OO-SPL modelling of the focused case study

Afredo Capozucca[1], Betty H.C. Cheng[2], Nicolas Guelfi[1], and Paul Istoan[1,3]

[1]LASSY Research Team, University of Luxembourg, Luxembourg
[2]Department of Computer Science and Engineering, Michigan State University, USA
[3]ISC Department, CRP Gabriel Lippmann, Luxembourg

September 21, 2011

Abstract

This document overviews an object-oriented (OO) modeling approach and a software product line (SPL) methodology used to model the Barbados Crash Management System Product Line (referred to as bCMS-SPL), as well as for a *reference variant* of such a product line (referred to as bCMS). The approaches and the modeling languages used have been chosen in order to comply with widely used practices and/or (de facto) standards.

The starting point for this effort is a set of requirements described in a brief requirements document [5]. Products of bCMS-SPL are intended to support distributed crisis management by police and fire personnel for automotive accidents on public roadways. While police and fire personnel have complementary responsibilities to be done concurrently, these efforts need to be coordinated in order to ensure efficient and effective management of a given crisis. As such, the bCMS-SPL models focus on the functionality of the Police Station Coordinator (PS coordinator) and the Fire Station Coordinator (FS coordinator) and their interactions. The scope of the bCMS-SPL starts with the notification of a crisis to the PS coordinator and FS coordinator concludes at the point when all fire and police personnel have been released from the given crisis. The assumption is that high-level requirements have been gathered and stated in the form of use cases for the bCMS. The modeling of the bCMS and bCMS-SPL target the late requirements briefly overviews the refinement to system architecture.

The OO modeling approach provides structural context information and behavior information to be used to provide a late requirements specification of the bCMS. The structural context information is captured in terms of a domain model using the class diagram notation, including a data dictionary. The behavior information is described in terms of sequence diagrams that model specific scenarios between the key elements of a system, and interacting state diagrams that describe the behavior of a given element as it behaves across multiple scenarios.

In order to create a SPL model for the bCMS-SPL, we use model elements from the bCMS late requirements OO model and the variabilities described in the high-level requirements document [5]. The SPL model comprises a feature diagram and OO model fragments, each of which describes the structural and behavioral

information for a given variation point. These OO model fragments are described in terms of the same UML diagrams as those used to model the bCMS. The key difference is that inheritance and stereotypes are used to capture the SPL concerns. The SPL models can then be used to derive the complete OO late requirement specification of each planned SPL variant.

1 Assumptions

The following are those assumptions we made when modeling the bCMS and bCMS-SPL.

- In the Sequence Diagrams, messages between actors are synchronous and instantaneous.
- It is out of the scope of the system to propose a route plan for the police cars.
- The PS coordinator proposes at least one route plan to the FS coordinator.
- As a police car/fire truck can be recalled in case the crisis is less severe than expected, it is assumed that at least one police car/fire truck has arrived to the crisis location. It is up to the police officers/fireman that have first arrived to check the severity of the crisis, and notify to its coordinator. The coordinator, depending on the provided information, may recall any number of vehicles.
- The case in which a police car/fire truck does not reach its destination within the ETA because of traffic jams or blocked routes (i.e. alternative scenario *5.b* in [5]) it is considered as a particular case in which the vehicle is delayed because it broke down (i.e alternative scenario *5.a* in [5]). Therefore, a route plan that was already agreed between the PS coordinator and FS coordinator will not be rescheduled due to the delay of a vehicle.

2 OO modeling approach

During the late requirements phase, the structural view describes the key elements of the system and their relationships, while the behavioral view describes the high-level behavior of those elements. The class diagram notation is used to construct a domain model of the system that describes the elements of the system and the portion of the environment with which those elements interact. For example, while the PS coordinator and FS coordinator are (some of) the key elements of the bCMS system, the PS coordinator needs to interact with individual police units that are also modeled, but their detailed functionality is beyond the scope of this modeling effort. Therefore, the individual police unit is included in the domain model to provide context for the system elements serving as an entity to receive and send information to the PS coordinator. For each class element in the domain model, we include a data dictionary entry that briefly describes

the purpose of the class, attributes, and operations. It also includes descriptions of any relationships in which the class participates (e.g., aggregation, association, etc.).

Next, we describe the key scenarios outlined in the requirements document using sequence diagrams. Each sequence diagram contains interactions between object instances of the classes (from the domain model) using messages that have been declared as operations in the domain model for the respective class. Messages may also be predicated by boolean guards involving attributes (also defined in the domain model for the respective class element). Then by focusing on individual object instances across all the scenarios, we can collect all the relevant behavior for a given class to create its state diagram. Therefore, a state diagram is created for each key class that includes its behavior across all the modeled scenarios. As such, as a means of validation, we should be able to trace the behavior of a given scenario by traversing the state diagrams for all the objects involved in the scenario. But the state diagram should also include complementary transitions from those reference in the sequence diagrams in order to provide a comprehensive description of the behavior of a given class. For example, there might be a scenario that describes the interaction between the PS coordinator and a police unit that is still at the police station when a crisis has been identified. Then there should also be behavior added to the PS coordinator state diagram that describes what should happen when a police unit is not at the police station (e.g., different route, etc.).

In general, we should be able to check for consistency between the structural models and the behavior models. For example, all messages in the sequence diagrams should be defined as operation names in the class diagram and should appear as triggers or actions in the state diagrams. Moreover, any guards on the messages in sequence diagrams and state diagrams should involve attributes that have been defined in the class diagram.

During the high-level design, the structural view describes the software architecture and the behavior of the software components of the architectural elements. As part of the OO process, high-level design involves refining the information from the requirements stage to include design-level information. First, a decision needs to be made as to what software architecture should be used to describe the target system; this decision process refers to the structural design of the system. A number of architectural styles have been proposed, each of which facilitates different types of interactions between the subsystems of a system. In the case of the bCMS system, a variation of the peer to peer (P2P) architectural style is used. In this case, the PSC System and FSC System, each can act as a server and client, depending upon the specific situation. The key elements may be refined to include additional subcomponents that provide aggregate behavior. For the behavior portion of the high-level design, the state diagrams of the key elements may be refined to include more implementation details. And the subcomponents of the key elements are described in terms of state diagrams. Additional sequence diagrams can be created to capture the additional behavior details. In addition, we can add sequence diagrams to capture exceptional cases. In short, all sequence diagrams should be validated against the collection of state diagrams.

2.1 Domain Model

This section introduces the key elements of the system captured in the domain model (see Figure 1), both the bCMS-SPL elements, as well as physical and environmental elements that interact with bCMS-SPL and/or provide context for its behavior. The key software elements of the domain model have been described in a data dictionary. For brevity, we only include the descriptions of the key operations of the FSC System, since many of the operations have analogous counterparts in PSC System. Furthermore, the FS Coordinator and PS Coordinator are humans interfacing with the respective System elements, and therefore are not described in detail for this preliminary submission. The data dictionary entries includes the attributes and operations of each class element, as well as any relationships in which the element participates (e.g., aggregation, association, etc.). Due to its length, we include the data dictionary as an appendix, so that it may be reviewed as one entity, rather than breaking up into several figures.

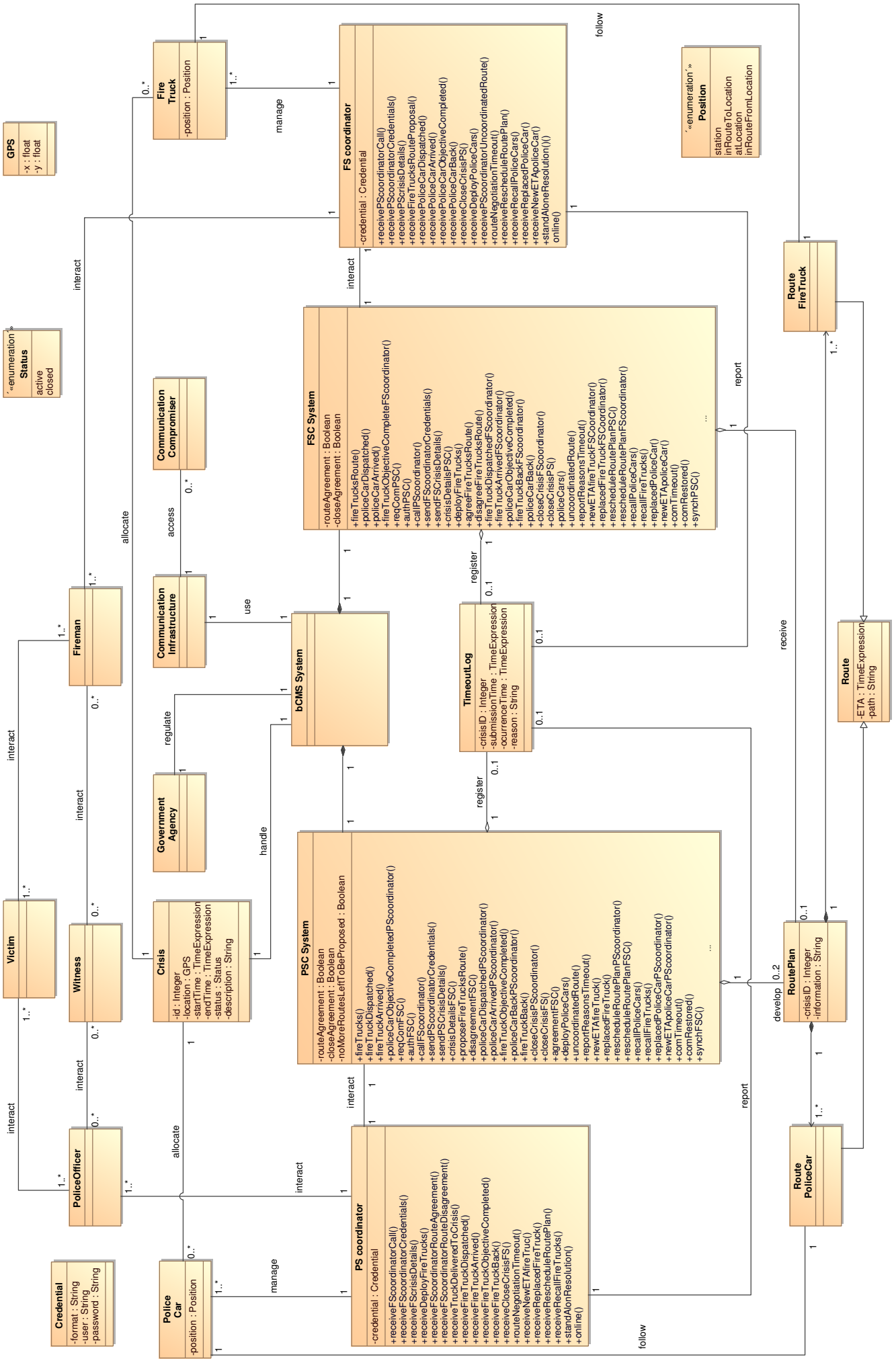


Figure 1: Domain model.

2.2 Scenarios

This section overviews the key functional scenarios that were enumerated in the requirements document [5]. For reader convenience, we include the list of scenarios from the requirements document as follows. Each of the steps of the following scenario descriptions have been modeled by means of sequence diagrams (see Figures 2- 11). Each of these sequence diagrams are detailed next.

Main Scenarios:

1. PSC and FSC establish communication and identification of coordinators.

2. PSC and FSC exchange crisis details.

3. PSC and FSC develop a coordinated route plan in a timely fashion for number of vehicles to be deployed to specific locations with respective ETAs.

3.1. PSC and FSC state their respective number of fire trucks and police vehicle to deploy.

3.2. PSC proposes one route for fire trucks and one route for police vehicles to reach crisis site.

3.3. FSC agrees to route.

4. PSC and FSC communicate to each other that their respective vehicles have been dispatched according to plan (per vehicle).

5. PSC and FSC communicate to each other their arrival (per vehicle) at targeted locations.

6. PSC and FSC communicate to each other completion (per vehicle) of their respective objectives.

7. PSC and FSC agree to close the crisis.

Alternative and Exceptional Scenarios:

At step 3 when the duration of the negotiation exceeds a predefined limit:

3.a1. A timeout is recorded in the system.

3.a2. PSC and FSC are alerted that a timeout has occurred for completing the negotiation.

3.a3. PSC and FSC are allowed to continue with the sub-step of step 3 where the timeout occurred.

3.a4. In parallel to 3.a3, PSC and FSC report the reason for timeout.

At step 3.3 when the FSC disagrees with the proposed route:

3.3.a1. The PSC removes the proposed route from the possible routes.

3.3.a2. Continue with step 3.2.

At step 3.3.a2 when there is no more route left to be proposed:

3.3.a2.a1. The PSC informs the FSC that the route will not be coordinated and that updates of vehicle locations and crisis details are still to be exchanged.

3.3.a2.a2. Continue with step 4.

At step 5 when a police vehicle/fire truck does not reach its destination within the ETA because of vehicle break down:

5.a1. The PSC/FSC informs the other coordinator of the new ETA and, if necessary, that a replacement vehicle is on its way.

5.a2. Continue with step 5.

At step 5 when a police vehicle/fire truck does not reach its destination within the ETA because of traffic or blocked routes:

5.b1. Continue with step 3.

At step 5 when the crisis is more severe than expected:

5.c1. Continue with step 3.

At step 5 when the crisis is less severe than expected:

5.d1. The PSC/FSC informs the other coordinator of recall of one or more police vehicles/fire trucks, respectively.

5.d2. Continue with step 5.

At any step M when communication is not available:

M.a1. PSC and FSC continue to address the crisis individually, and both will coordinate through their personnel once their personnel have reached the crisis site (this resolution is out of scope for bCMS).

At any step N when communication has been restored after a period of unavailable communication:

N.a1. If the crisis has been resolved (i.e., the objectives of all vehicles have been reached), then continue with step 7.

N.a2. If communication between PSC and FSC has not yet been established (step 1 has not yet been reached), then continue with step 1.

N.a3. If the route agreement has been reached (the use case is between step 4 and 6, inclusive), then exchange information on routes established for police and fire, location of vehicles, and status of crisis and for each vehicle continue with step 4, 5, or 6 depending on the location of a vehicle.

N.a4. If the route agreement has not been reached and the time limit for the route negotiation has not yet expired (the use case is between step 2 and 3.2, inclusive), then continue with step N.

N.a5. If the route agreement has not been reached and the time limit for the route has expired (the use case is between step 3.1 and 3.2, inclusive), then exchange information on routes established for police and fire, location of vehicles, and status of crisis and for each vehicle continue with step 4, 5, or 6 depending on the location of a vehicle.

2.2.1 Steps 1 and 2

The sequence diagram shown in Figure 2 models the establishment of the communication between the coordinators and their respective identification (step **1**), and the information exchange about the crisis details (step **2**). The sequence diagram starts with an alternative block to capture the non-determinism about who (i.e., either the PS coordinator or FS coordinator) would send the first request to establish the communication. As we assume that messages between actors are synchronous and instantaneous, is enough a peer receives a message to consider that the communication is established.

The following the alternative, comes a parallel block, which is used to model the different possibilities in which the authentication can be done: either the PS coordinator sends his credential first, or the FS coordinator does so. In any case, both behaviors are possible.

The last block is also a parallel block, and it is used in the same manner as the previous one, but to model the way in which the coordinators exchange the details about the crisis.

2.2.2 Steps 3

The sequence diagram on Figure 3 models the entire sequence of interactions for step 3 of the scenario, including its different alternatives. The sequence diagram starts by modeling the way in which the coordinators state their respective number of vehicles to deploy at the crisis location (step 3.1). It follows with a parallel block. The first part of this parallel block contains the messages the PS coordinator and FS coordinator may exchange to achieve an agreement about the route plan that allows fire trucks to arrive to the crisis location. The PS coordinator will keeping proposing to the FS coordinator a rout plan for the fire trucks until either the FS coordinator agrees to the proposal, or the PS coordinator has no more propositions to offer (alternative scenario **3.3.a2**).

The second part of the parallel block is used to capture the case in which the negotiation of the route plan between the PS coordinator and FS coordinator exceeds a predefined time (alternative scenario **3.a**). In that case, each coordinator, after being notified about the time out by the message *routeNegotiationTimeout*, he must fulfill a report explaining the reason why such a timeout was reached.

2.2.3 Steps 4

Figure 4 shows the sequence diagram that models the step in which the coordinators notify one another that their respective vehicles have been dispatched. Since the notification is done for each vehicle, and OCL [16] expression in used to describe the predicate used by the loop block. This predicate says that the messages enclosed in the parallel block are exchanged while the number of dispatched police cars, plus the number of fire trucks is lower than the number of requested police cars, plus the number of requested fire trucks. The number of requested police cars (resp. fire trucks) is known by counting the number of existing instances of type PoliceCar are bound to the Crisis by the association *allocate*. On the other hand, the number of dispatched police cars (resp. for fire

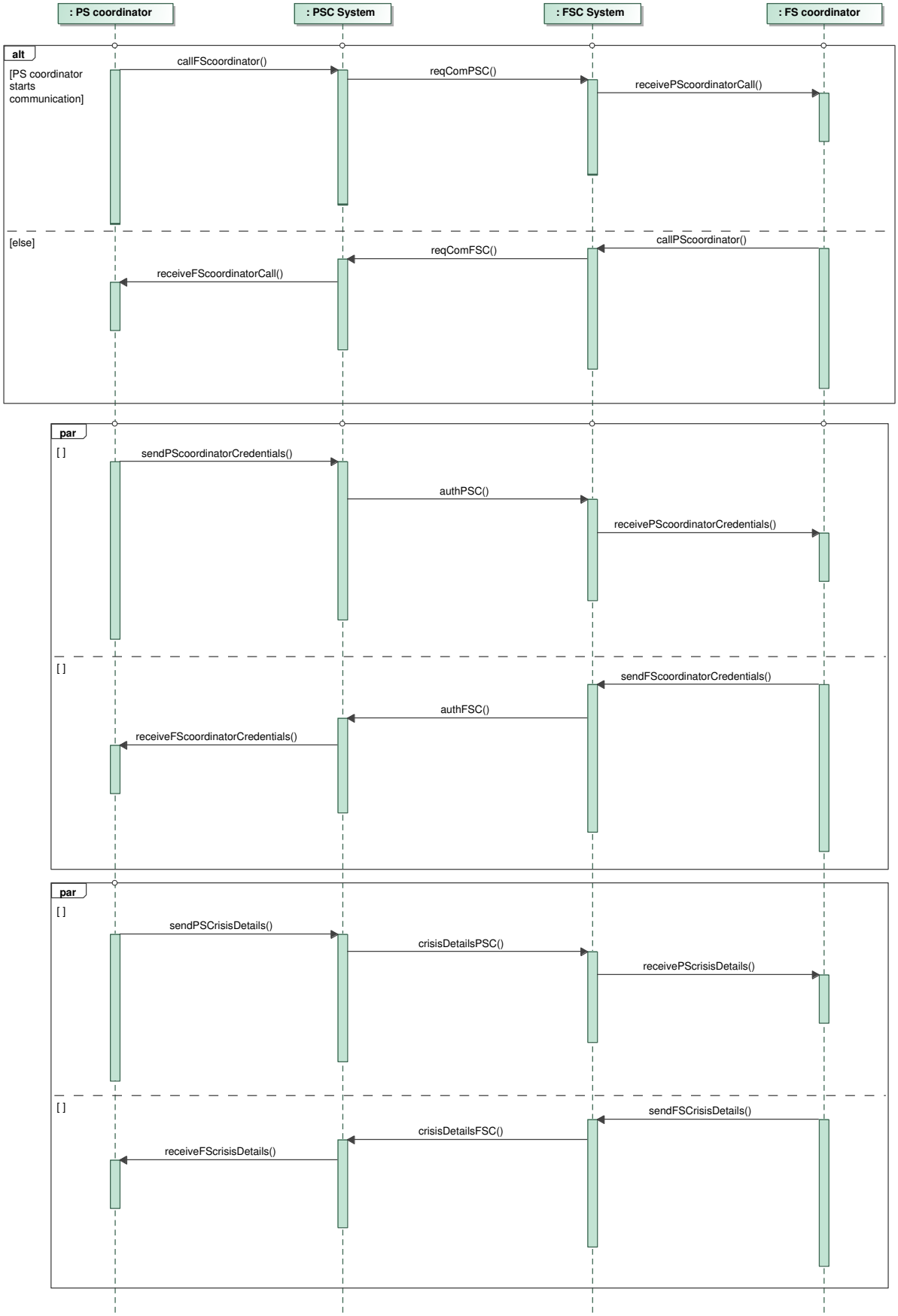


Figure 2: Sequence Diagram for steps 1 and 2.

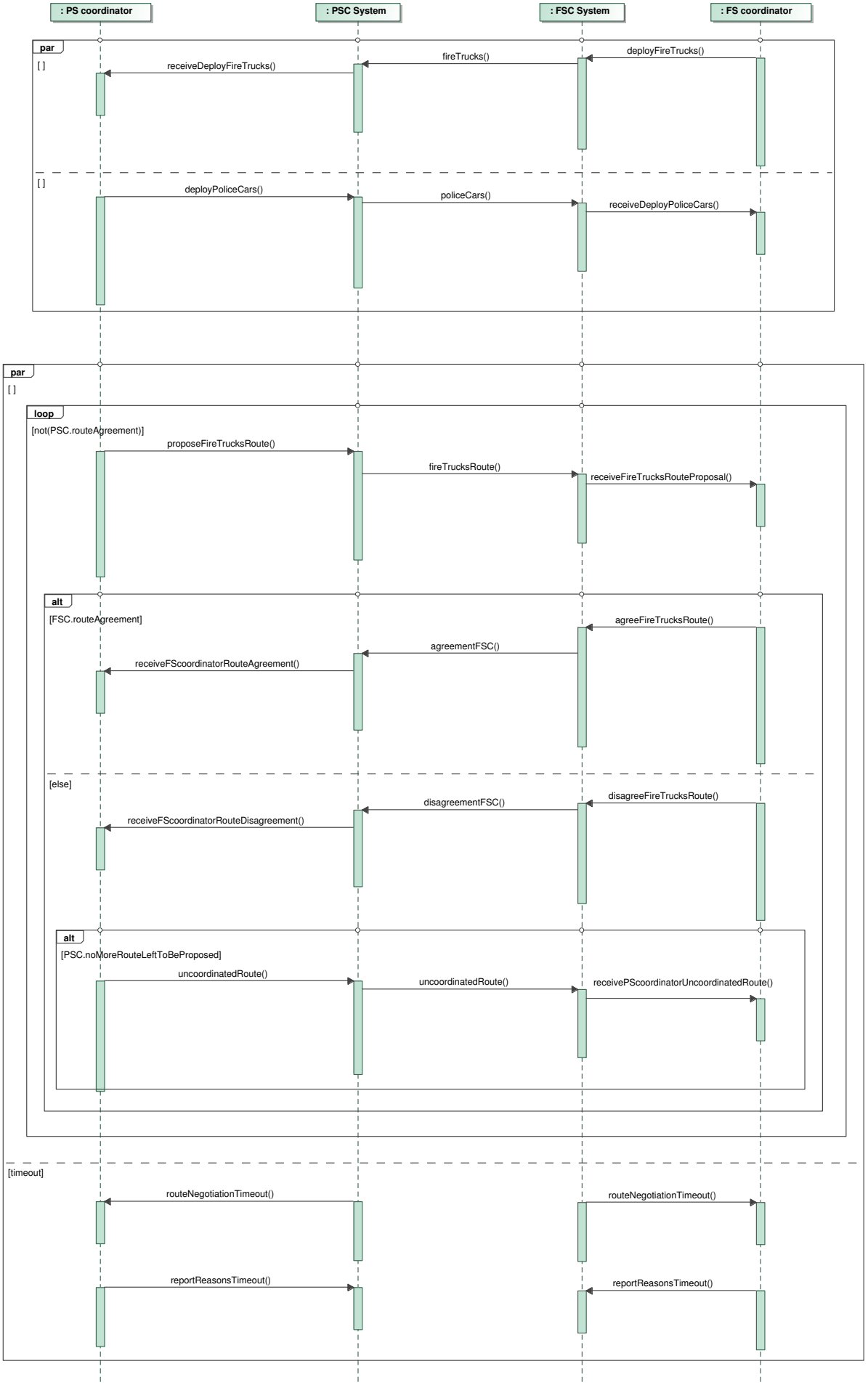


Figure 3: Sequence Diagram for step 3.

trucks) is known by counting the number of PoliceCar instances whose status is equal to *"inRouteToLocation"*.

2.2.4 Steps 5

For easing the visibility and reading of the sequence diagram that models the arrival of the vehicles to the crisis location, the diagram is presented in three parts through Figures 5-7. The aim of this sequence diagram is to model the notification that each coordinator does to once a dispatched vehicle arrives to the crisis location.

The sequence diagram (see Figure 5) starts with a loop block. The predicate of this block says that the contents are performed while the number of arrived police cars plus the number of fire trucks is lower than the number of requested police cars plus the number of fire trucks. The way of finding this numbers is quite the same as already explained in the step 4.

The content of the loop block is determined by a parallel block divided into four parts. The first part of the parallel block describes the different messages the FS coordinator may send to the PS coordinator, depending on the fire truck has arrived on time or not. In case the fire truck is delayed (alternative scenario **5.a.1**), the PS coordinator may decide either to replace such a fire truck by another one, or to simply assign a new ETA. In any case, each decision has to be communicated to the PS coordinator.

