation `PSCAgreesCloseCrisis`

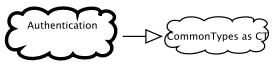


(g) Operation `FSCAgreesCloseCrisis`

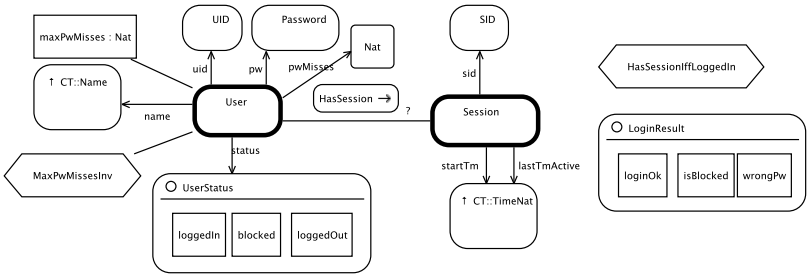
Figure 23: Global operations of package `CrisisManagement` (II)

11 Package `Authentication`

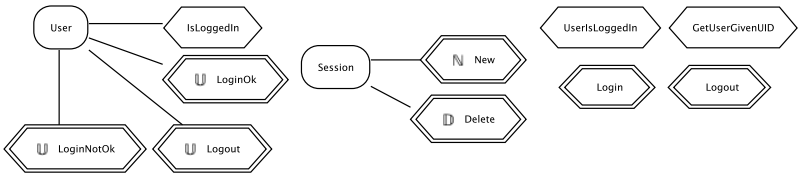
This package factors the authentication security concern. Fig. 24a presents this package's PD, which sees the package `CommonTypes` (identified with alias `CT`). The SD (Fig 24b) defines the classes (or domain blobs) `User`, which holds the information associated with system users, and `Session`, which holds information regarding system sessions open in the system; other blobs introduce sets to represent authentication sensitive information (such as `UserStatus`); the association `HasSession` relates users and their system sessions. The package's BD (Fig. 24c) introduces several operations for the classes `User` and `Session` and several other global operations.



(a) PD of `Authentication`

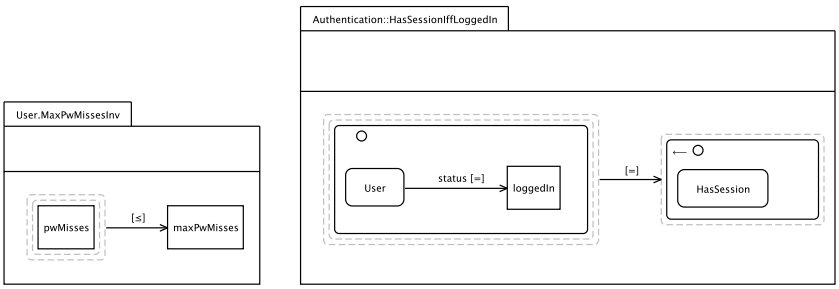


(b) SD of `Authentication`



(c) BD of `Authentication`

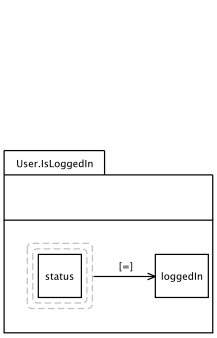
Figure 24: PD, SD and BD of package `Authentication`



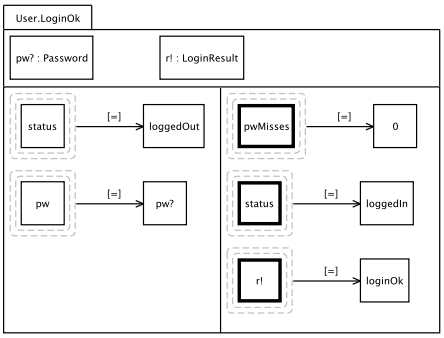
(a) AD of local invariant `User.MaxPwMissesInv`

(b) AD of global invariant `HasSessionIffLoggedIn`

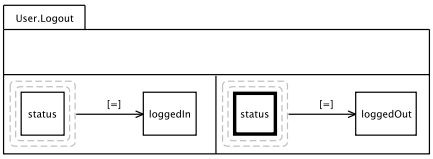
Figure 25: Invariants of `Authentication`