The different messages that the PS coordinator may send to the FS coordinator are described in the second part of the parallel block (see Figure 6). Since the rationale behind these messages is similar to what was explained for the FS coordinator, no further details about these messages is given.

The third part of the parallel block (see Figure 7) models the alternative scenarios that may take place when (at least) a police car has arrived at the crisis location. Once a police officer is at the crisis location, he might communicate to the PS coordinator that the crisis is either more or less severe than expected. In case the crisis is more severe that expected (alternative scenario **5.c1**), the PS coordinator must notify such a situation to the FS coordinator to start developing a new route plan for the extra required vehicles. In case the crisis is less severe than expected (alternative scenario **5.d1**), the PS coordinator notifies the FS coordinator that he has recalled some police cars since they were not required any more.

The same situations, but from the FS coordinator viewpoint are described in the fourth (and last) part of the parallel block.

2.2.5 Steps 6

The sequence diagram shown in Figure 8 models the way in which each coordinator notifies its respective peer about the completion of the vehicle objectives. Since the notification is done for each vehicle, then the parallel block (used to capture the non-determinism about the order in which the notifications are exchanged between the coordinators) is enclosed within a loop block. This loop block allows the parallel block to be performed while either a police car or fire truck remains at the crisis location. This

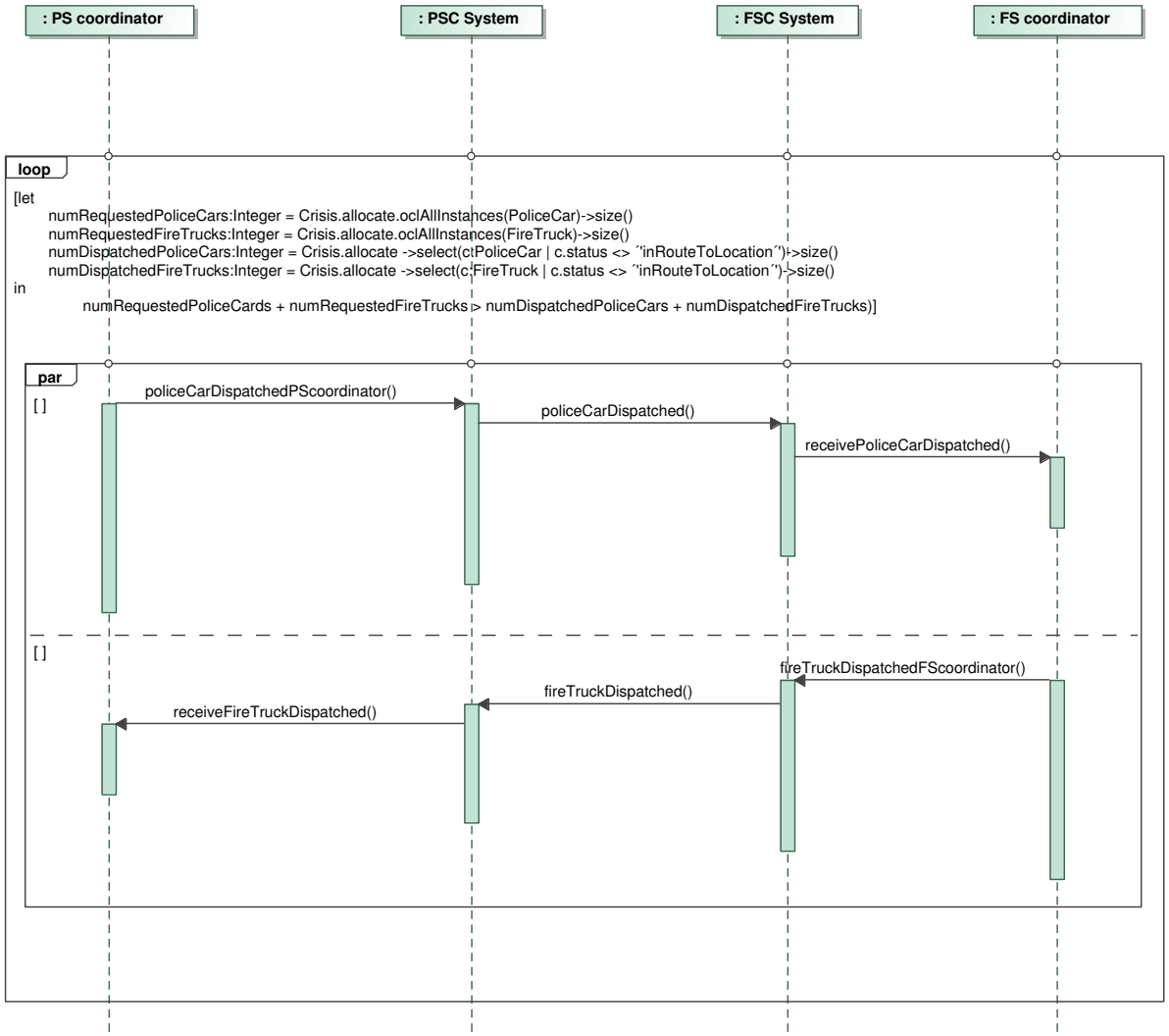


Figure 4: Sequence Diagram for step 4.

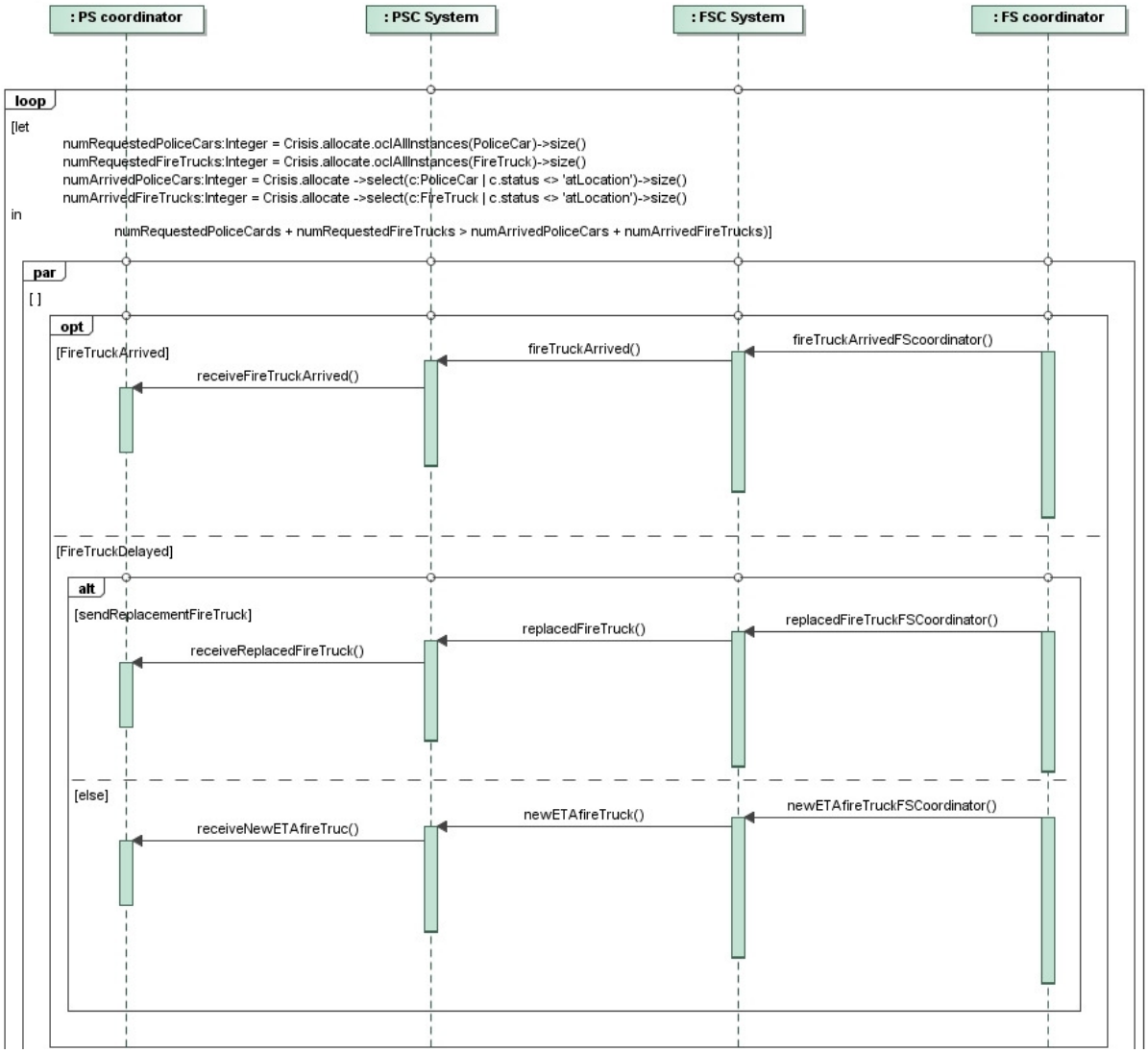


Figure 5: Sequence Diagram for step 5 - Top part.

Figure 6: Sequence Diagram for step 5 - Middle part.

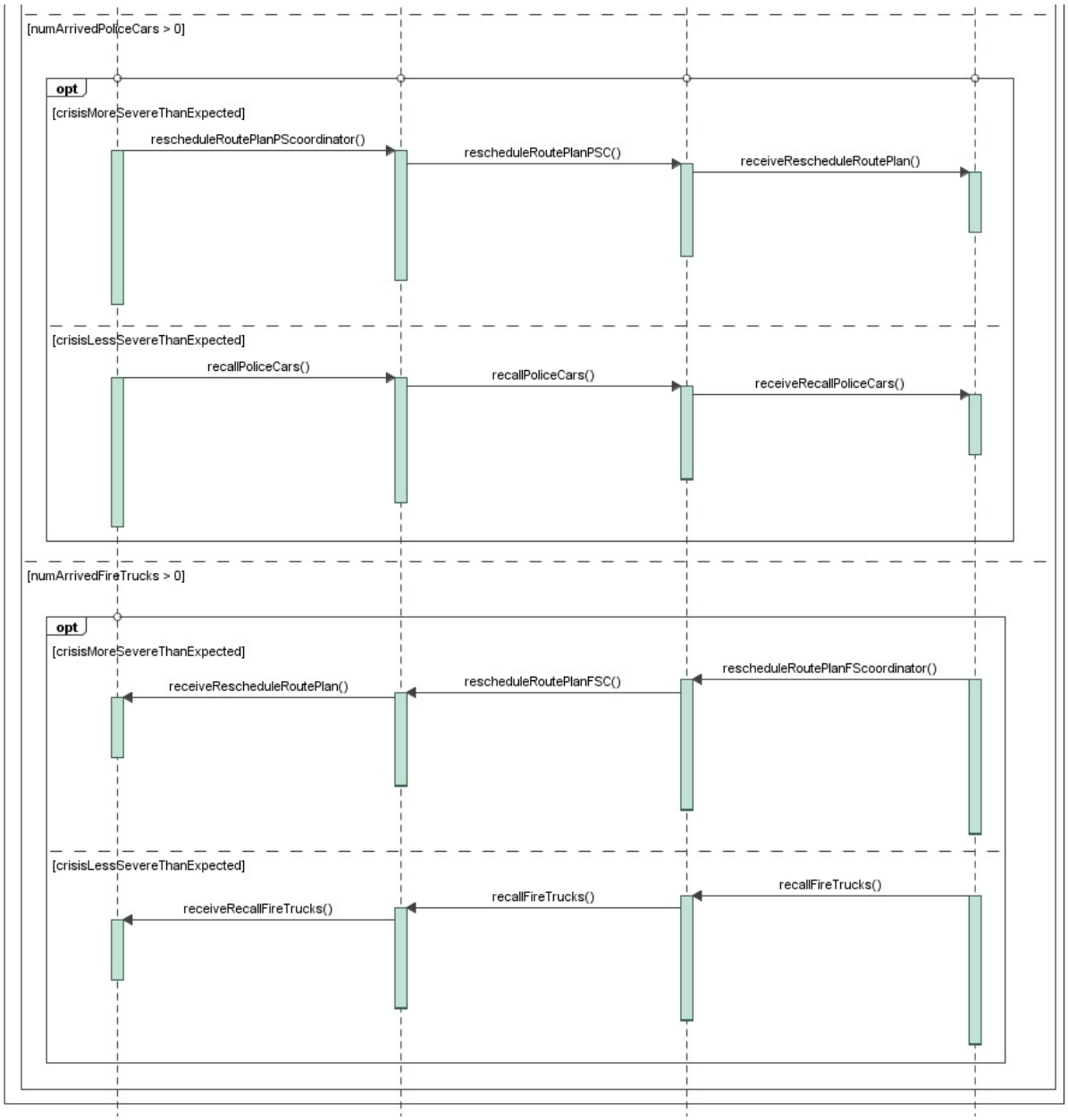


Figure 7: Sequence Diagram for step 5 - Last part.

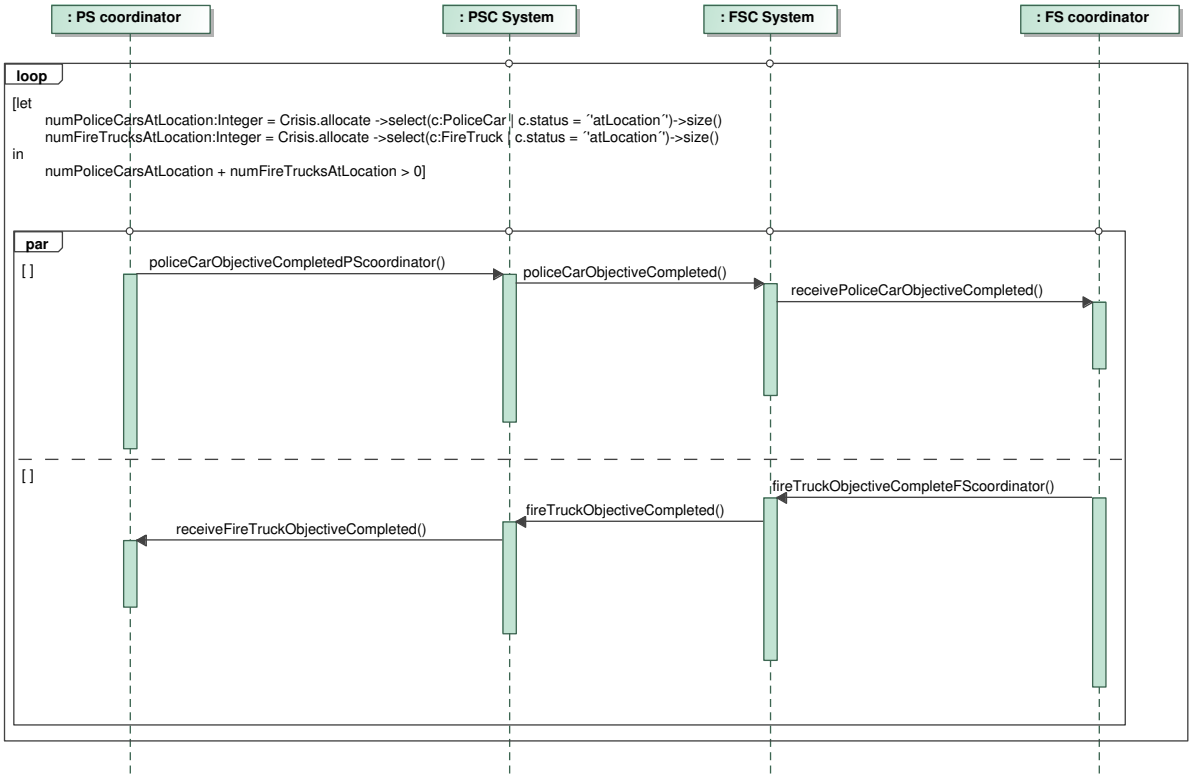


Figure 8: Sequence Diagram for step 6.

information is retrieved by counting the number of PoliceCar and FireTruck instances whose status is *"atLocation"*.

2.2.6 Steps 7

The sequence diagram shown in Figure 9 models the agreement reached by the coordinators to close the crisis. It is was assumed that a crisis can be only closed once every single vehicle has returned back to its central station. This fact is captured in the sequence diagram by the loop block, which has a predicate an OCL expression that remains true while the status of each PoliceCar and FireTruck is different from *"station"*.

Once each vehicle has returned back to its station, the coordinators are ready to notify to each other of their willingness to close the mission. This information is modeled by the parallel block, at the bottom of the sequence diagram.

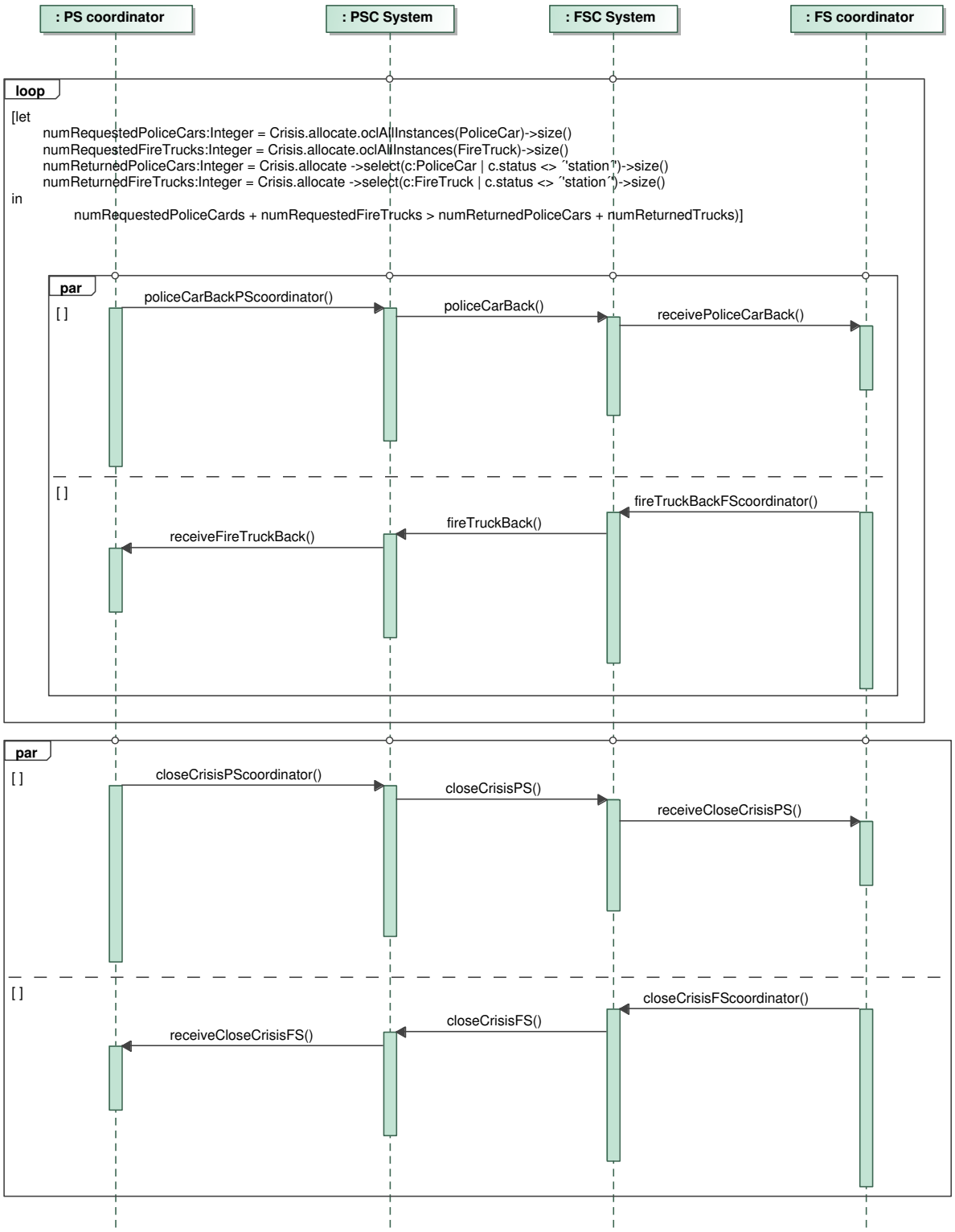


Figure 9: Sequence Diagram for step 7.

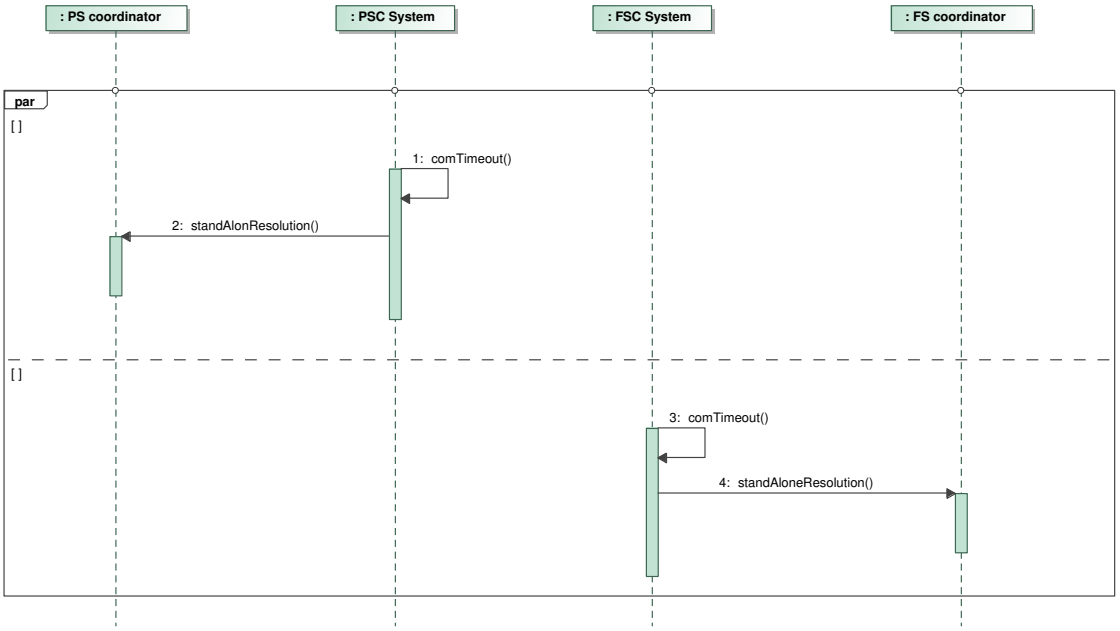


Figure 10: Unavailable communication.

2.2.7 Communication is not available/has been restored

Here, the scenario description considers the case in which the communication infrastructure that enables the coordinators to interact is not available. The sequence diagram shown in Figure 10 models the messages sent by the PSC System and FSC System to their respective coordinators to notify them that they will continue dealing with the crisis in a stand-alone mode.

As soon as the communication infrastructure has been restored, the PSC System and FSC System will synchronize. The synchronization of a system is considered as the communication of its current state to its peer system. Once the system has synchronized, it notifies its respective coordinator that the crisis will resume being handled in a collaborative manner. Figure 11 shows the sequence diagram that models the restoration of the communication.

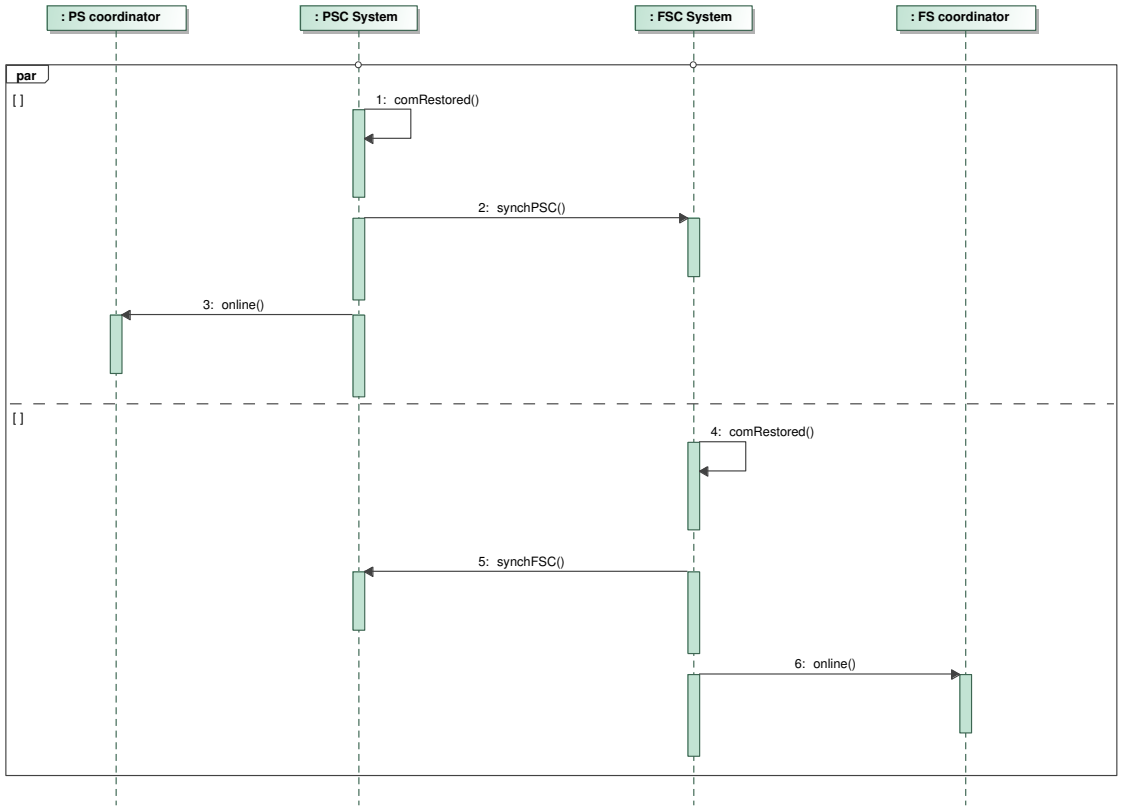


Figure 11: Restored communication.