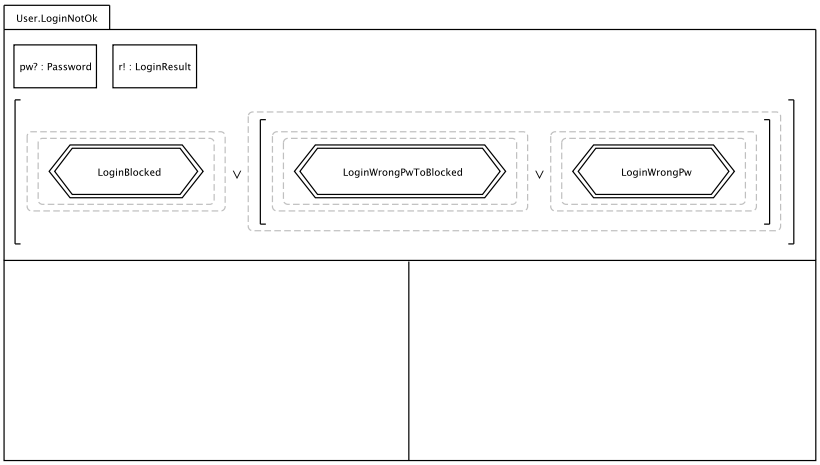
(a) Operation `User.IsLoggedIn`



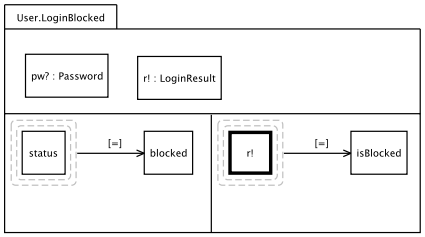
(b) Operation `User.LoginOk`



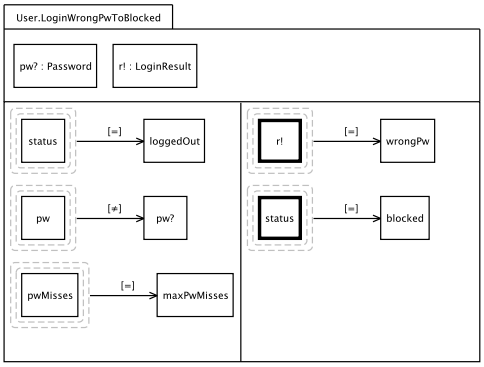
(c) Operation `User.Logout`



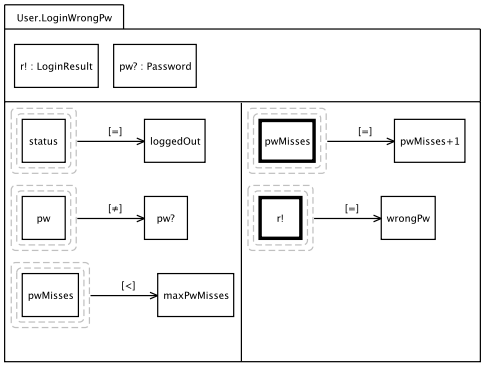
(d) Operation `User.LoginNotOk`



(e) Operation `User.LoginBlocked`

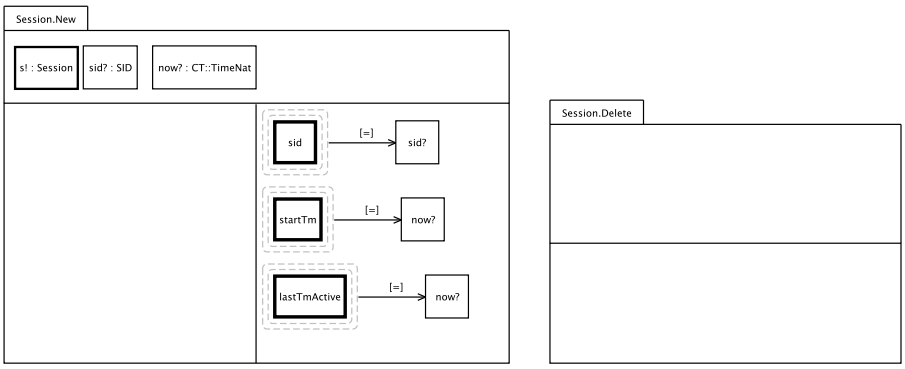


(f) Operation `User.LoginWrongPwToBlocked`



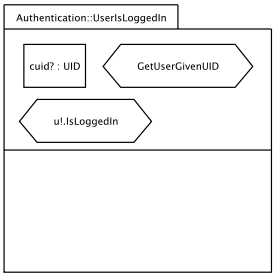
(g) Operation `User.LoginWrongPw`

Figure 26: Local operations of blob `User`

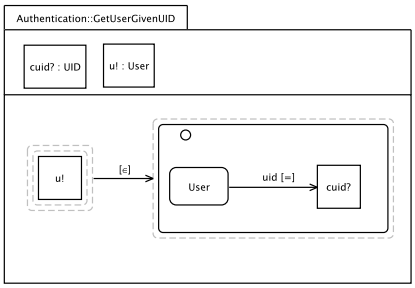


(a) Operation `Session.New` (b) Operation `Session.Delete`

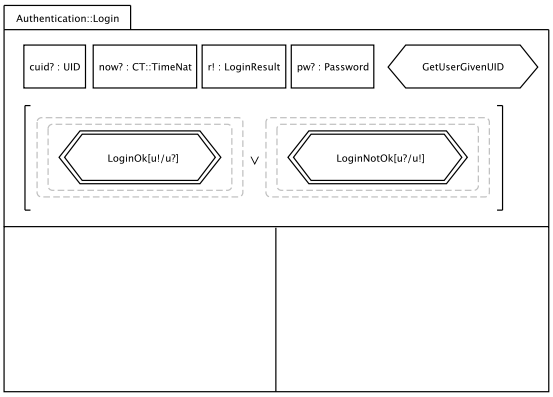
Figure 27: Local operations of blob `Session`



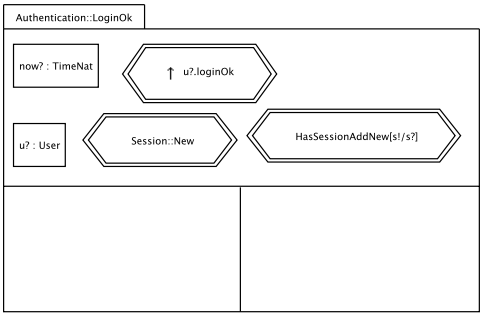
(a) Operation `UserIsLoggedIn`



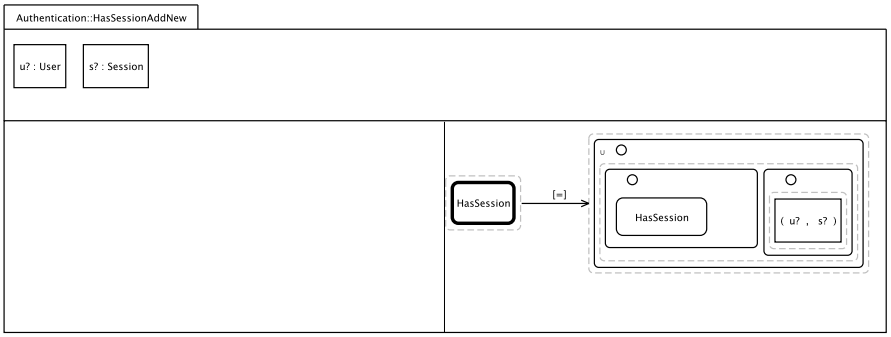
(b) Operation `GetUserGivenUID`



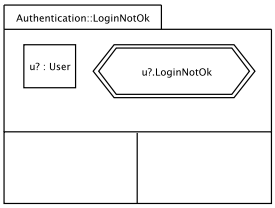
(c) Operation `Login`



(d) Operation `LoginOk`

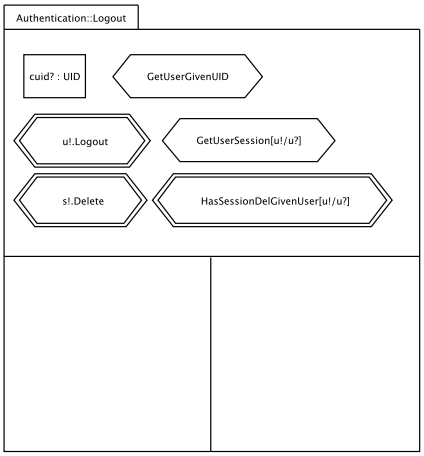


(e) Operation `HasSessionAddNew`

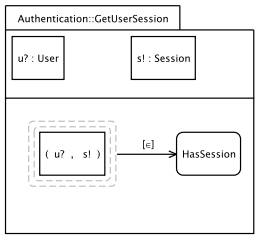


(f) Operation `LoginNotOk`

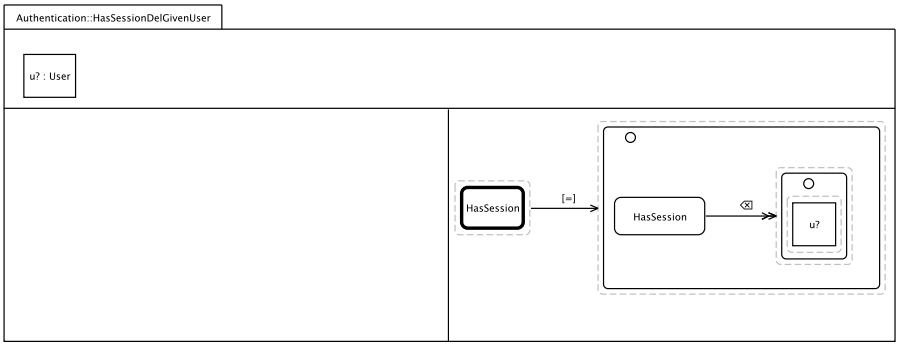
Figure 28: Global operations of package `Authentication` (I)



(a) Operation `Logout`



(b) Operation `GetUserSession`

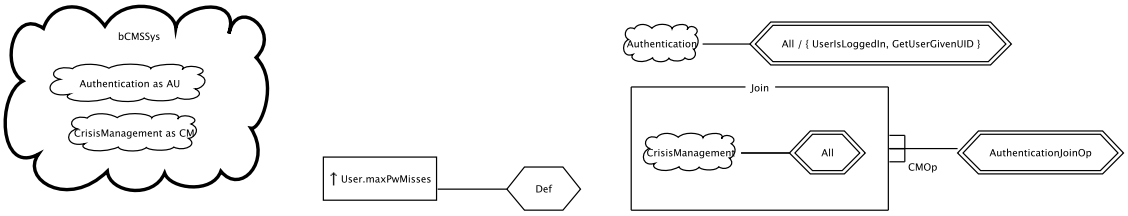


(c) Operation `HasSessionDelGivenUser`

Figure 29: Global operations of package `Authentication` (II)

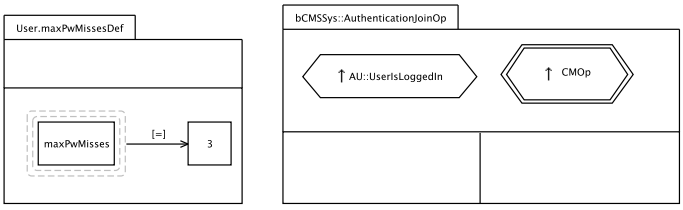
12 Package `bCMSSys`

This package represents the overall bCMS system that puts together crisis management and authentication. Fig. 30a presents this package's PD, which incorporates package `Authentication` (alias `AU`) and `CrisisManagement` (alias `CM`). The package SD (Fig. 30b) defines the value of the constant `maxPwMisses` of `Authentication`. The BD (Fig. 30c) defines the operations of the package, which are built as a composition of certain operations from `Authentication` and the operations from `CrisisManagement`, which are added the `authentication` concern through the VCL join composition.



(a) PD of `bCMSSys` (b) SD of `bCMSSys` (c) BD of `bCMSSys`

Figure 30: PD, SD and BD of package `bCMSSys`



(a) AD defining constant `maxPwMisses` (b) Join contract for authentication

Figure 31: Definition of constant `User.maxPwMisses` and join contract

References

[1] Alfredo Capozuca, Betty H. C. Cheng, Nicolas Guelfi, Paul Istoan, and Gunter Mussbacher. bCMS – requirements definition document. Technical report, 2012.

[2] Nuno Amálio and Pierre Kelsen. Modular design by contract visually and formally using VCL. In *VL/HCC 2010*, 2010.

[3] Nuno Amálio, Pierre Kelsen, Qin Ma, and Christian Glodt. Using VCL as an aspect-oriented approach to requirements modelling. *TAOSD*, VII:151–199, 2010.

[4] Nuno Amálio, Christian Glodt, and Pierre Kelsen. Building VCL models and automatically generating Z specifications from them. In *FM 2011*, volume 6664 of *LNCS*, pages 149–153. Springer, 2011.