2.3 State Machines

This section presents the state diagram used to describe the behavior of the key classes PS coordinator, FS coordinator, PSC System, and FSC System. The state diagrams are modeled according to the semantics defined by Harel *et al.* [7, 8]. Thus, each state diagram describes what is widely known as a *Harel's statechart*. In the remaining part of this paper, the terms *state diagram, state machines* and *statecharts* are considered synonymous. Statecharts comprise states that are connected by transitions. Each state can be simple or complex, meaning that it could contain entry/exit conditions, internal activities, etc. A transition has three possible elements, a trigger that captures an event to warrant movement from one state to another; a boolean guard that indicates that conditions that have to be true in order to take the transition; and an action list that comprises one or more actions that are generated as a result of the transition. Triggers and actions typically correspond to operation definitions in a class; and a guard is usually a boolean expression involving one or more attributes defined in the classes.

As the key elements within the *Domain Model* are assumed to exist concurrently, the statecharts that model their behavior must also co-exist. This modeling requirement is captured by putting in parallel the respective statecharts. These statecharts in parallel define a new statechart that is shown in Figure 12. It is considered the starting point for understanding the overall behavior of the system (defined by PSC System and FSC System) with its environment (defined by PS coordinator and FS coordinator). The next sections provide details about each of these statecharts.

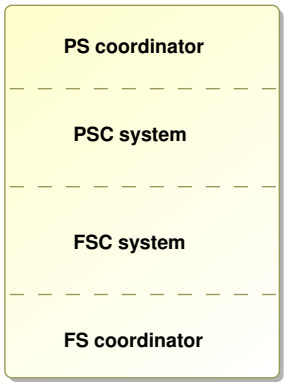


Figure 12: Parallel composition of the different existing state machines.

2.3.1 PS/FS coordinator

Due to the similarity between the PS coordinator and FS coordinator with respect to their behavior, we only provide details for one of them. The PS coordinator statechart is described in detail here, where the statechart for the FS coordinator would be comparable.

Figure 13 shows the statechart that models the behavior of the PS coordinator class, and Figure 14 shows the one that corresponds to FS coordinator. These behaviors were inferred by the messages that are sent or received by instances of type PS coordinator (FS coordinator, respectively), which were modeled through the sequence diagrams depicted in Figures 2-11. Every incoming message modeled in a sequence diagram, represents an *event* in a statechart, whereas an outgoing messages represents an *action*.

In summary, the statechart shown in Figure 13 indicates that a PS coordinator either calls the FS coordinator or receives a call from it. Any of these events make the PS coordinator ready to start the authorization step. The *authorizing* state defines two alternatives to achieve the authorization: either the PS coordinator first sends its credentials, and then the PS coordinator does the same, or vice versa. Once the authorization step is accomplished, the PS coordinator moves into the *ExchangingCrisisDetails* state. The parallel operator is used again to model the different alternatives in which the PS coordinator may go through this state. The same model principles are used when stating the number of vehicles.

We note that we have several cases where a trigger on a transition has a boolean guard. That is the case within the *NegotiatingRoutePlan* state. In this particular case, assuming the PS coordinator is in the state **S.3.2.1.2**[1], it may transition to the state **S.3.2.1.1** only at the moment the event *proposeFireTrucksRoute* happens, there exists at least one route to be proposed to the FS coordinator. Otherwise the PS coordinator remains in the same state.

Once the PS coordinator receives the agreement from the FS coordinator for the proposed route plan (modeled by the event *receiveFScoordinatorRouteAgreement*), it transitions to the *Dispatching* state. Once the dispatching activities have been completed, then the PS coordinator can transition to the *DealingWithCrisis* state. This state comprises three parallel compartments: the bottom partit models the notification of the arrival of a vehicle to the crisis location. The middle part models the notification of the completion of a vehicle's mission. And the top part models the notification of the returning of a vehicle.

These three different behaviors are placed in parallel since the notifications must be given/received for each vehicle. Thus, while some vehicles are still on their way to the crisis location, others may have already arrived, or even completed their mission. It is worth noticing that predicates are meant to ensure that a vehicle has to first arrive at the crisis location to be able to perform its mission, and then return to its station.

Once all dispatched vehicles have returned to their respective stations, the PS coordinator may transition to the state where he is allowed to close the crisis. Closing a crisis is achieved either by receiving a closing request from the FS coordinator, and the sending of the closing event, or first sending the request for closing and then later receiving the same request from the FS coordinator. An alternative to this behavior may take place when the PS coordinator is dealing with a crisis in a stand-alone mode. In that case, he

[1]The name of the states have been chosen such that the reader may easily find the relationship between the state machine's behavior and the scenario description presented at the beginning of Section 2.2

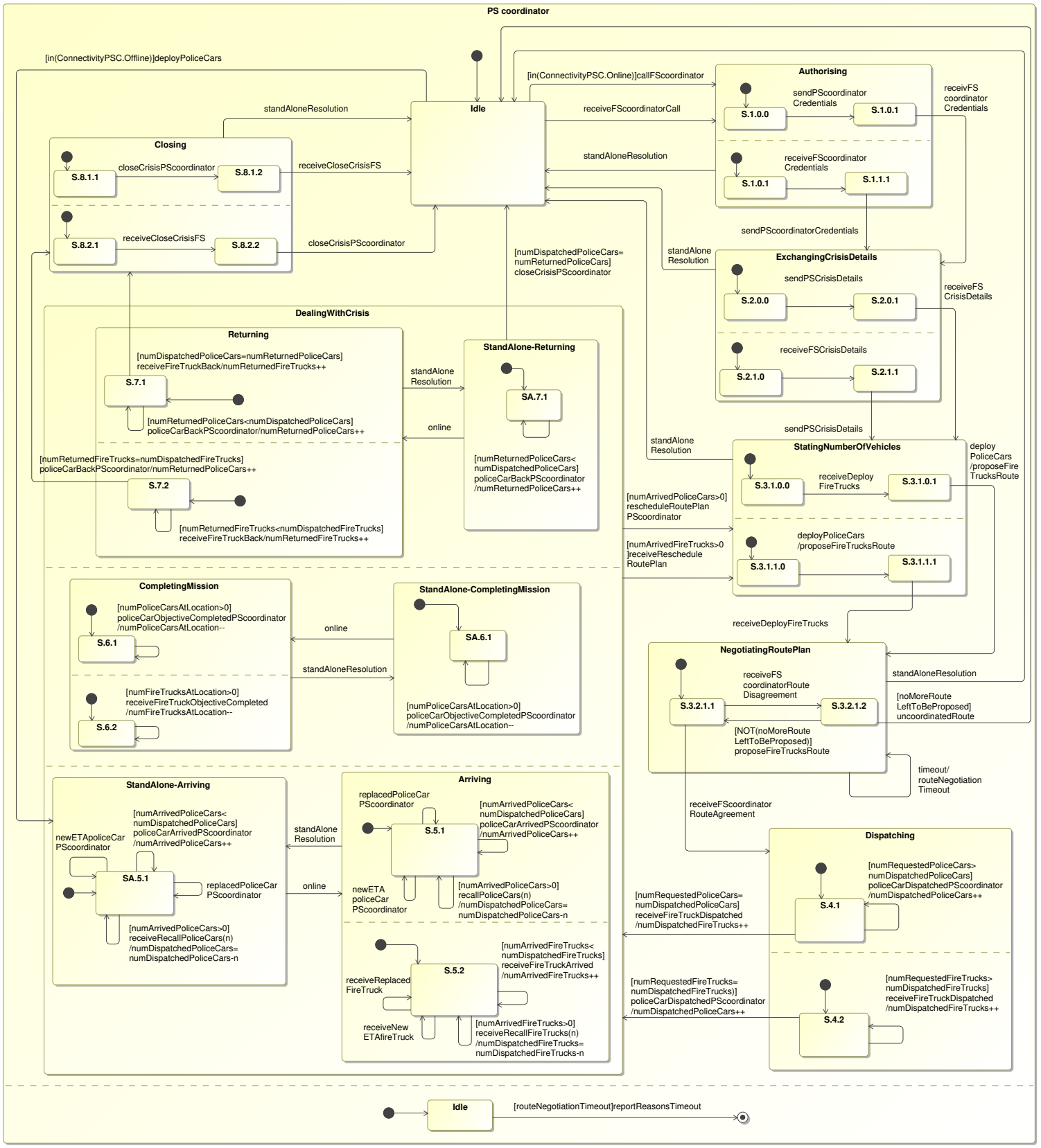


Figure 13: PS coordinator State Machine.

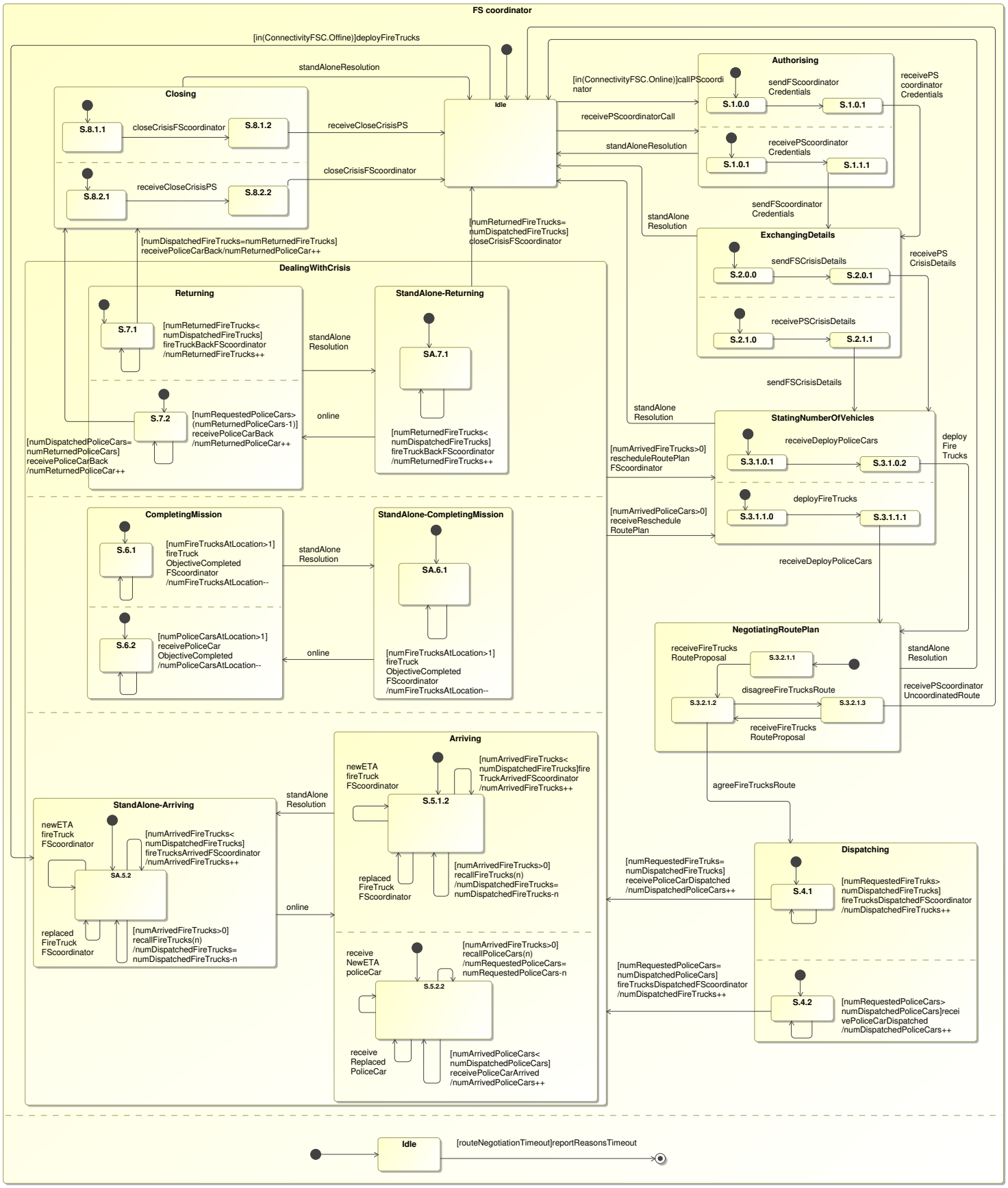


Figure 14: FS coordinator system State Machine.

does not need to wait for FS coordinator-related events to transition. Notice that this is not the only alternative behavior that is modeled in the statechart. All the alternatives scenarios modeled in the sequence diagrams are also included in the statecharts. That is the reason why there are events like *recallPoliceCars* or *replacedPoliceCarPScoodinator*, in line with the information modeled in the sequence diagrams.

2.3.2 PS/FS system

Given the details for the PS coordinator and FS coordinator statecharts, and the relationships between these elements and the PSC System and FSC System elements, respectively, we can then model the statechart behavior as shown in Figures 15 and 16, respectively.

It is important to note that the statecharts for these four elements illustrate how the four elements, in fact, interact, where transitions in a given statechart generate actions that serve as triggers for transitions in one of the other statecharts. As a result, we have a set of interacting statecharts that reflect how the four elements interact to deliver the overall desired behavior.

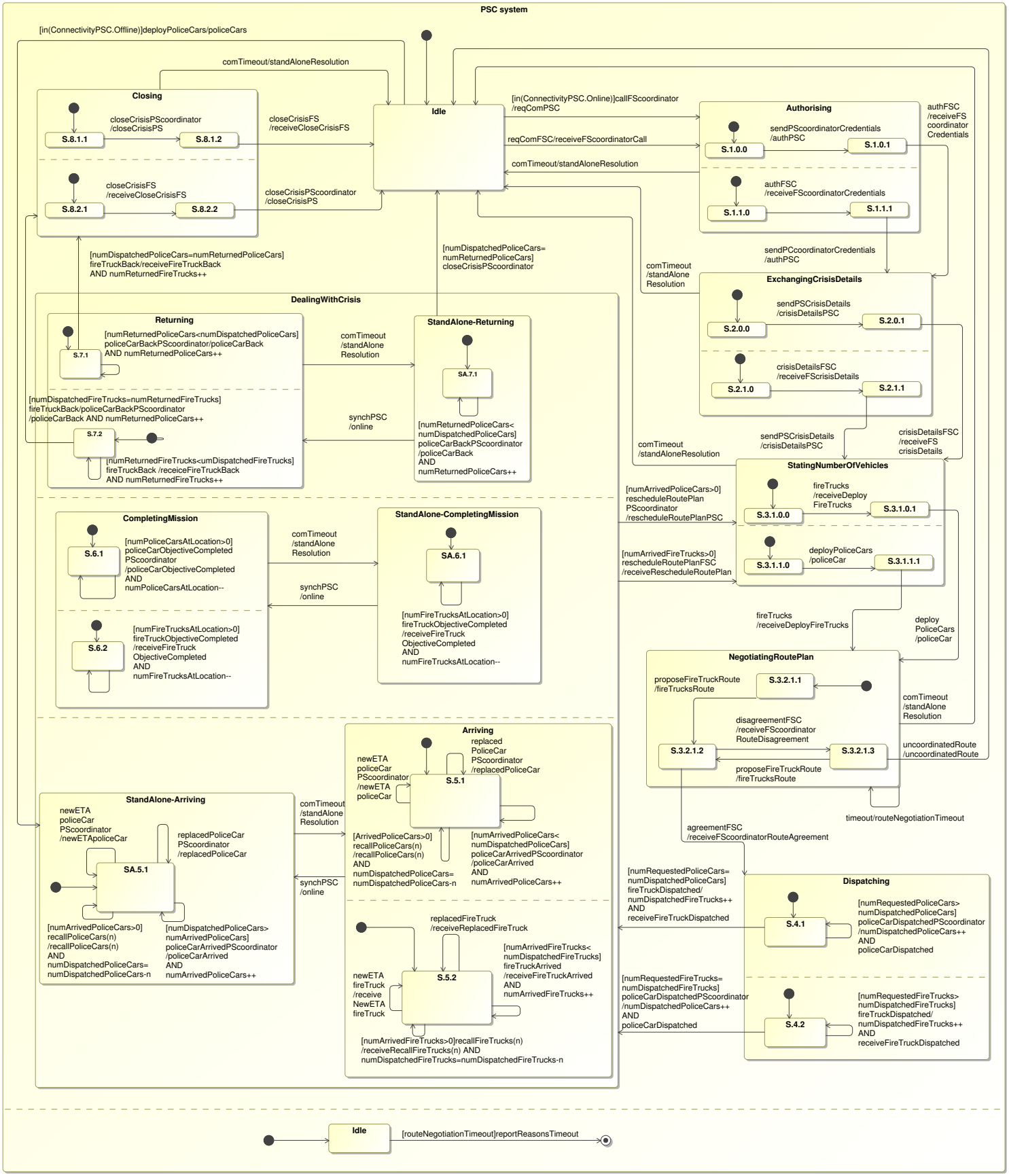


Figure 15: PSC system State Machine.

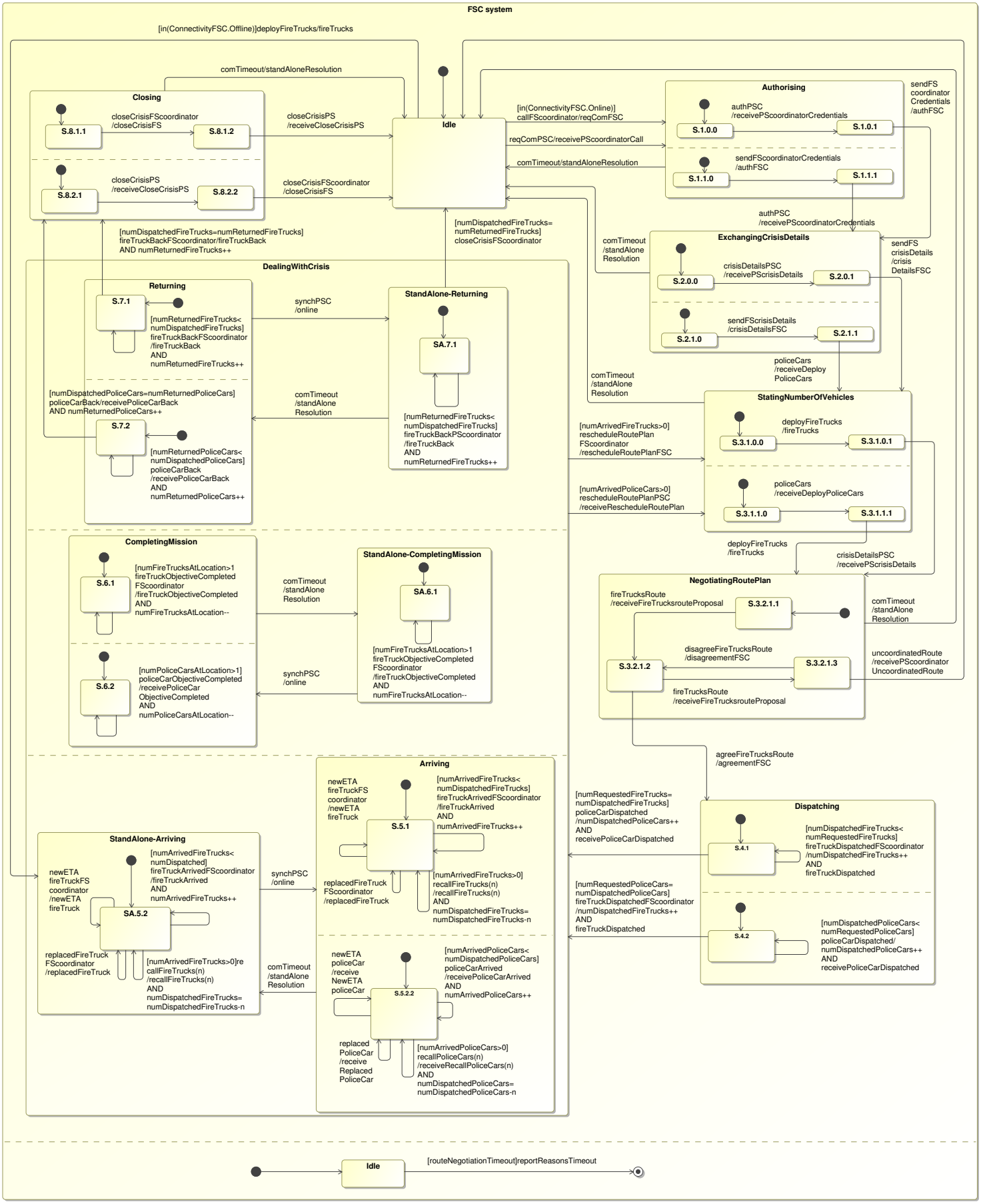
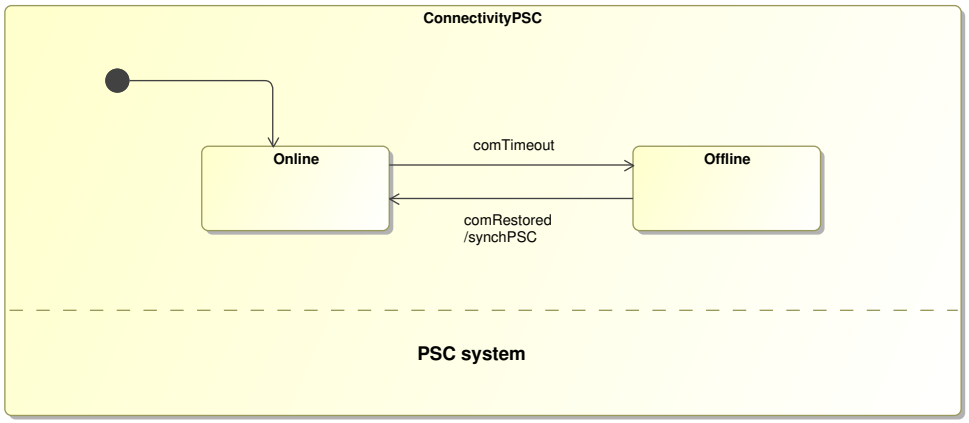
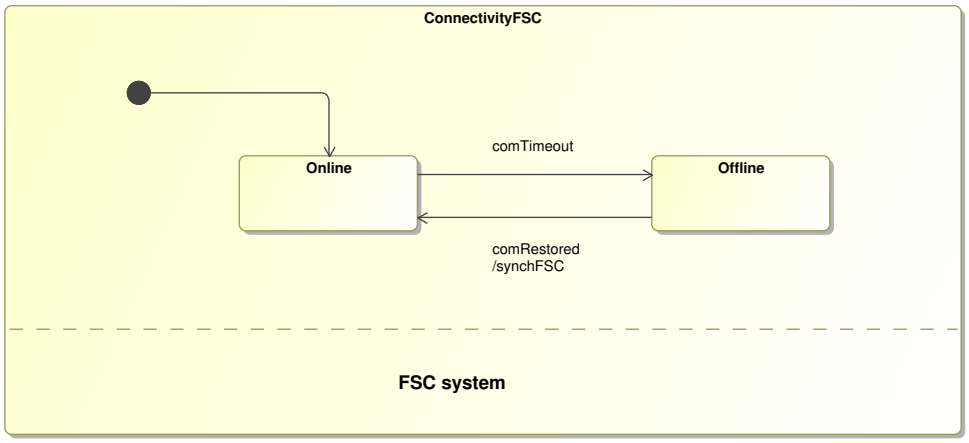


Figure 16: FSC system State Machine.



(a) State Machine about the Connectivity of the PSC system



(b) State Machine about the Connectivity of the FSC system.

2.3.3 PS/FS system connectivity

Regarding the point about the availability of the communication channel, specific statecharts were modeled to describe the availability/unavailability of the communication channel on each system. Figures 17(a) and 17(b) show the statecharts that allow each system to realise whether it is in a collaborative crisis management mode, or in a standalone mode. Notice that each time the communication is restored, a synchronization action takes place, as already mentioned and previously described by the sequence diagrams.

2.4 High-level design

The high-level design is aimed at describing the software architecture and the behavior of the components that form part of such architecture. Figure 17 depicts the peer to peer (P2P) architectural style chosen for the bCMS system. We use the UML class diagram notation to present the architecture as we want to adhere to widely used notations. Thus, components are modeled as classes, whereas their interaction is described by directed associations. PSC System and FSC System are the components (or peers) that behave either as client or server, depending on the particular situation. The interaction between the components is achieved by operation calls.

The refinement to the high-level design phase from the late requirements models only affects the structural part of the bCMS system. This means that the behavior of the PSC System and FSC System components remain the same as described in the previous phase (i.e. late requirements) by the state diagrams in Figures 15 and 16, respectively.

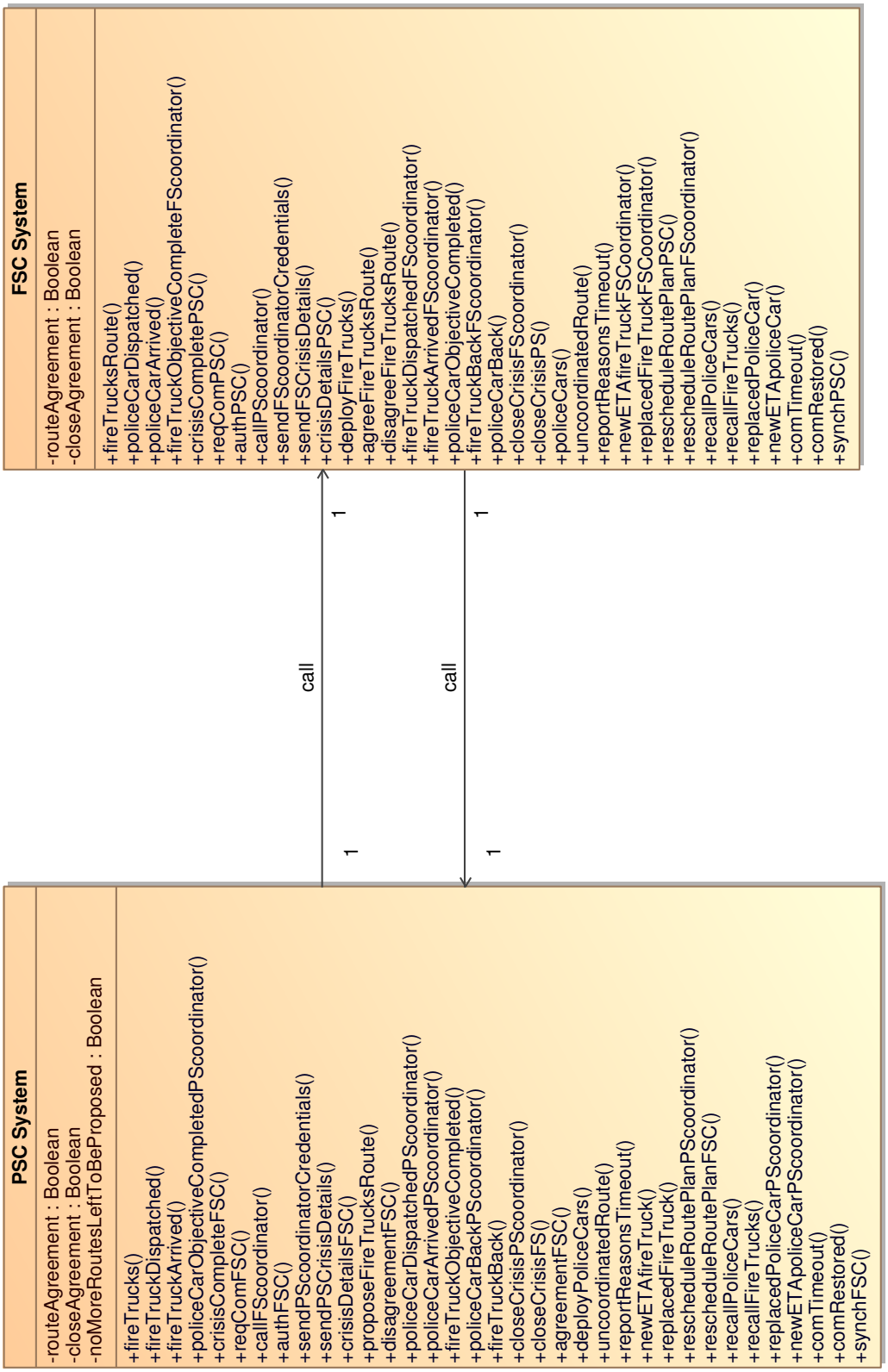


Figure 17: bCMS software architecture.

3 Software Product Line (SPL) modelling approach

3.1 SPL - general concepts and engineering process

Software Product Lines (SPL), or software families, are rapidly emerging as a viable and important software development paradigm designed to handle such issues [12]. Use of SPL approaches allowed renowned companies like Hewlett-Packard, Nokia or Motorola to achieve considerable quantitative and qualitative gains in terms of productivity, time to market and customer satisfaction [1]. Their increasing success relies on the capacity to offer software suppliers/vendors ways to exploit the existing commonalities in their software products. This new concept drew the attention of the software community when software started to be massively integrated into hardware product families, with cellular phones [11] probably being the most well known example. More generally, automotive systems, aerospace or telecommunications are also areas relevant for SPLs.

Several definitions of the software product line concept can be found in the research literature. Clements et al. define it as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way" [13]. Bosch provides a different definition [4]: "A SPL consists of a product line architecture and a set of reusable components designed for incorporation into the product line architecture. In addition, the PL consists of the software products developed using the mentioned reusable assets". In spite of the similarities, these definitions provide different perspectives of the concept: market-driven, as seen by Clements et al., and technology-oriented for Bosch.

SPL engineering focuses on capturing the commonality and variability between several software products [6]. Instead of describing a single software system, a SPL model describes a set of products in the same domain. This is accomplished by distinguishing between elements common to all SPL members, and those that may vary from one product to another. Reuse of core assets, which form the basis of the product line, is highly encouraged. These core assets extend beyond simple code reuse and may include the architecture, software components, domain models, requirements statements, documentation, test plans or test cases [18].

The SPL engineering process consists of two major steps:

- **Domain Engineering**: or *development for reuse*, focuses on core assets development. It consists of collecting, organizing, and storing past experiences in building systems in the form of reusable assets and providing an adequate means to reuse them for building new systems [14]. It starts with a domain analysis phase to identify commonality and variability among SPL members. During domain design, the PL architecture is defined in terms of software components and is implemented during the last phase.

- **Application Engineering**: or *development with reuse*, addresses the development of the final products using core assets and following customer requirements.

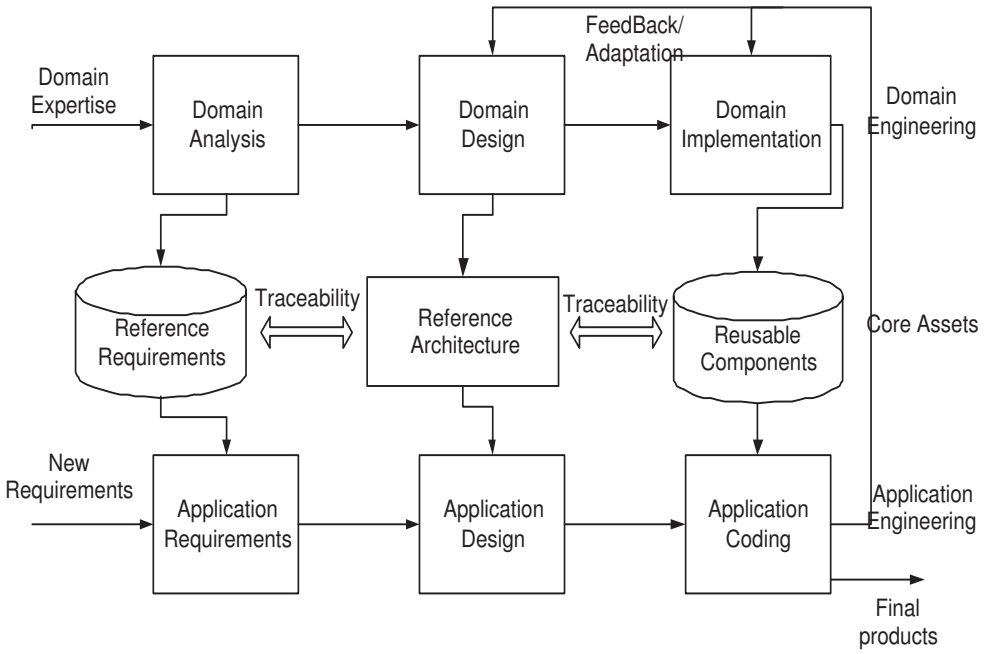


Figure 18: General SPL engineering process.

This phase is also known as *product derivation*, and consists of building the actual systems from the core assets base. According to the derivation technique used, currently available product derivation approaches can roughly be sorted in: configuration and transformation.

Figure 18 graphically represents the general SPL engineering process, as it can be found in the research literature [13]. As illustrated, the two phases are intertwined: application engineering consumes assets produced during domain engineering, while feedback from it facilitates the construction or improvement of assets.

3.2 SPL methodology used for bCrash SPL

The process of creating the bCrash SPL follows the general SPL engineering methodology presented in the previous section. For each step of the process, we make use of the most popular and well known available approaches. In this way, we assure that the design of the bCrash SPL follows a traditional and well known SPL design approach. The applied methodology follows the two well known phases of SPL engineering:

Domain engineering: during this step, the following activities are performed:

- Capture the commonality and variability between all the members of the product line. This is done by using Feature Diagrams, the most popular and wide spread method for representing variability in SPL. We will therefore create a feature diagram of the bCrash SPL system, based on the requirements document provided. This is an essential step, as variability modelling is a key aspect of product line engineering.

- We also focus on core asset development. For this purpose, we will first reuse the OO models created for the reference variant of bCrash SPL. This reference variant consists of features (functionalities) which are mandatory for all the products of the product line. Therefore, the OO models of this reference variant will be the basis for creating (deriving) other products. The core asset development phase also consists of creating models for the variation points and their associated variants. These represent extra functionalities of the system, and allow the creation of a diverse set of products. The variation point models, together with the models for the reference variant, make up the entire set of models needed for representing the bCrash SPL.

Application engineering: this phase of the process leads to the creation of an actual product. During this step, the following activities are performed:

- Feature diagram configuration: based on the specific requirements of the user, the feature diagram is configured. This implies that all the variation points need to be resolved. Decisions need to be made regarding the optional features: which of them are kept for a specific product. All the mandatory features are kept in the new product that is derived. The result of this step is a feature diagram that does not contain any variabiliy.

- Based on the selections made during the feature diagram configuration phase, we obtain a set of OO models from which the model of the actual product will be obtained. The OO models of the reference variant correspond to the mandatory features that were selected. The models for each variation point will be configured (transformed), based on the choices that were made for each variation point during the feature diagram configuration step. The resulting models will be composed with the models of the reference variant, thus obtaining the actual models of the specific product that we are deriving.

For the design of the object-oriented models for the reference variant, the "approaches and the modelling languages used have been chosen in order to comply with widely accepted standards". This imposes some constraints SPL methodology presented here. As UML class diagrams, sequence diagrams and state machines are the types of models used for modelling the reference variant, the same type of models must be used to represent the variation points.

3.3 Variability in SPL and Feature Diagrams

Variability is seen as the key feature that distinguishes SPL engineering from other software development approaches. Variability has been defined in different ways in a product line context. For Weiss et al. [17] it is "an assumption about how members of a family may differ from each other". According to Bachmann et al. [2, 3] "variability means the ability of a core asset to adapt to usages in different product contexts that are within the product line scope". For Pohl et al. [15] it is the "variability that is modelled

to enable the development of customized applications by reusing predefined, adjustable artefacts".

Central to the SPL paradigm is the modelling and management of variability, the commonalities and differences in the applications in terms of requirements, architecture, components and test artefacts. At all these levels, a popular way to model variability is through *Feature Diagrams (FD)*. They are the first proposal of the product line community for dealing with variability. According to Kang et al. [9, 10] a feature is "a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems".Feature diagrams emerged as a popular SPL variability modelling technique ever since Kang et al.s proposal in 1990 to express feature relations using a feature model. It consists of a feature diagram and other associated information: constraints and dependency rules.

Feature diagrams provide a *graphical tree-like* notation depicting the hierarchical organization of high level product functionalities represented as features. The root of the tree refers to the complete system and is progressively decomposed into more refined features (tree nodes). Relations between nodes (features) are materialized by decomposition edges and textual constraints.

Variability can be expressed in several ways. Presence or absence of a feature from a product is modelled using *mandatory* or *optional* features. Features can be organized into feature groups. Boolean operators exclusive alternative (XOR), inclusive alternative (OR) or inclusive (AND) are used to select one, several or all the features from a feature group. Moreover, dependencies between features can be modelled using *textual constraints*: requires (presence of a feature imposes the presence of another), *mutex* (presence of a feature automatically excludes another). Several factors contribute to the popularity of FD: they provide a concise way to describe allowed variabilities between products of the same family, represent feature dependencies, guide feature selection, allow the construction of a specific product.

3.4 Creating the Feature Diagram of bCrash SPL

The feature diagram of the bCMS SPL is created based on the requirements document. It is created in a two steps process. We start by analysing the possible variation points listed in Section 7 of the requirements document. Then, in a second step, we analysis the "reference variant", in order to identify the features corresponding to the basic actions defined in the given scenario. For this purpose we analyse both the functional and non-functional requirements of the "reference variant", corresponding to Sections 4 and 5 of the requirements document, respectively.

3.4.1 Analysis of the variation points

In the requirements document, variations are divided into *functional* (section 7.1 to 7.4) and *non-functional* (section 7.5 to 7.7) depending the type of requirement they target. Thus, the FD of the bCMS has two mandatory features called *Functional* and

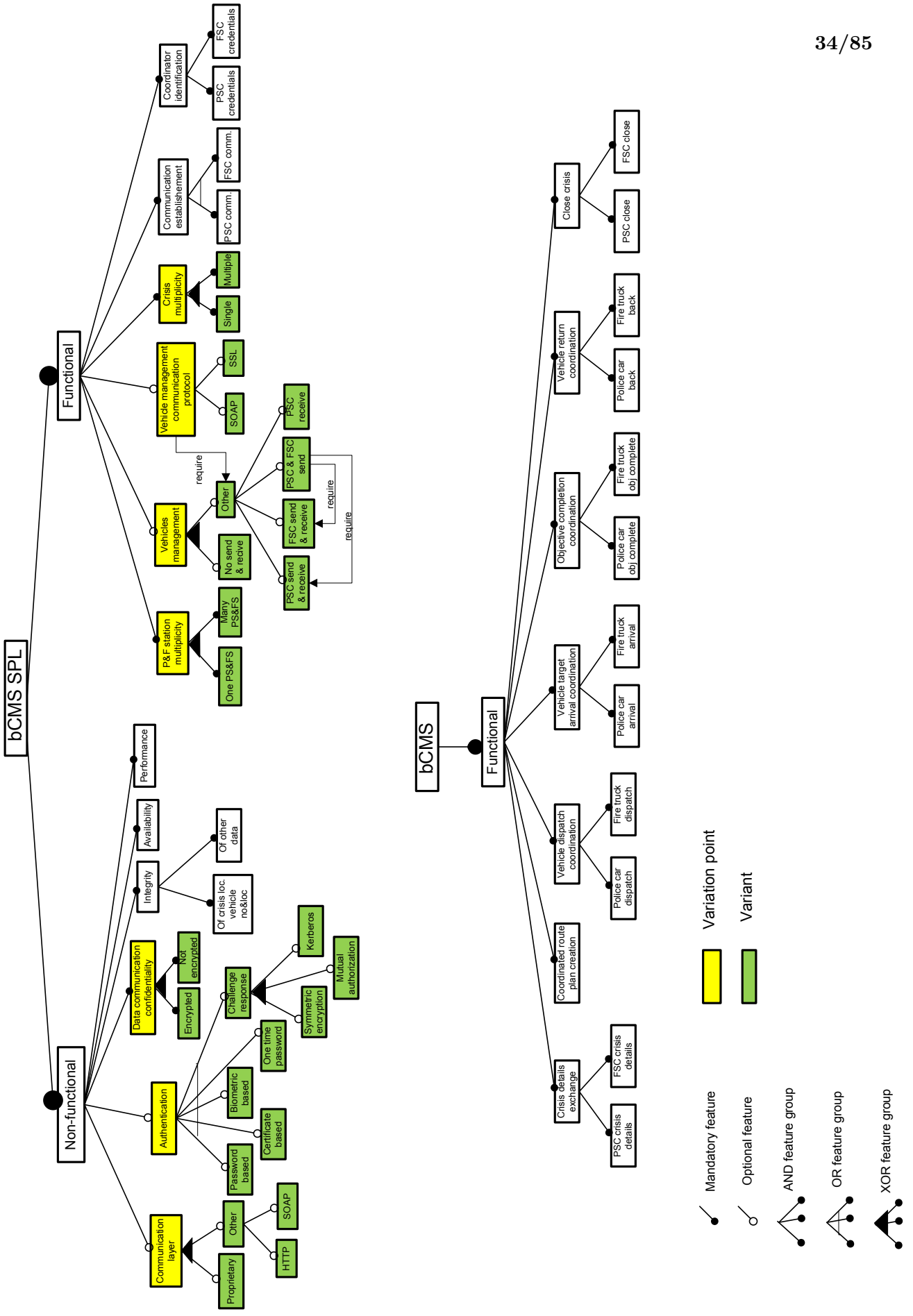


Figure 19: Feature diagram of bCMS SPL system.

Non-functional, which descend of the root feature named *bCMS*. An *AND feature group* decomposition is used to relate the (4^2) features with the parent feature *Functional*.

The priorities "*must have*" and "*may have*" defined in Table 1 of the requirements document are translated as *mandatory* and *optional* features into the FD, respectively. The concrete variants to be implemented by each variation point correspond to those items listed within the "*Variations*" part of each variation point. In line with that, the information placed within the "*Constraints*" part is used to determine the type of relationship between the different concrete variants of a variation point. Therefore, this information leads towards the creation of the following features:

- **Police and Fire Stations Multiplicity feature**
  This is a mandatory feature whose parent is *Functional*. It is decomposed into 2 mutual exclusive (XOR) features named *One PS & FS* and *Many PS & FS*.

- **Vehicles Management feature**
  This is an optional feature whose parent is *Functional*. It is decomposed into 2 mutual exclusive variants. In the constraints part of Section 7.2, it is indicated that the variant *No send & receive* (referred to as $variant_1$) excludes the variants *PSC send & receive*, *FSC send & receive*, *PSC & FSC send*, and *PSC receive* (referred to as $variants_{2..5}$, respectively). Thus, for easing the modelling, $variants_{2-5}$ have been regrouped in a unique variant called *Other*, which is mutually exclusive with $variant_1$. In this way, the first constraint is fulfilled. $Variants_{2..5}$ defined in Section 7.2 become children of feature *Other*, are all optional, and related by an AND decomposition. The second and third constraints are represented as "require" feature dependencies between the different variants.

- **Vehicles Management Communication Protocol feature**
  This is an optional feature whose parent is *Functional*. It is decomposed into 2 optional sub-features named *SOAP* and *SSL*. These features are related to each other by an AND decomposition. The phrase *"In case the system offers the functionality of communication between PSC/FSC and their respective vehicles"* is translated into a "require" feature dependency between these features and the feature *Other* from the feature *Vehicles Management*.

- **Crisis Multiplicity feature**
  This is a mandatory feature whose parent is *Functional*. It is decomposed into 2 also mandatory features named *Single* and *Multiple*. These 2 features are related by a mutual exclusion (XOR) feature decomposition.

- **Communication Layer feature**
  This is an optional feature whose parent is *Non-functional*. It is decomposed in 2 mutually exclusive features named *Proprietary* and *Other*. The *Other* feature is introduced in order to fulfil the first constraint given in Section 7.7. Concrete

[2]These variations are: *Police and Fire Stations Multiplicity, Vehicles Management, Vehicles Management communication Protocol,* and *Crisis Multiplicity*

variants *HTTP* and *SOAP* are children of *Other* feature, and they are connected to each other by an AND feature decomposition.

- **Authentication of System's Users feature**
  This is an optional feature whose parent is *Non-functional*. It is decomposed into 5 optional children which correspond (and are named after) to the variations defined in section 7.6. There is a further decomposition of feature *Challenge response* into 3 mutual exclusive sub-features called *Symmetric encryption, Mutual authorization,* and *Kerberos*.

- **Data Communication Confidentiality feature**
  This is a mandatory feature whose parent is *Non-functional*. It is decomposed into 2 mutually exclusive sub-features named *Encrypted* and *Not encrypted*.

3.4.2 Analysis of the reference variant

As the requirements document specifies, Sections 4 and 5 describe the characteristics of a "reference variant of the SPL". Therefore, the analysis of these sections will produce a set of mandatory features that should be captured in the feature diagram.

Based on the main scenario defined in Section 4, mandatory features corresponding to the basic actions defined in this scenario can be extracted. These features become children of the *Functional* feature previously defined. It is worth adding that the OO models created for the reference variant were very useful when extracting features.

As result of this analysis, the following mandatory features are extracted:

- communication establishment
- coordinator identification
- crisis details exchange
- coordinate route plan creation
- vehicle dispatch coordination
- vehicle target arrival coordination
- objective completion coordination
- vehicle return coordination
- close crisis

The same kind of analysis is performed now on Section 5 from the requirements document, and results in a set of features that become mandatory children of *Non-functional* feature. These mandatory features are related to each other by an AND feature decomposition, are the following:

- integrity
- availability
- performance

The end result of this entire process is the feature diagram of the bCMS SPL system, presented in Figure 19. To make its understanding easier, the features coloured in yellow represent variation points, while the variants of a variation point are coloured in green.

3.5 UML models

This section describes the process of creating the UML models for the variation points of the bCrash SPL and their associated variants. Two kinds of UML models are used: class diagrams and sequence diagrams. These models are created starting from the models provided initially for the reference variant. Each models adapts the model described for the reference variant in order to express the particularities of the specific variation point. Different types of mechanisms need to be used in order to capture in these models the *"variability"* aspect.

3.5.1 Vehicle Management variation point

Modelling the class diagram:

- A new class called *Dispatch service* is created. This class is related to the *PSC system* and *FSC system* classes by a relation named *call*. This relation describes that each class will call the dispatch service in order to contact its respective vehicles, as well as the citizen vehicles.
- The class *Dispatch service* is also connected with the *Police Car, Fire Truck* and *Citizen Vehicle* classes to express the fact that the dispatch service communicates with all these vehicles.
- Since the communication management of vehicles corresponds to an optional feature in the FD, and in UML there is no direct way of expressing the fact that a class is "optional", we make use of the stereotyping functions. Thus, the class *Dispatch service* is tagged with the stereotype $\ll$ *optional* $\gg$. This class will be connected with the variant *Other* from the *Vehicle Management* feature. This means that it will only be selected when the feature *Other* is selected in the FD. This is the role of the $\ll$ *optional* $\gg$ stereotype.
- Every method of this class is also tagged as $\ll$ *optional* $\gg$.
- The model of the reference variant corresponds to the selection of the feature *No send and receive*. No change to the model is required in case this feature is selected for the SPL configuration part.

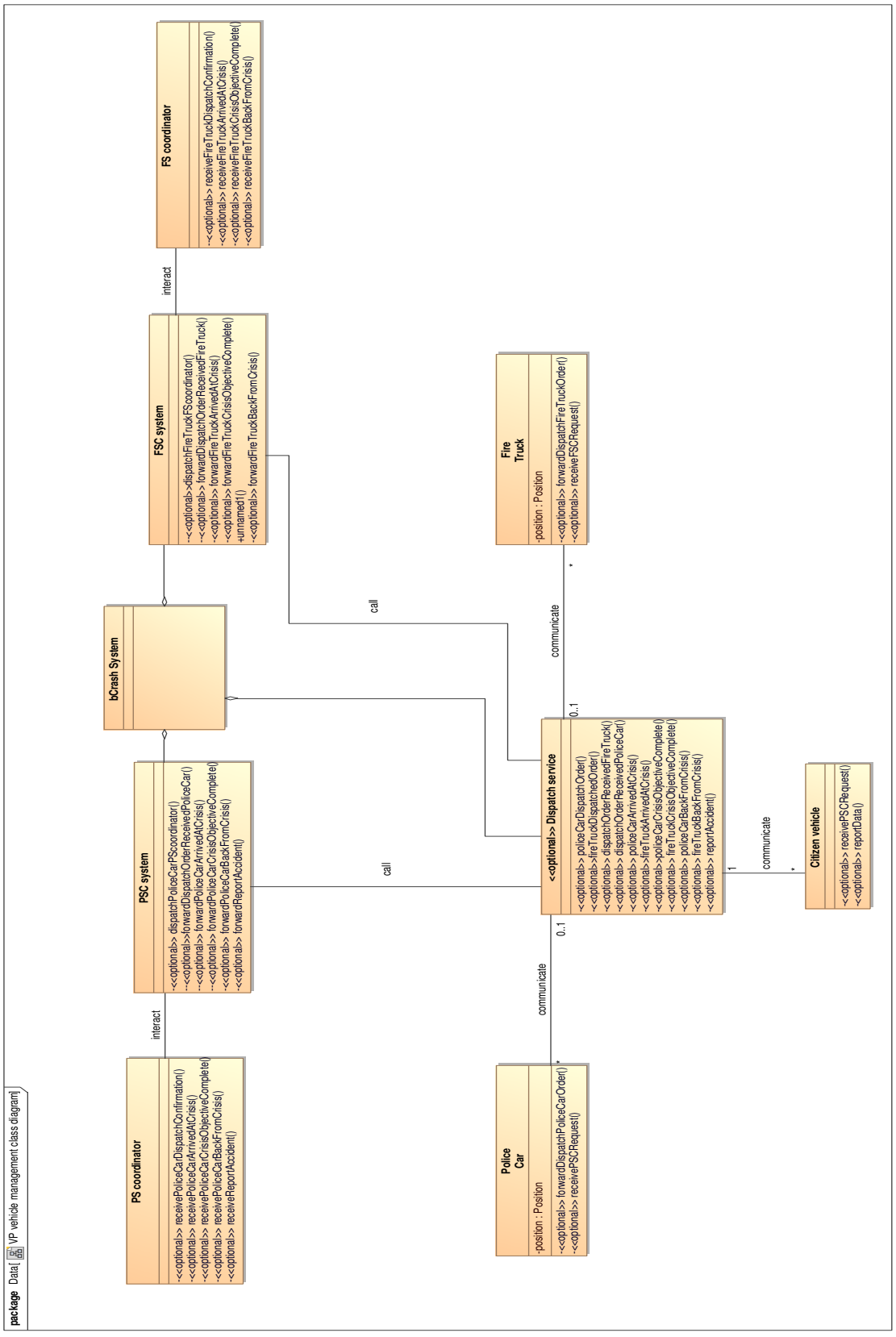


Figure 20: Vehicle Management variation point.

- When the choice is made in the FD and the *Other* feature is selected, then the connections (relations) between the *PS coordinator* class and the *Police Car* class (relation *manage*) needs to be destroyed. The same is applied to the *manage* relation between the *FS coordinator* and the *Fire truck* class. This represents the fact that the PS coordinator and FS coordinator do not interact with the vehicles any more, which will be done from now on through the dispatch service.

- In order to model the 4 variants of the *Other* feature, representing different types of communication between the PSC, FSC and their respective vehicles and the citizen vehicles, new methods are introduced in the *PS coordinator, PSC system, FS coordinator, FSC system, Dispatch service, Police car, Fire Truck,* and *Citizen vehicle* classes. All these methods are tagged with the stereotype ≪ *optional* ≫, as they belong to optional features in the feature diagram and represent optional behaviour. We explain in the following how the variant "Only the PSC can send to and receive messages from police vehicles and citizen vehicles" (which corresponds to the feature *PSC send & receive*) is modelled in details. The remaining variants descending from the feature *Other* are modelled in a very similar fashion.

  - Five new ≪ *optional* ≫ methods named *receivePoliceCarDispatchConfirmation()*, *receivePoliceCarArrivedAtCrisis()*, *receivePoliceCarCrisisObjectiveComplete()*, *receivePoliceCarBackFromCrisis()* and *receiveReportAccident()* are added to the *PS coordinator* class.
  - Five ≪ *optional* ≫ methods are added to the *PSC system* class: *forwardDispatchOrderReceivedPoliceCar()*, *forwardPoliceCarArrivedAtCrisis()*, *forwardPoliceCarCrisisObjectiveComplete()*, *forwardPoliceCarBackFromCrisis*, *forwardReportAccident()*.
  - Six ≪ *optional* ≫ methods are added to the *Dispatch service* class: *policeCarDispatchOrder()*, *dispatchOrderReceivedPoliceCar()*, *policeCarArrivedAtCrisis()*, *policeCarCrisisObjectiveComplete()*, *policeCarBackFromCrisis()*, *reportAccident()*.
  - One ≪ *optional* ≫ method is added to the *Police Car* class: *forwardDispatchPoliceCarOrder()*.

    The UML class diagram model for this variation point is presented in Figure 20.

Modelling the sequence diagrams:

The introduction of the *Vehicle Management* variation point affects steps 4,5 and 6 of the use case scenario described in Section 4 of the requirements document. Therefore, these use cases corresponding to these steps need to be modified, in order to reflect the specificities of the variation point.

Now, in order to model the existence and possible selection of the other variants and represent the optional behaviour each of these variants introduces, we make use of the *"optional interaction fragment"* that exists in UML sequence diagrams. The semantics

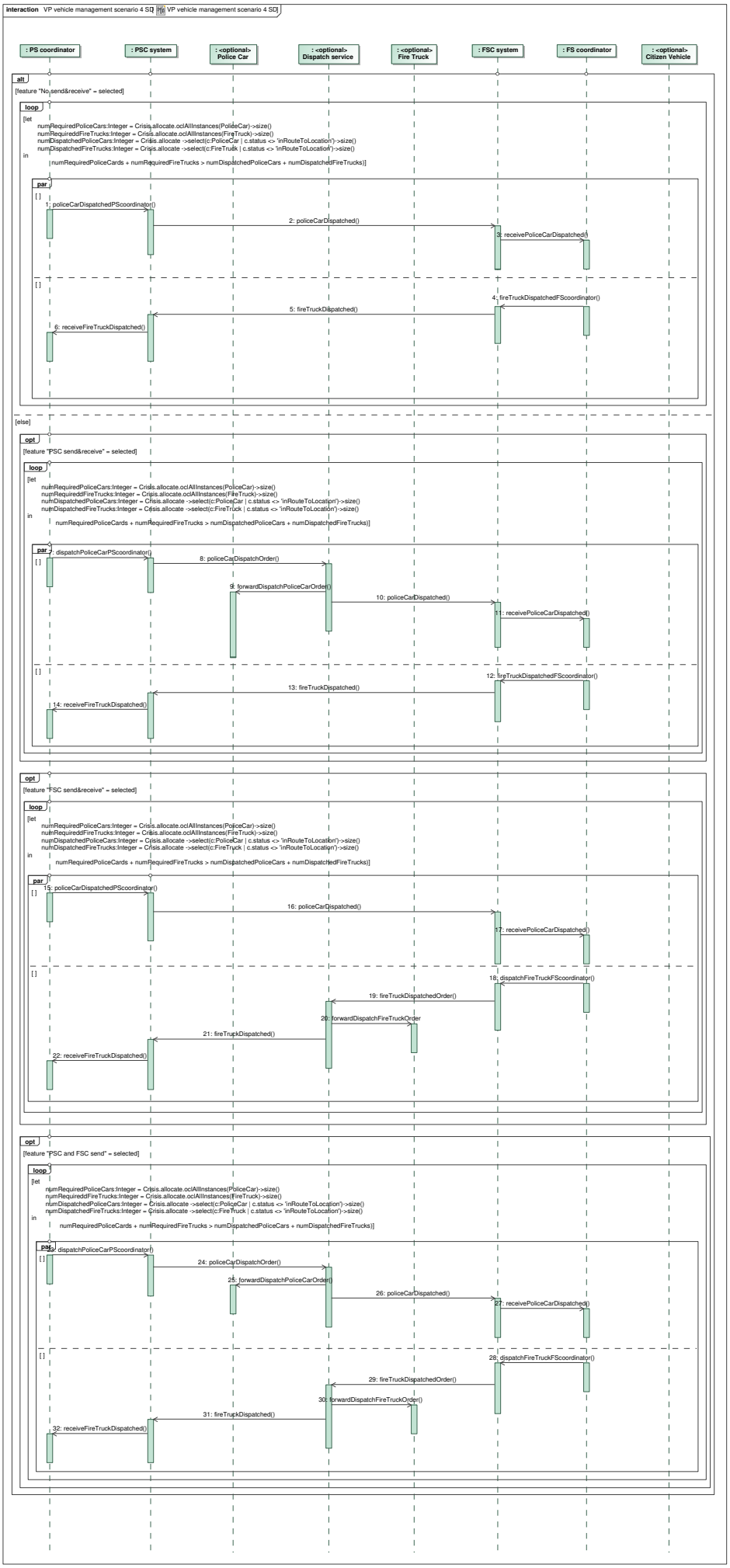


Figure 21: Vehicle Management variation point - sequence diagram for scenario 4.

of this concept is the following: it contains a guard condition that evaluated to either true or false. If the guard condition evaluates to true, the behaviour and the interactions described in the optional fragment are present in the described sequence diagram. In case the guard evaluates to false, then the described sequence diagram will not contain the behaviour described in the optional fragment. We use these optional fragments in the following manner: we describe, in an optional fragment, the specific behaviour introduced by a certain feature; we assign to such an optional fragment a guard condition that evaluates to true if that feature is selected. In the following, we briefly present how each of the sequence diagrams corresponding to the steps mentioned above are created:

Scenario 4 : we create the sequence diagram corresponding to this scenario starting from the sequence diagram of the reference variant. We create an "alternative" interaction fragment. In the first operand of this alternative fragment, we add the normal behaviour of the reference variant, which actually corresponds to the selection of the "No send and receive" variant in the feature diagram. The guard of this operand is *feature "No send and receive" = selected*. In the second operand of the alternative interaction fragment, we insert three optional interaction fragments, used to capture the variability. The first optional interaction fragment created has the guard condition *feature "PSC send and receive" = selected*. It encapsulates the specific message exchanges introduced by this feature. The same procedure is repeated for the second and third optional interaction fragments, which correspond to the remaining variants of the Vehicle management variation point: *FSC send and receive* and *PSC and FSC send*. The resulting sequence diagram is presented in Figure 21.

Scenario 5 : as for the modelling of scenario 4, we start from the sequence diagram of the reference variant for this scenario, which actually corresponds to the *"No send and receive"* feature. We use the same idea of the alternative interaction fragment, with the first operand corresponding to the *"No send and receive"* feature and describing its behaviour. For this scenario, only the variants *"PSC send and receive"* and *"FSC send and receive"* will have an impact and introduce new behaviour. Therefore, in the "else" part of the alternative fragment, we introduce two new optional interaction fragments containing the specific behaviour introduced by these two features, and set appropriate guards for these fragments. Selection of the feature *"PSC and FSC send"* has no impact on this scenario, so no optional fragment corresponding to it has to be introduced. The final sequence diagram for this scenario is presented in Figure 22.

Scenario 6 : the modelling of this scenario closely resembles the one for scenario 5. New optional interaction fragments need to be introduced only for variants *"PSC send and receive"* and *"FSC send and receive"*. The result is presented in Figure 23.

Scenario 7 : the modelling of this scenario closely resembles the one for scenario 5. New optional interaction fragments need to be introduced only for variants *"PSC send and receive"* and *"FSC send and receive"*. The result is presented in Figure 24.

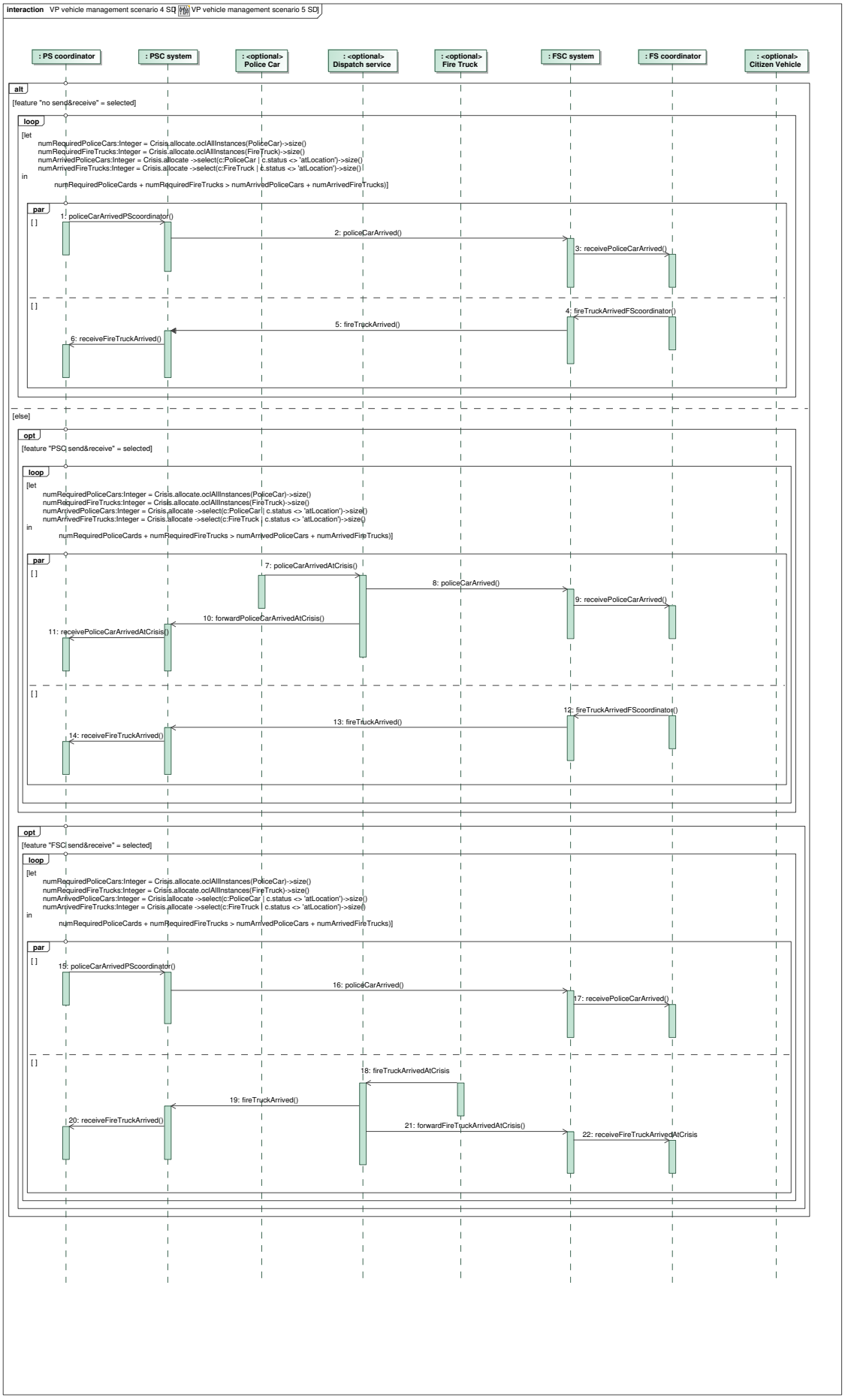


Figure 22: Vehicle Management variation point - sequence diagram for scenario 5.

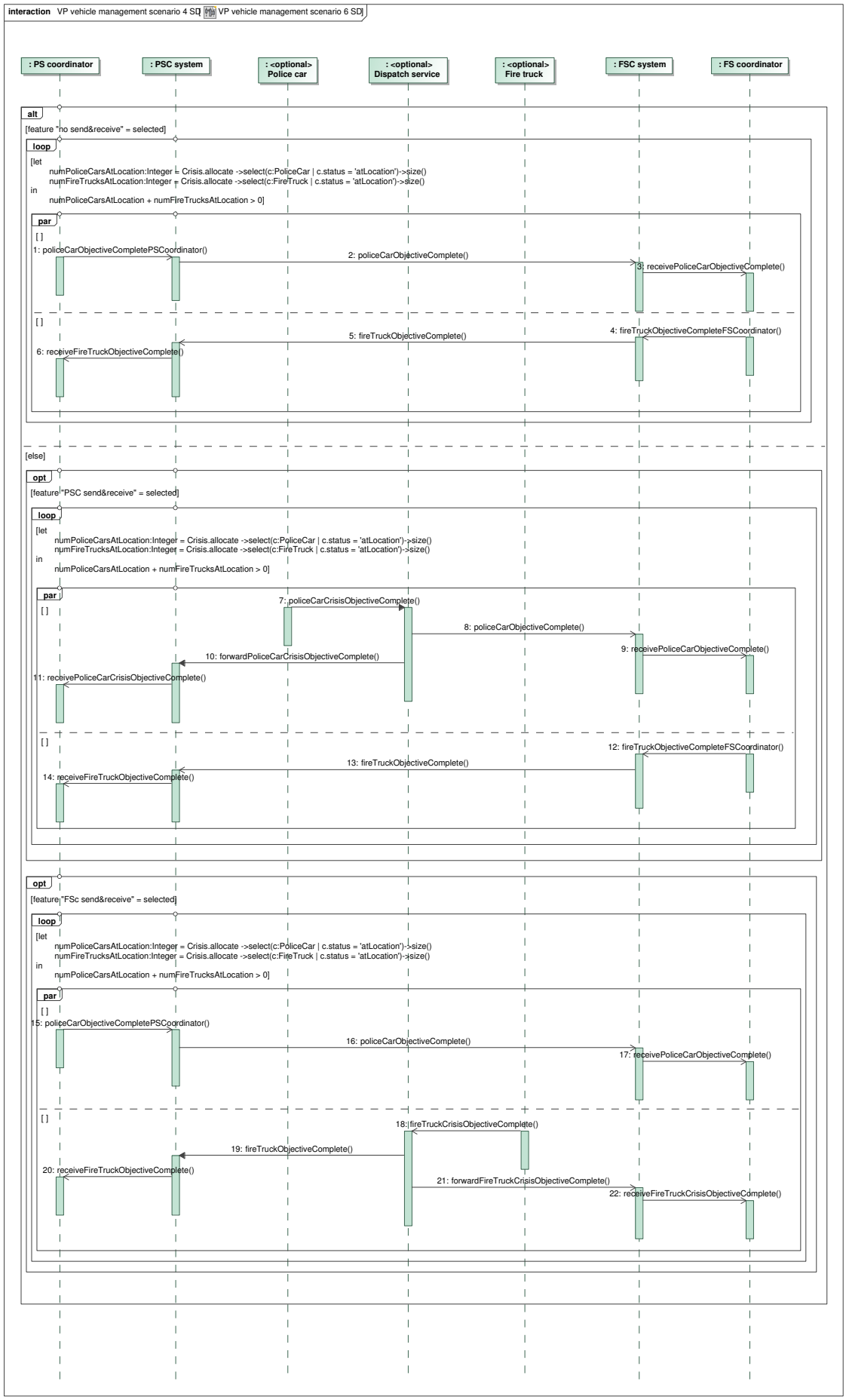


Figure 23: Vehicle Management variation point - sequence diagram for scenario 6.

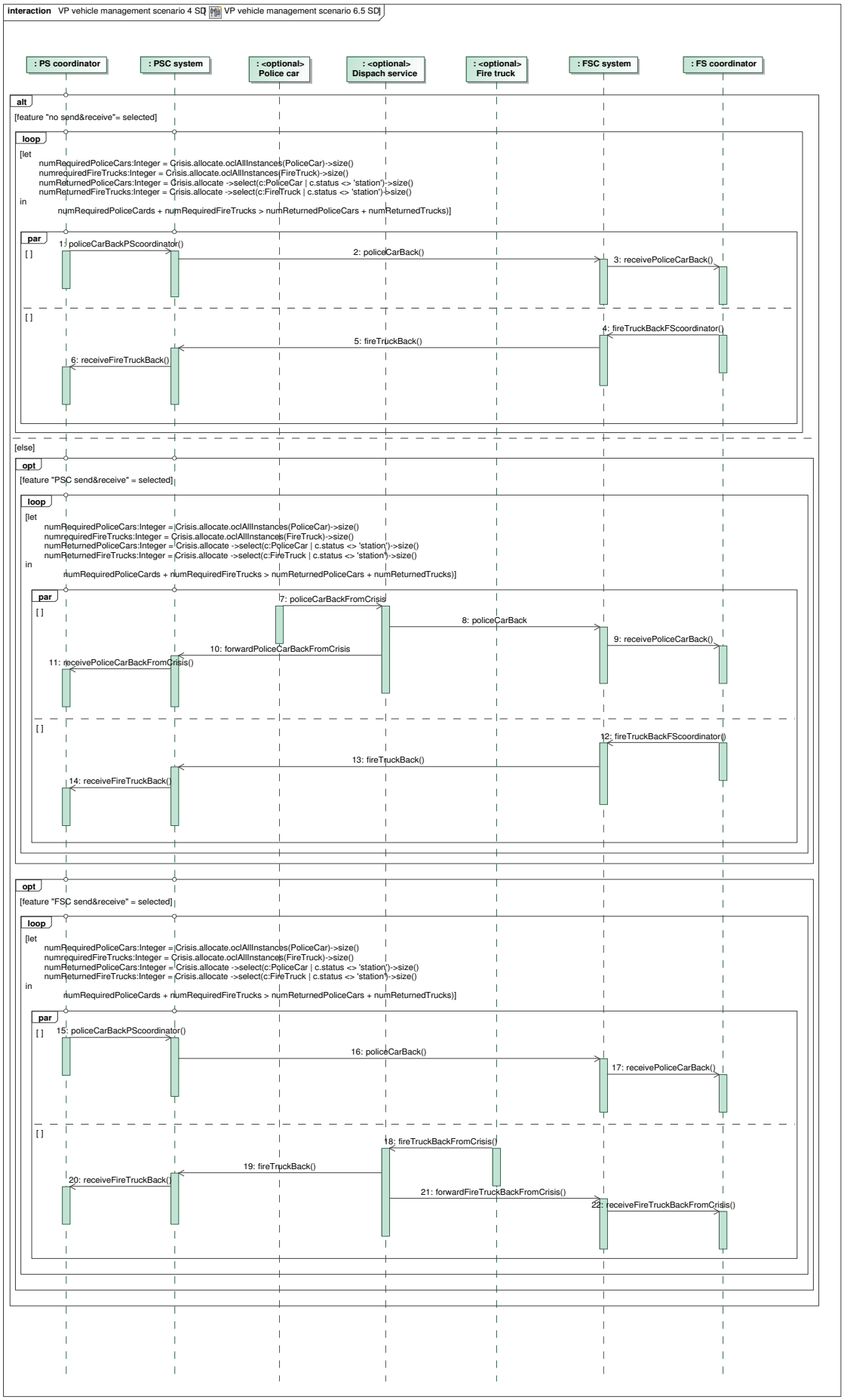


Figure 24: Vehicle Management variation point - sequence diagram for scenario 7.

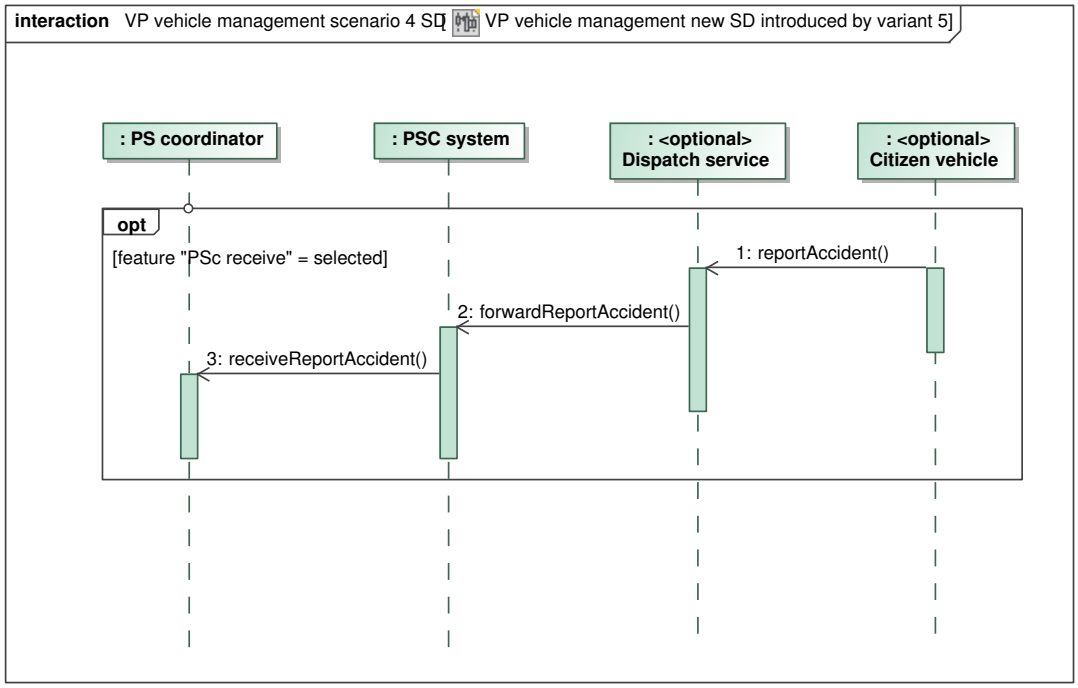


Figure 25: Vehicle Management variation point - new sequence diagram introduced by variant 5.

An interesting aspects that needs to be mentioned regarding the modelling of the Vehicle Management variation point concerns the *"PSC receive"* variant, which models the fact that the PSC can receive reports of accidents from citizen vehicles. This behaviour is completely nee and does not appear at all in any of the scenarios described in section 4 of the requirements document. Therefore, the selection of this feature implies the creation of a new sequence diagram to describe this particular behaviour. This is represented in Figure 25.

Modelling the state machines:

We describe here how the state machines corresponding to this variation point were created. They are based on the state machines initially created for the reference variant and the sequence diagrams of the *Vehicle management* variation point described above. Eight state machines need to be created, one for each actor concerned by this variation point.

- **Dispatch service state machine**: the *dispatch service* actor appears only for this variation point. The state machine for it contains four *composite states*, each of them corresponding to one of the four variants of the *Dispatch service* variation point. Each such composite state describes how the behaviour of the dispatch service is influenced by the selection of a particular variant for this variation point. Figure 26 presents this state machine.

- **Police car state machine**: this state machine is created based on the sequence

Figure 26: Vehicle Management variation point - state machine for dispatch service.

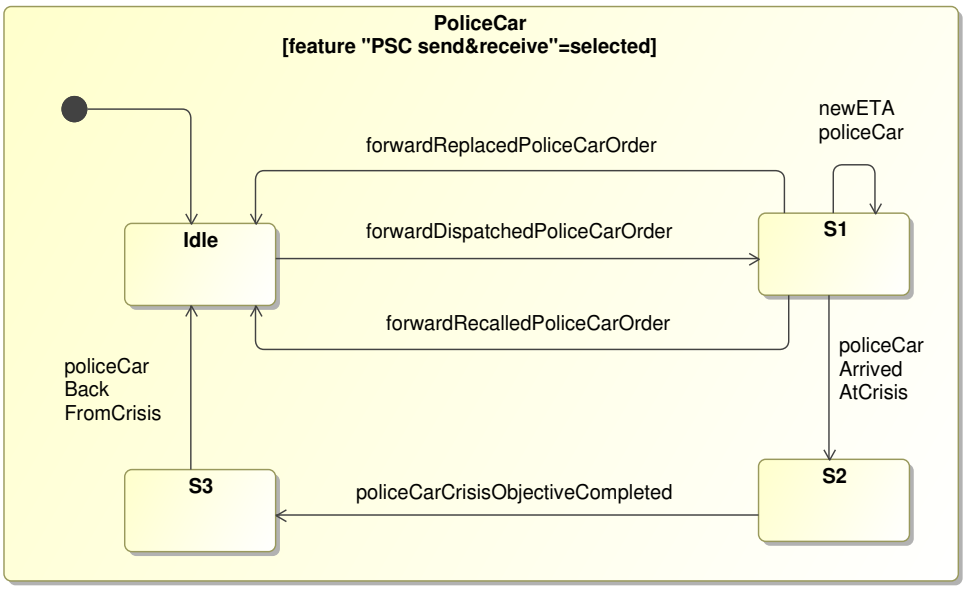


Figure 27: Vehicle Management variation point - state machine for police car.

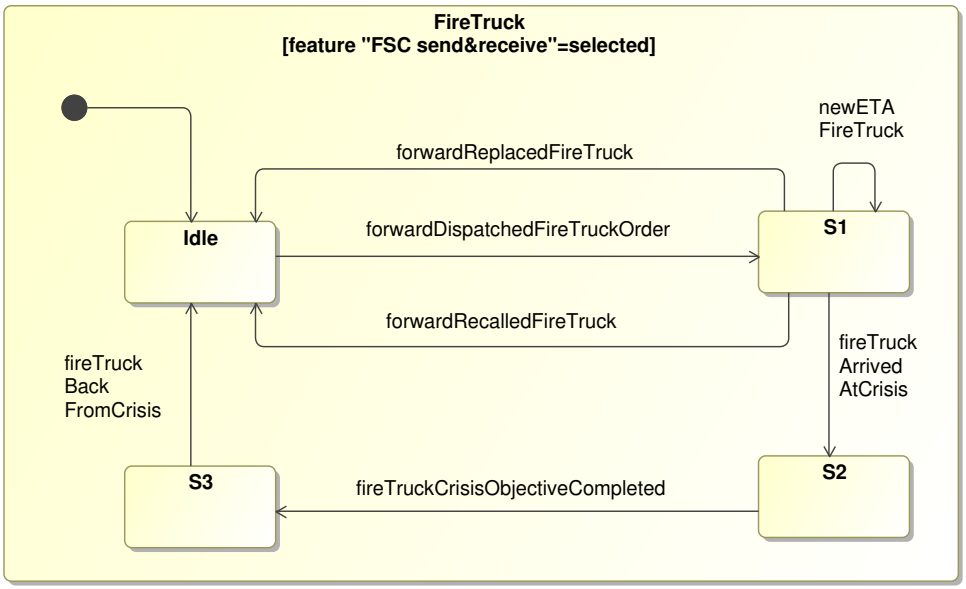


Figure 28: Vehicle Management variation point - state machine for fire truck.

diagrams created for the *Vehicle management* variation point. In these sequence diagrams, we identify all the messages sent or received by this actor. They will all be present in the state machine for this variation point. We can observe that all such messages appear only when the variant *"PSC send and receive"* is selected. Therefore only this variant influences the state machine for this variation point. The resulting state machine is presented in Figure 27.

- **Fire truck state machine**: the creation of this state machine is quite similar with the one previously presented for *Police Car*. Here, only the *FSC send and receive* variant has impact on the state machine. The resulting state machine is presented in Figure 28.

- *Citizen vehicle state machine*: the citizen vehicle only appears when the variant *PSC receive* is selected, and reflects the fact that citizen vehicle can report accidents to police cars and the PSC. The state machine is available in Figure 29.

- *Fire station coordinator state machines*: this actor is impacted by the *"FSC send and receive"* and *"PSC and FSC send"* variants. For simplicity and to increase the clarity of the models, we create a separate state machine for each of them. When feature *"PSC and FSC send"* is selected, states *"StandAlone-Arriving"* and *"Arriving"* from the original state machine need to be adapted. Figure 30 presents these changes. The state machine corresponding to the selection of feature *"FSC send and receive"* is available in Figure 31.

- *Police station coordinator state machines*: this actor is impacted by the *"PSC send and receive"* and *"PSC and FSC send"* variants of the variation point. For simplicity, we create a separate state machine for each of them. When feature *"PSC and FSC send"* is selected, states *"StandAlone-Arriving"* and *"Arriving"* from the original state machine are impacted. Figure 32 presents these changes. The state machine corresponding to the selection of feature *"PSC send and receive"* is available in Figure 33.

- *FSC system state machines*: the same variants as before impact also this actor. We present in Figure 35 the changes made when feature *"PSC and FSC send"* is selected, and in Figure 34 when feature *"FSC send and receive"* is selected.

- *PSC system state machines*: the same variants as before impact also this actor. We present in Figure 37 the changes made when feature *"PSC and FSC send"* is selected, and in Figure 36 when feature *"PSC send and receive"* is selected.

Creating the architectural model:

For this variation point, there is a significant change in terms of architectural style used, compared to the architectural model of the reference variant. Due to the presence of the *Dispatch service* class, the architectural model changes from a P2P system where the *PSC system* and *FSC system* communicate directly, to a system with a central node (the *Dispatch service*) that handles all communication between the *PSC system* and

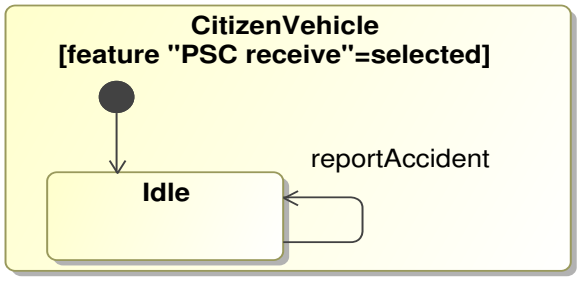


Figure 29: Vehicle Management variation point - state machine for citizen vehicle.

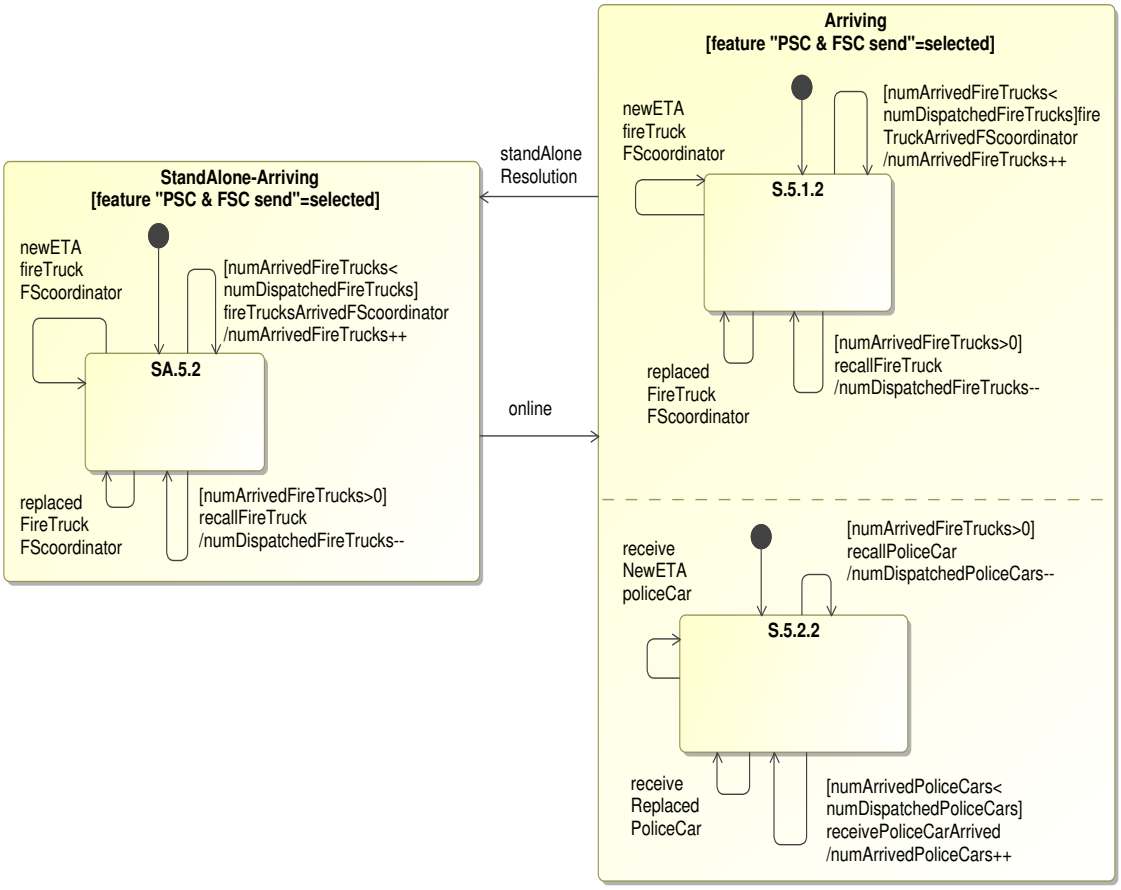


Figure 30: Vehicle Management variation point - state machine for FS coordinator when feature "PSC and FSC send" is selected.

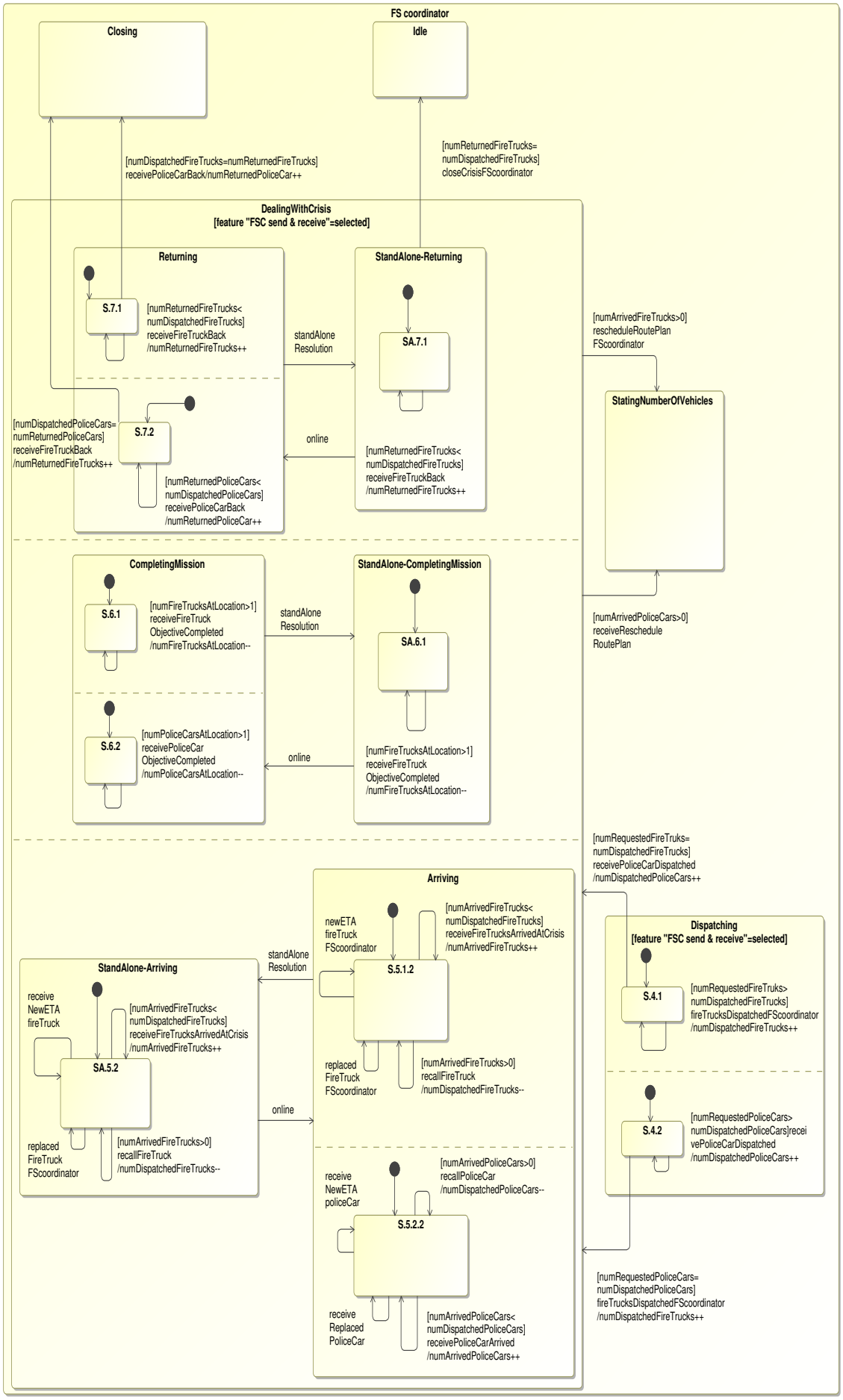


Figure 31: Vehicle Management variation point - state machine for FS coordinator when feature ”FSC send and receive” is selected.

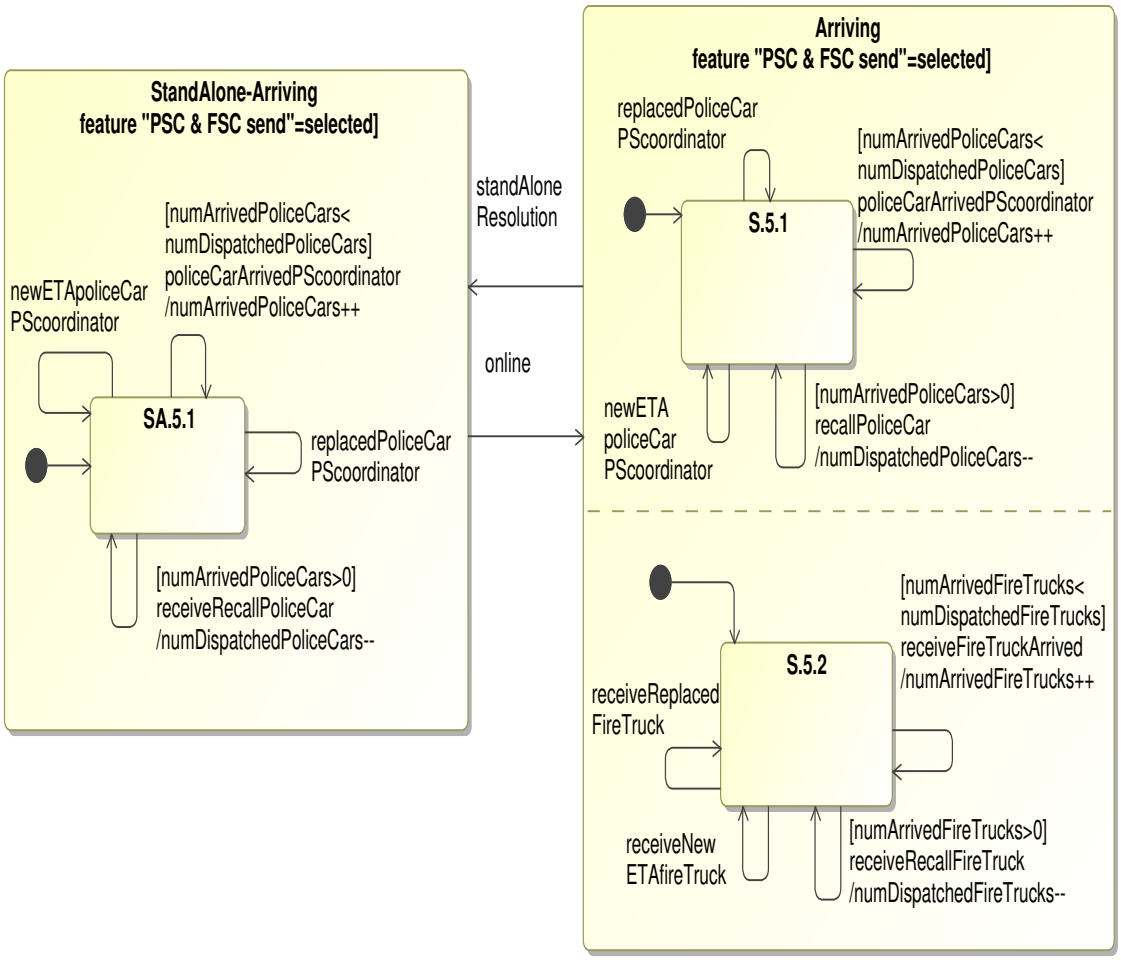


Figure 32: Vehicle Management variation point - state machine for PS coordinator when feature ”PSC and FSC send” is selected.

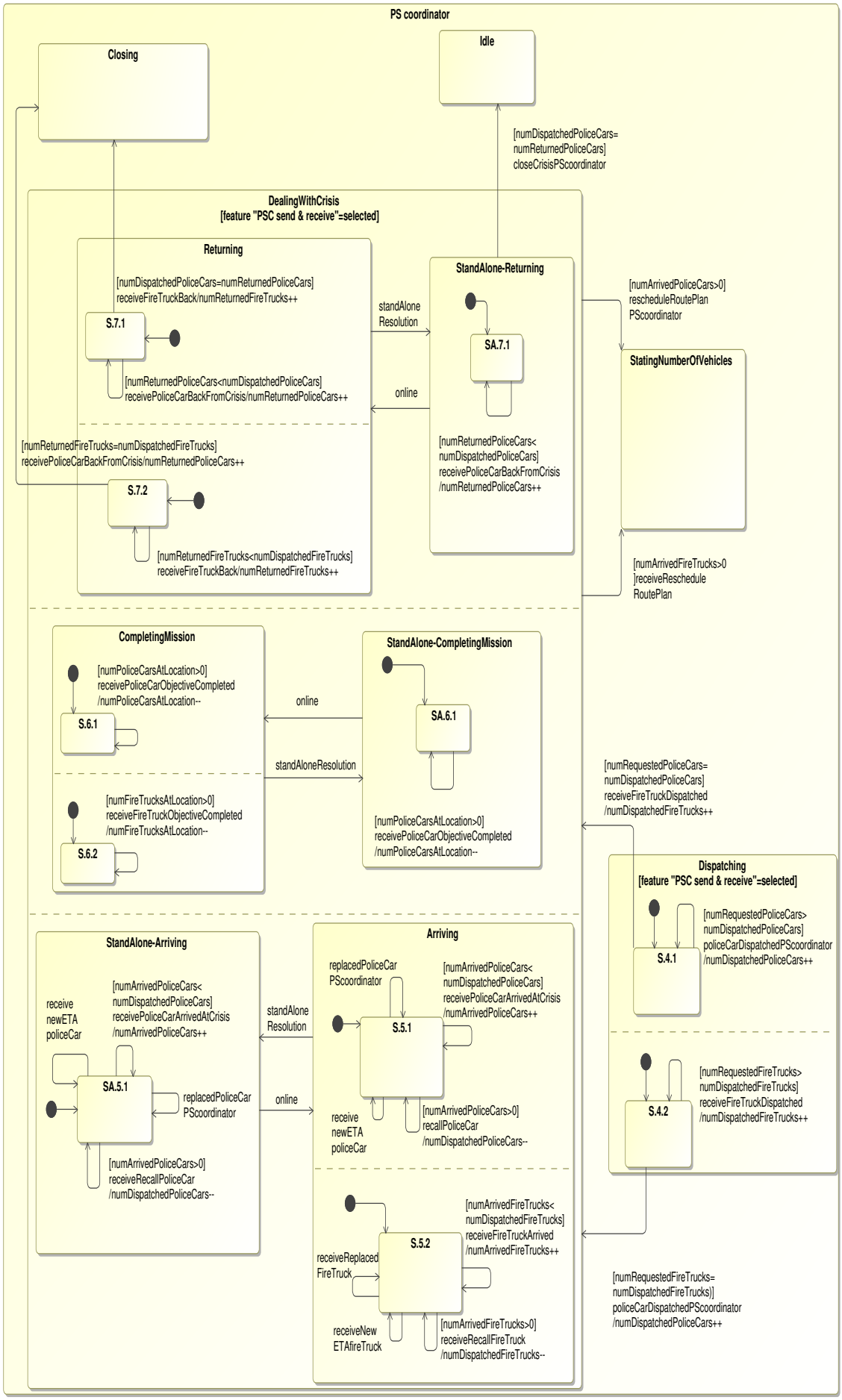


Figure 33: Vehicle Management variation point - state machine for PS coordinator when feature ”PSC send and receive” is selected.

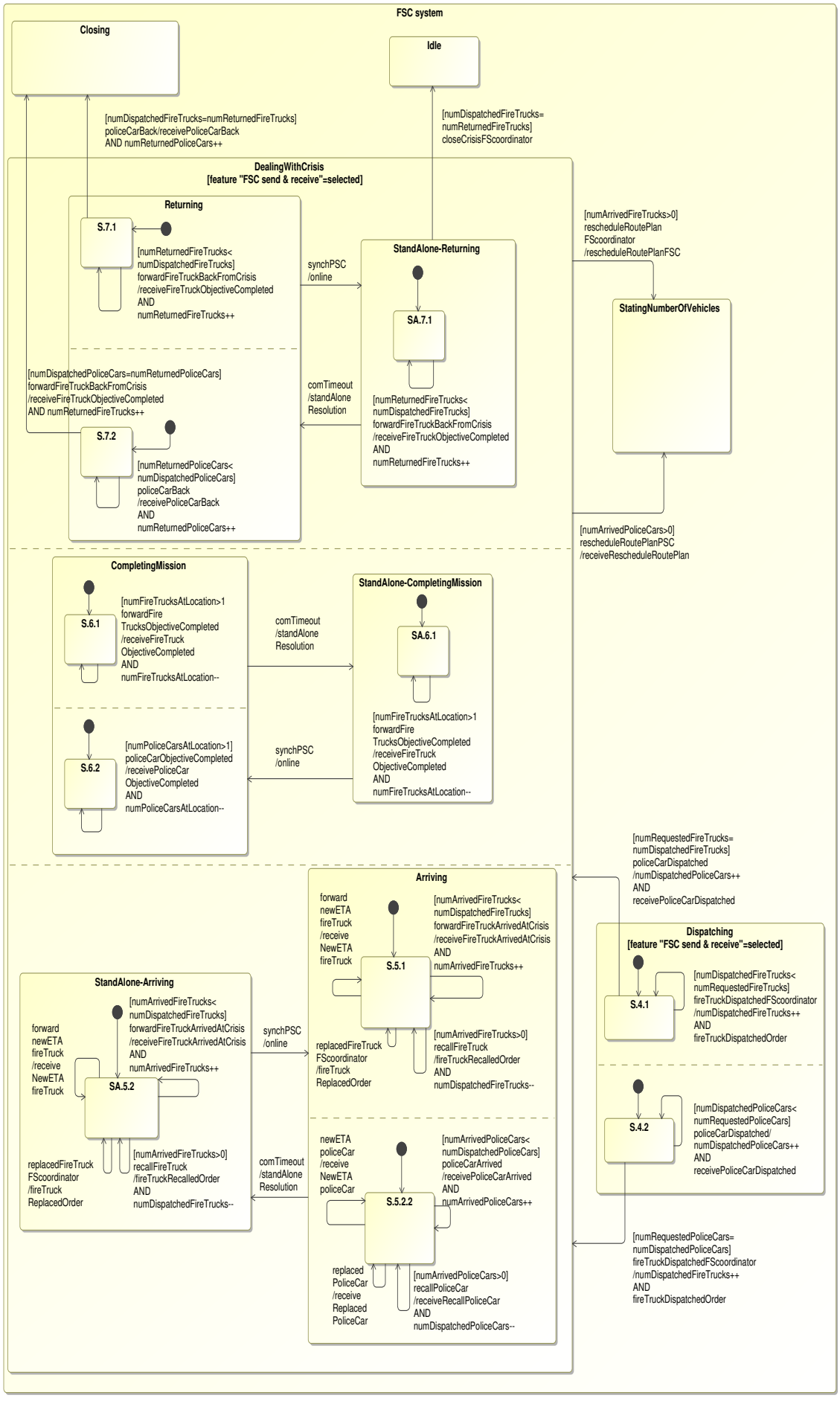


Figure 34: Vehicle Management variation point - state machine for FSC system when feature "FSC send and receive" is selected.

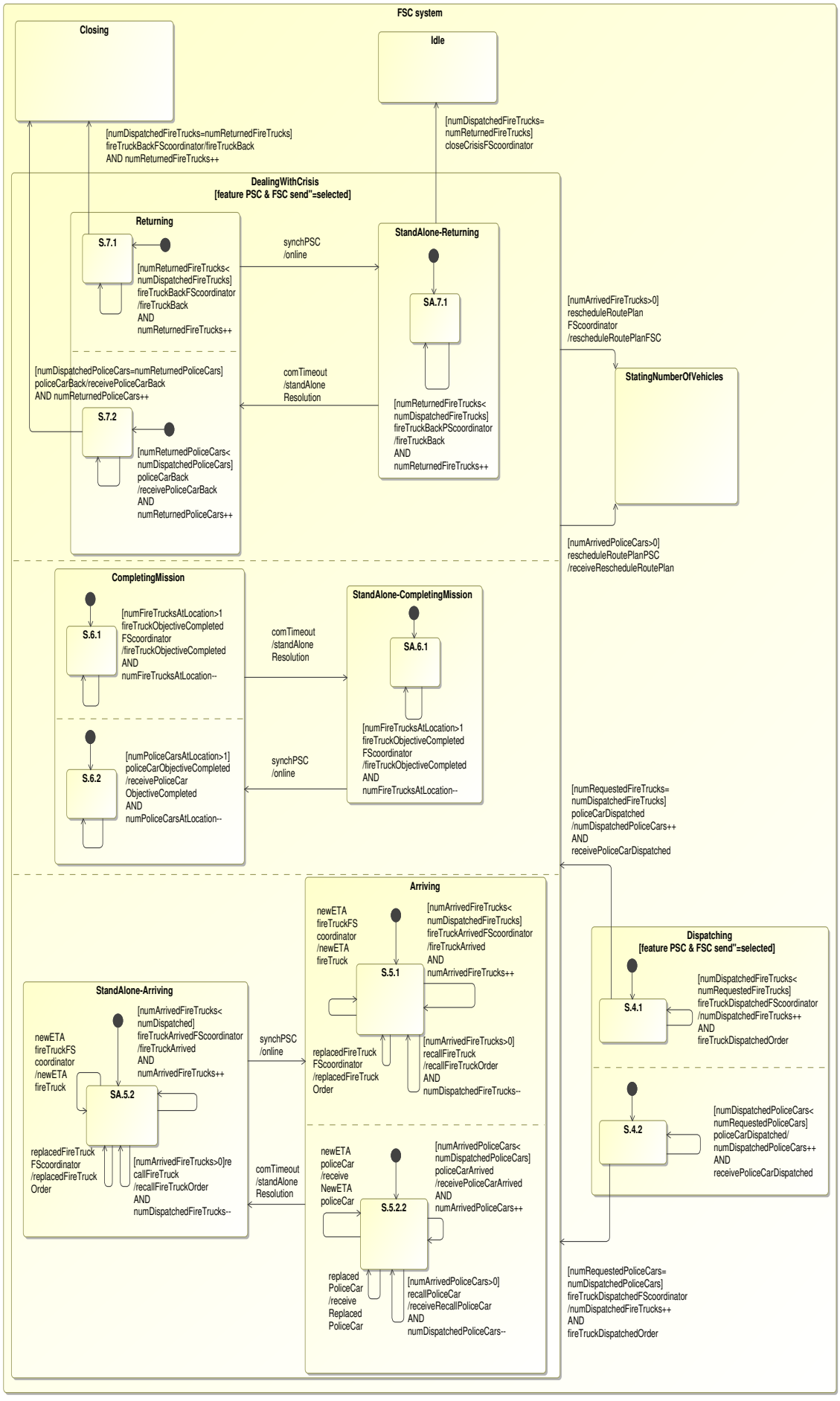


Figure 35: Vehicle Management variation point - state machine for FSC system when feature "PSC and FSC send" is selected.

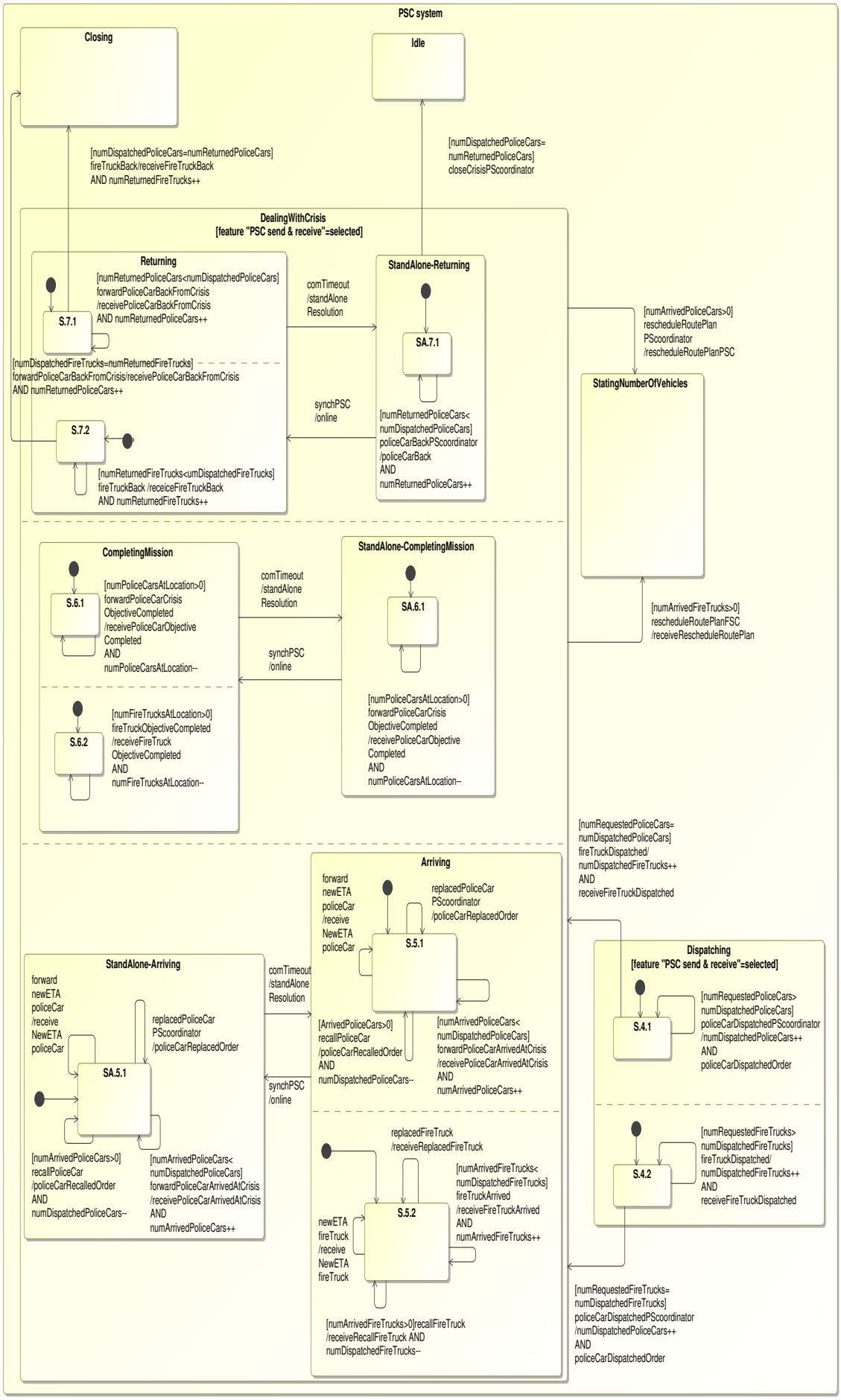


Figure 36: Vehicle Management variation point - state machine for PSC system when feature "PSC send and receive" is selected.

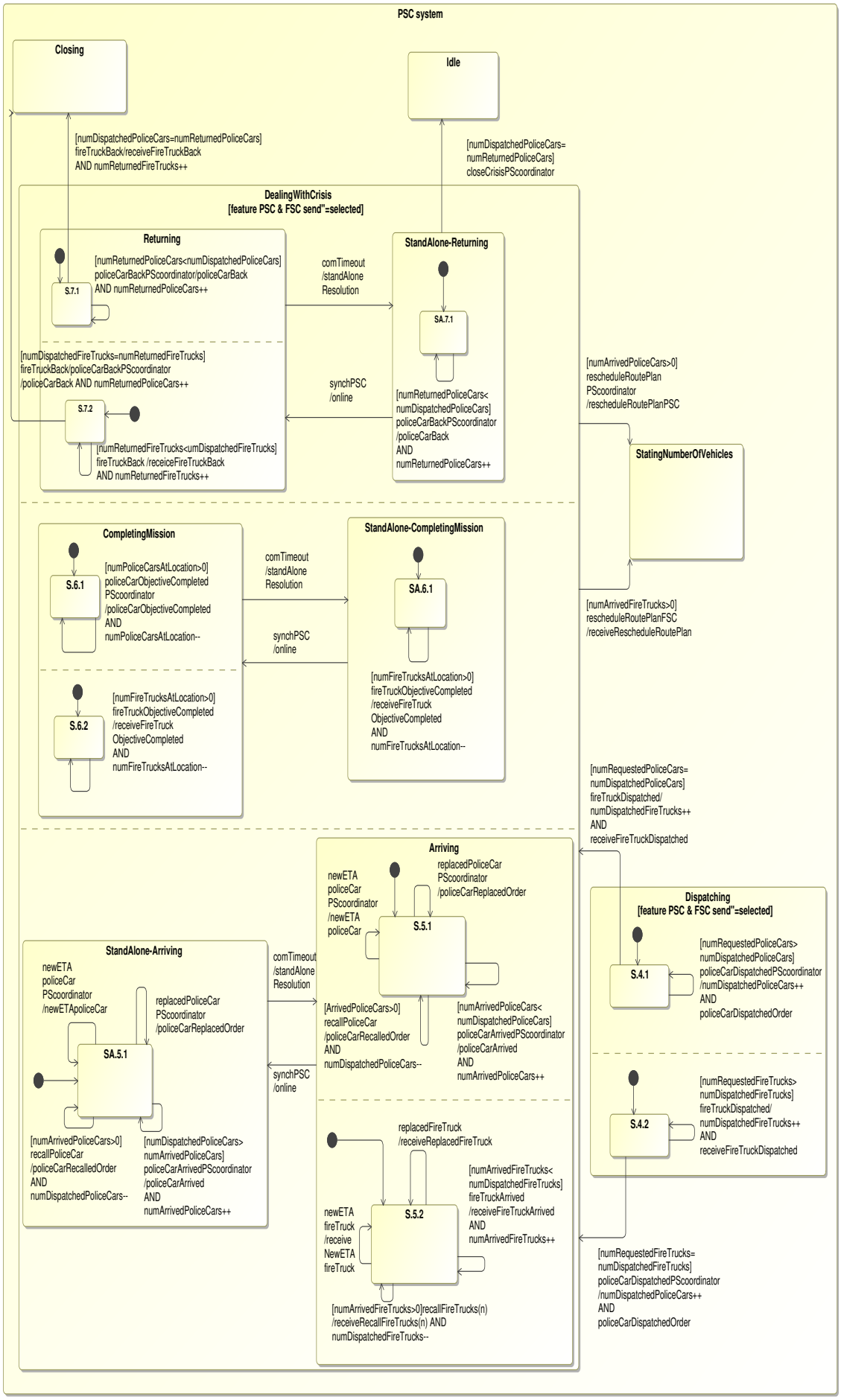


Figure 37: Vehicle Management variation point - state machine for PSC system when feature "PSC and FSC send" is selected.

FSC system components. Figure 38 presents the architectural model corresponding to this variation point.

3.5.2 Crisis Multiplicity variation point

Modelling the class diagram:

- to represent the fact that the system can now handle multiple crises, we change the multiplicity of the *"handle"* relation between the *bCrash system* class and the *Crisis* class at the end corresponding to the *Crisis* class, from *1* (as it is for the reference variant) to *1..**; this allows to model either the fact that only one crisis can be managed at a time, or that more of them could be handled.
- the reference variant of the bCrash system, modelled initially and described in the requirement doc, corresponds to the *"Single crisis"* feature, one of the variants of the *"Crisis multiplicity"* variation point
- several other changes need to be made, when the *"Multiple crisis"* feature (variant) is selected
- we add the *CrisisID* and *type* attributes to the *Crisis* class. The *CrisisID* attribute is the most important, as it represents that every crisis that can be handled by the system is unique, and allows to uniquely identify each crisis
- we also add the *isActive* Boolean attribute to the Crisis class; this will specify if the particular crisis is active or not in the system
- to represent the fact that a police car and a fire truck can be allocated to more than one crisis, we add the *multiplicity 1..** to the *"allocate"* relation connecting *Police car* and *Fire truck* to the *Crisis* class.
- we chose to model the fact that the system can handle multiple crisis in a simple way, one which requires a minimum amount of change from the reference variant already modelled: all the methods that belong to the *PS coordinator, PSC system, FS coordinator, FSC system* classes which refer to any action that concerns a crisis (eg exchange of crisis details between PSC and FSC, coordination of vehicle dispatch to crisis, arrival at crisis sight, objective completion and return from crisis, closing of the crisis) are parametrised with a *CrisisID parameter*, which will uniquely identify a crisis. In this way, all these actions are specific to a particular crisis. When a different crisis needs to be handled, this is simply modelled by a change in the value of the *CrisisID parameter*.

Taking all of this into consideration, the resulting class diagram for the *Crisis multiplicity* variation point is described in Figure 39.

Modelling the sequence diagrams:

As explained in the previous sub-section, the major difference between the UML class diagram of the reference variant and that of the *Crisis multiplicity* variation point

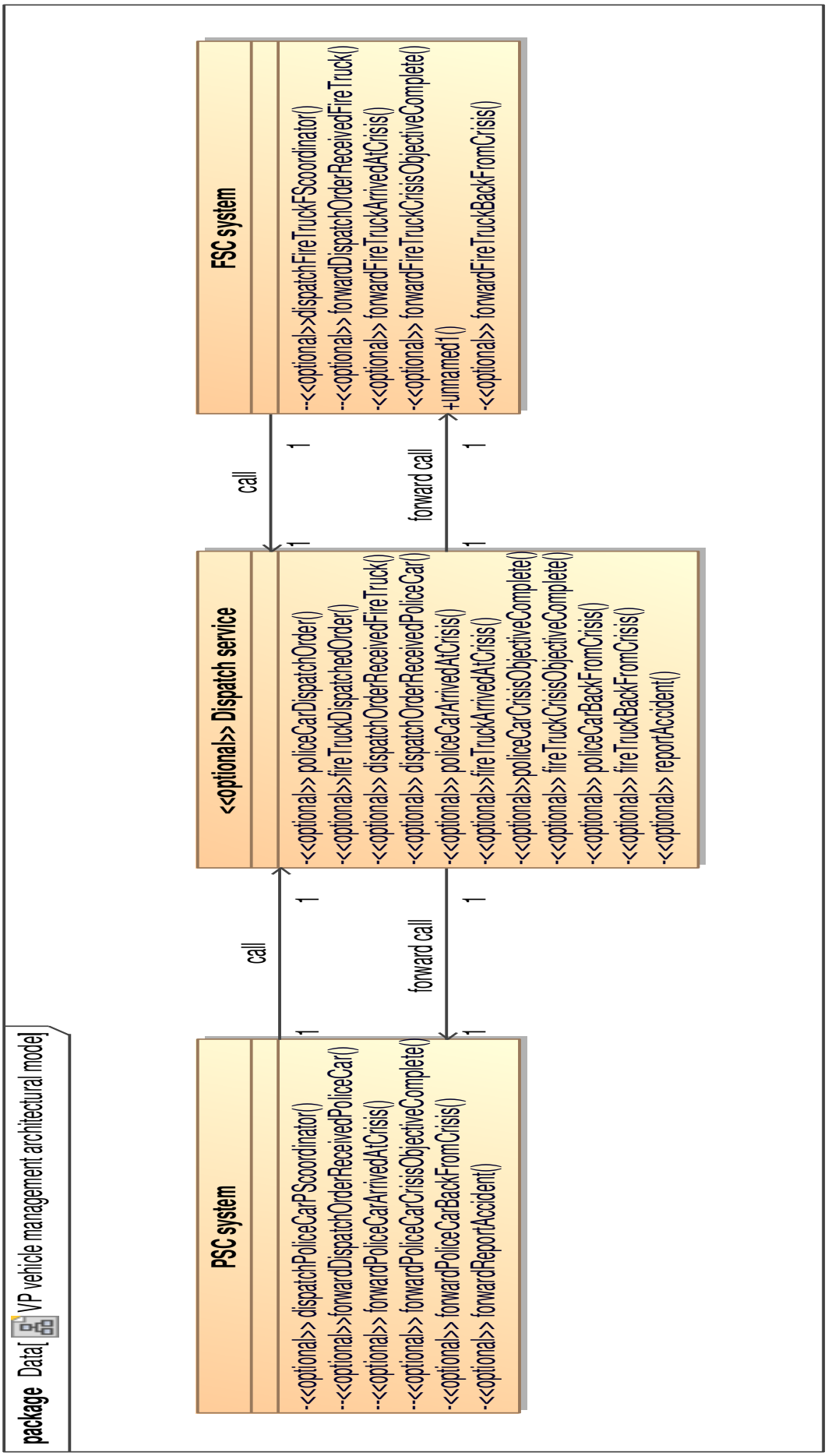


Figure 38: Vehicle Management variation point - architectural model

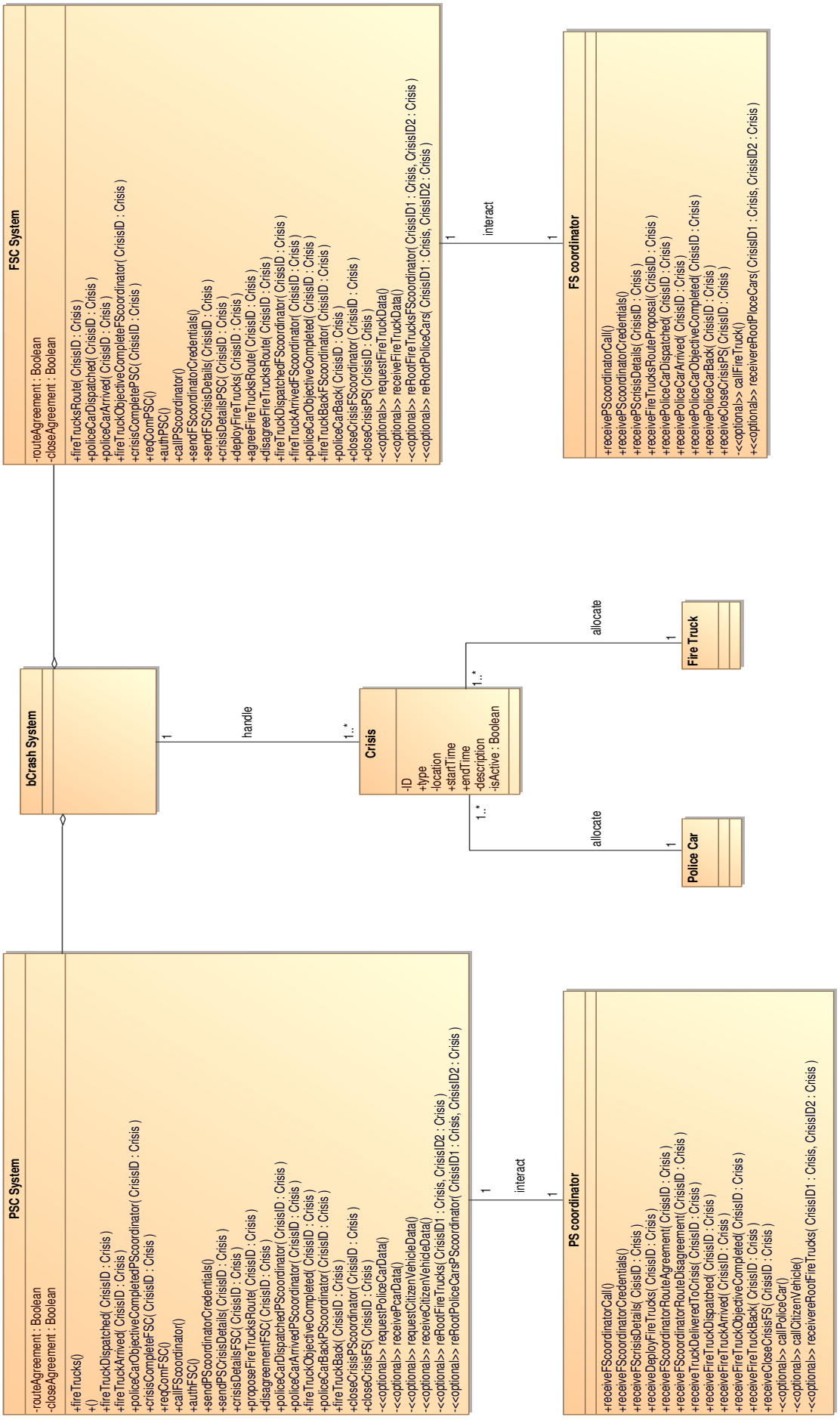


Figure 39: Crisis Multiplicity variation point.

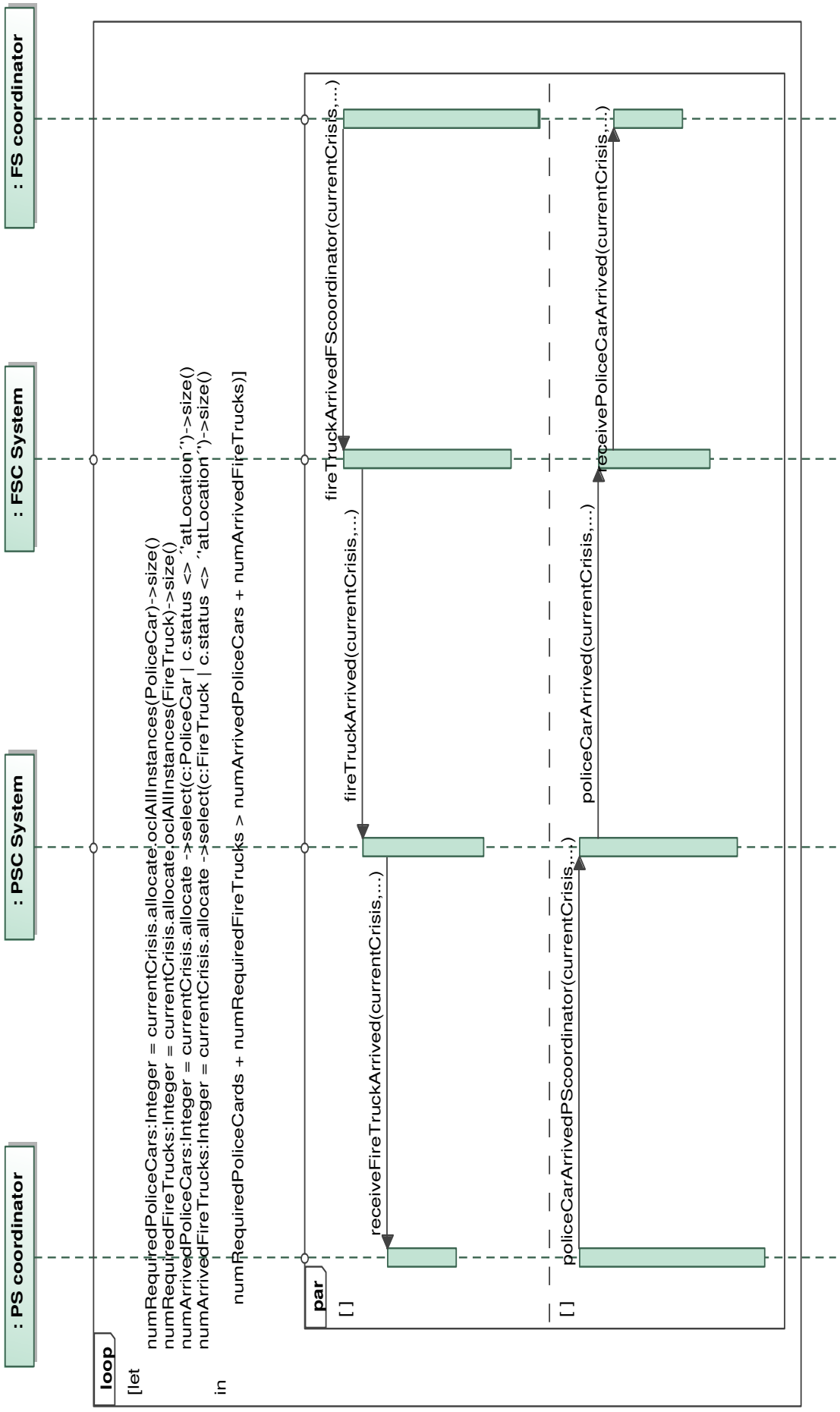


Figure 40: Crisis multiplicity variation point - sequence diagram for scenario 4.

is the parametrization of the methods with the *CrisisID* parameter. This reflects also on the sequence diagrams corresponding to this variation point. As a sequence diagram captures the message exchanges between different actors in the system, and because the same methods exist in the class diagrams of the reference variant and of the *Crisis multiplicity* variation point, then the behaviour that should be expressed in the sequence diagrams corresponding to this variation point coincides to the one described for the reference variant. Implicitly, there is no need to model different sequence diagrams for this variation point. The difference might be: the presence of the *CrisisID* parameter in case the system handles multiple crises, but this kind of information doesn't need to be captured in a sequence diagram and a small adaptation of the OCL conditions from the sequence diagrams, to reflect the fact that we are referring to the current crisis ou of the many possible crises that can be handled. We only exemplify this on the sequence diagram corresponding to scenario 4, presented in Figure 40. The sequence diagrams corresponding to scenarios 5-7 are the same as the ones described for the reference variant, with the small modifications already explained for scenario 4.

Creating the architectural model:

The architectural model corresponding to this variation point is similar to the one of the reference variant. The same P2P architectural style is used, the only small difference being the methods that appear in the *FSC system* and *PSC system* classes. The model is presented in Figure 41.

3.5.3 Communication protocol variation point

Modelling the class diagram:

- a first important remark is that this variation point is directly connected with the *Vehicle Management* variation point: existence of a communication management system (facility) is a pre-requisite for the existence of this variation point. As a consequence, we start to model the class diagram for this variation point from the class diagram done for the *Vehicle Management* variation point, which we will extend and adapt to the particular needs of this variation point

- there are 2 variants that need to be supported by the model for this variation point: SOAP communication and SSL communication. The easiest way to support this variation is to make use of the *abstraction* and *inheritance* mechanisms available in UML class diagrams. We therefore first transform the classes *PSC system, FSC system, Dispatch service, Police Car, Fire truck, Citizen Vehicle* into *abstract classes*. They will contain some abstract methods also tagged with the stereotype optional which refer to the communication protocol. Then, for each of these abstract classes, we add 2 concrete classes (e.g. *PSC system SOAP* and *PSC system SSL* for the abstract class *PSC system*), which inherit from the abstract class, and will specialize it with a particular behaviour. Each of these concrete classes (one corresponding to the SOAP implementation and one to the SSL implementation), *have methods that override the ones defined in the abstract class*

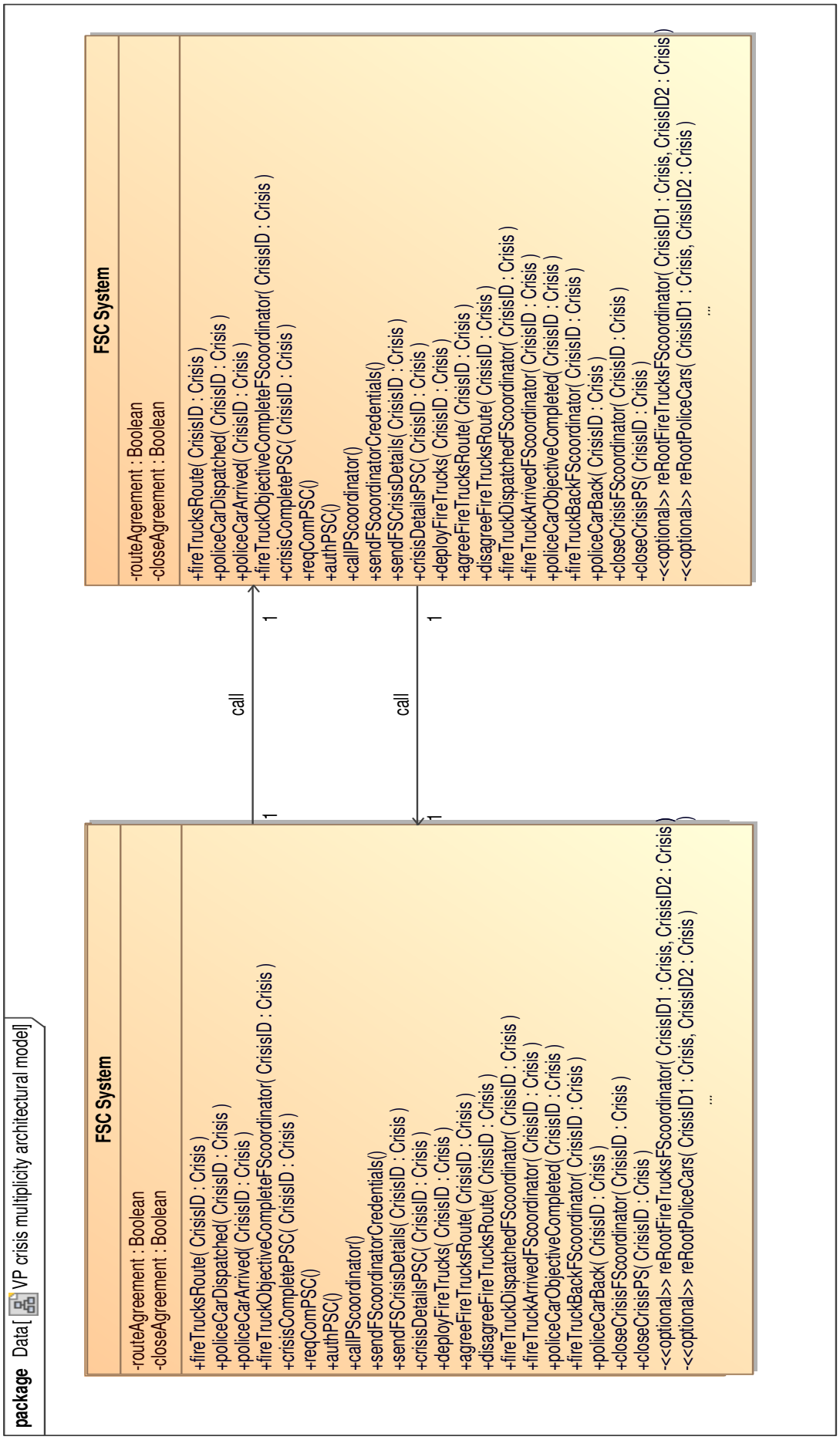


Figure 41: Crisis multiplicity variation point - architectural model

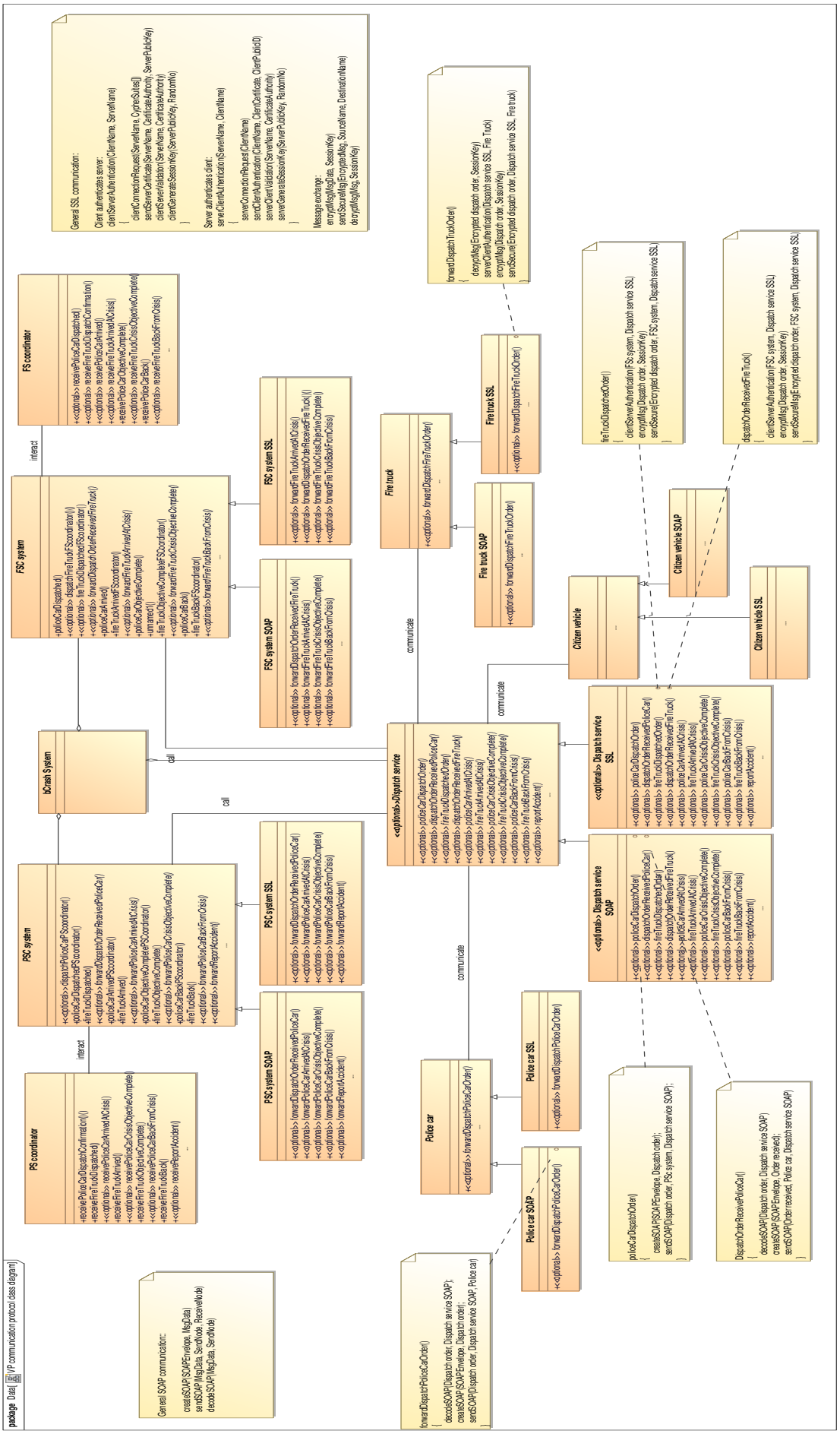


Figure 42: Communication protocol variation point.

with concrete methods that implement either the SOAP or the SSL version of the method.

- as an example, take the *PSC system abstract class* containing the forwardPoliceCarArrivedAtCrisis() abstract method: it has two concrete classes *PSC system SOAP* and *PSC system SSL* which inherit from it. Each of these classes contains a method forwardPoliceCarArrivedAtCrisis() which overrides the initial abstract method defined in the abstract class. To implement the override mechanism, the name of the methods in the concrete classes needs to be the same with the method in the abstract class which they override. Nevertheless, the *forwardPoliceCarArrivedAtCrisis( )*method from the *PSC system SOAP* class will differ in terms of actual implementation (implements a SOAP message exchange) from its counterpart from *PSC system SSL* class (implements an SSL message exchange).

- during the SPL product derivation phase, depending on the choices made by the user, for each of the abstract classes defined in this class diagram model, one of the concrete classes will be chosen to actually implement it.

- there are 2 UML notes in the model of the *Communication Protocol* variation point which present the general methods used for SOAP communication and for SSL communication. These general methods are then used to implement the SOAP or SSL versions of the methods from the PSC system, FSC system, Dispatch service, Police Car, Fire Truck, Citizen Vehicle classes. We provide this information in order to ease the understanding of the model.

- take for example the forwardDispatchPoliceCarOrder() method defined in *Police car SOAP*. This is its SOAP implementation:

Listing 1: Methods of class CAA to be used by the programmer.

```
forwardDispatchPoliceCarOrder()
{
decodeSOAP(Dispatch order, Dispatch service SOAP);
createSOAP(SOAPEnvelope, Dispatch order);
sendSOAP(Dispatch order, Dispatch service SOAP, Police car)
}
```

- as another example, we present below the SSL implementation of the *forwardDispatchTruckOrder()* method from the *Fire truck SSL* class:

Listing 2: Methods of class CAA to be used by the programmer.

```
forwardDispatchTruckOrder()
{
decryptMsg(Encrypted dispatch order, SessionKey)
serverClientAuthentication(Dispatch service SSL, Fire Truck)
encryptMsg(Dispatch order, SessionKey)
sendSecure(Encrypted dispatch order, Dispatch service SSL, Fire truck)
}
```

- in the class diagram, we provide several other examples of methods and how they are implemented, by simply using the general methods defined in the UML notes added to the design. Using these examples and having the general method at disposal, it is simple a matter of changing some parameters in order to obtain the implementation of any other method present in the design

Modelling the sequence diagrams:

As for the class diagram model of this variation point, also the sequence diagram will closely depend to the sequence diagram of the *Vehicle Management* variation point, due to the existent dependency between them. The use of inheritance and overriding as mechanisms to capture variability in the class diagram of this variation point will impact the sequence diagrams. The actual sequence diagrams for this variation point are created by extending the one created for the *Vehicle Management* variation point. But, due to the fact that in the class diagram we have abstract methods overridden by concrete methods with the same name, in the corresponding sequence diagrams, the same behaviour will be captured. This means that at the level of the sequence diagram, there is no visible difference from the diagrams created for the *Vehicle Management* variation point. It should be clear though that even though the sequence diagrams for the *Communication protocol* are identical to the ones for *Vehicle Management*, they actually describe different behaviours: for the *Communication protocol*, the methods present in the sequence diagrams correspond to one of the two possible implementations offered in the class diagram for this variation point. The methods have the same name, but their implementation will be different, depending on the choice the user has made between either the SOAP or the SSL variant.

Creating the architectural model:

For this variation point, there is a significant change in terms of architectural style used, compared to the architectural model of the reference variant. Due to the presence of the *Dispatch service* class, the architectural model changes from a P2P system where the *PSC system* and *FSC system* communicate directly, to a system with a central node (the *Dispatch service*) that handles all communication between the *PSC system* and *FSC system* components. Figure 43 presents the architectural model corresponding to this variation point.

3.5.4 Confidentiality of data communications variation point

Modelling the class diagram:

- This variation point impacts every message exchange between the actors of the system. We start modelling its class diagram starting from the class diagram of the reference variant.

- This variation point has two possible variants: communication with *no encryption* or *encrypted communication.*

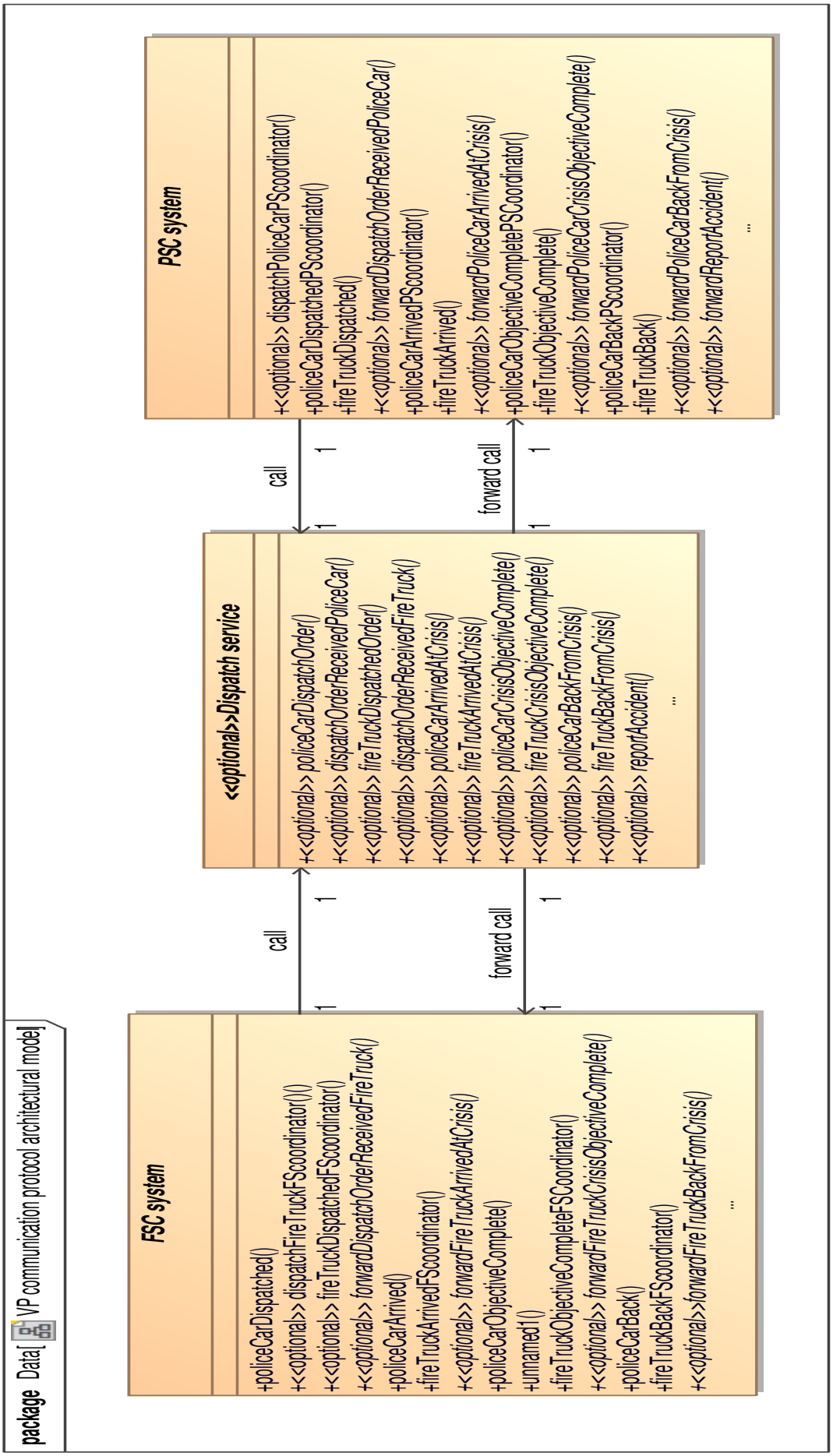


Figure 43: Communication protocol variation point - architectural model

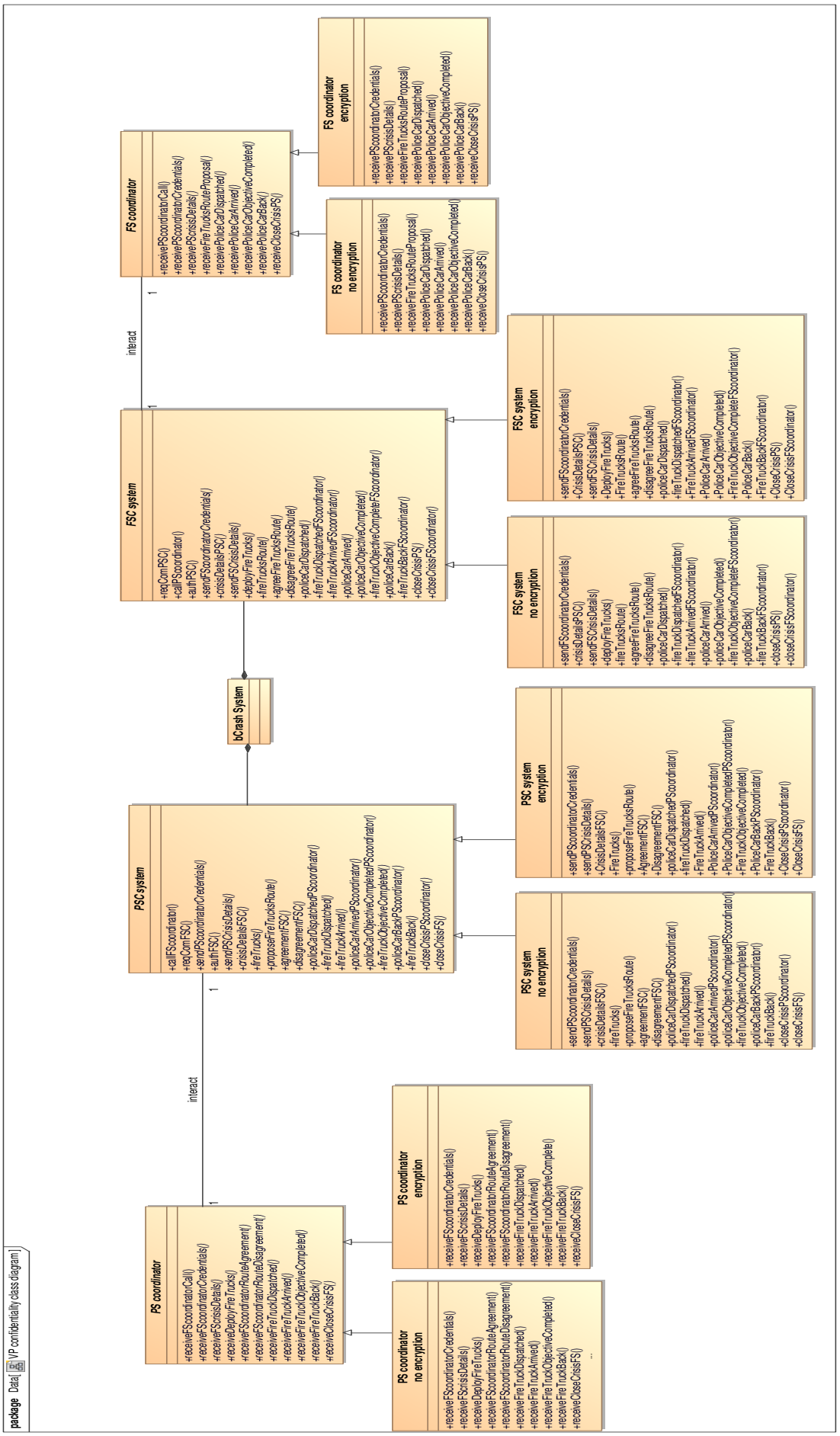


Figure 44: Confidentiality of data communications variation point.

- The *PS coordinator, PSC system, FSC system, FS coordinator* classes represent the actors of he system who exchange messages. Therefore, for these classes, we select all the methods that represent a message exchange between them.
- New abstract classes for the *PS coordinator, PSC system, FSC system, FS coordinator* are created. In each of them, the previously selected methods are added as abstract methods.
- Each of the abstract classes will have two concrete classes, that inherit from it, that will implement it. One corresponds to the version with no encryption, while the other to the version containing encryption of message exchanges.
- The overriding of methods mechanism is used: the concrete classes contain concrete methods that override the abstract methods defined in the parent abstract class, thus providing either the *encryption* or *no encryption* variants.

The resulting class diagram is available in Figure 44.

Modelling the sequence diagrams:

This type of variation impacts the functioning of the system in multiple places, as it applies every time there is a data communication between actors of the system. In the use case described in Section 4 of the requirements document, this variation points impacts steps 1-7. Implicitly, we should adapt all the sequence diagrams created for the reference variant, to capture the fact that communications between actors of the system be either without encryption (feature *Not encrypted* selected) or with encryption (feature *Encrypted* selected). The communication with no encryption actually corresponds to the reference variant. From the point of view of the sequence diagram, the encrypted communication behaviour will be the same as the one without encryption. For understanding why, we need to go back to the class diagram model of this variation point. Each actor of the system is represented by an abstract class which has two concrete classes that implement it and represent either the encrypted or non-encrypted versions. The methods of the concrete classes override the abstract methods in the parent abstract class, and therefore have the same name, but their actual implementation will be different. At the level of the sequence diagram, this means that the sequence diagram for the *Encrypted* variant of the *Data communication confidentiality* are exactly the same as for the *Non-encrypted* variant. The names of the methods and their sequencing is the same. Nevertheless, when deriving an actual product of the bCrash SPL, the behaviour modelled will be different, as the actual implementation of the methods is different for the two variants.

Creating the architectural model:

The same P2P architectural style is used also for this variation point. The corresponding model can be seen in Figure 45.

3.5.5 Authentication of system's users variation point

Modelling the class diagram:

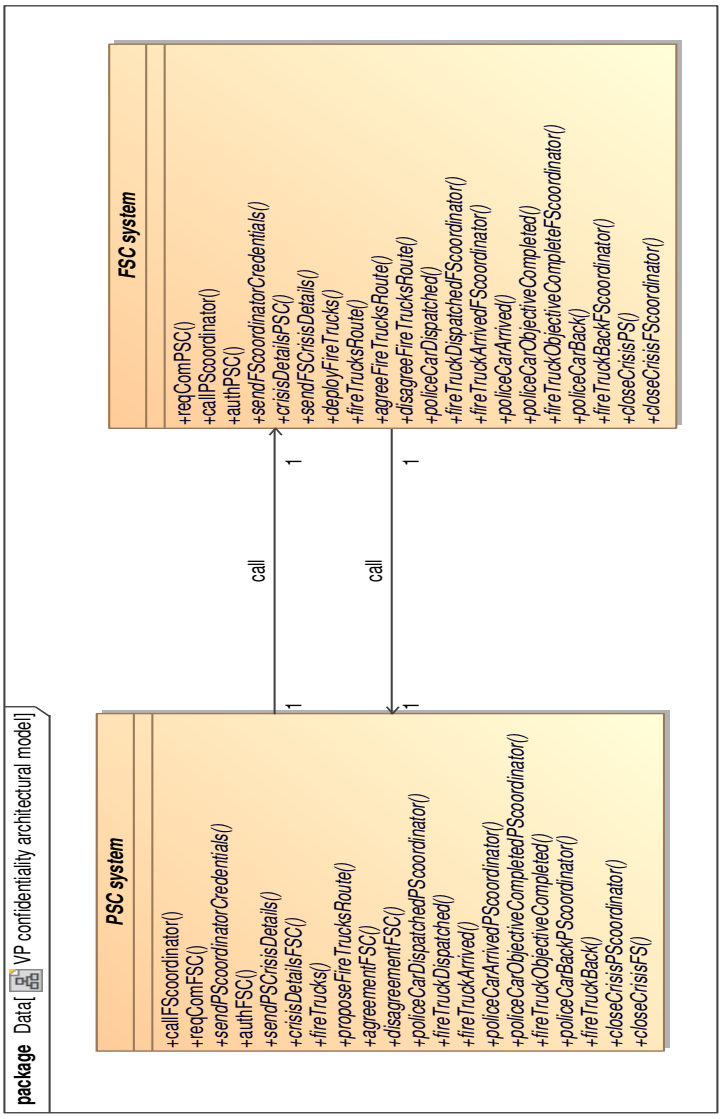


Figure 45: Confidentiality of data communication variation point - architectural model

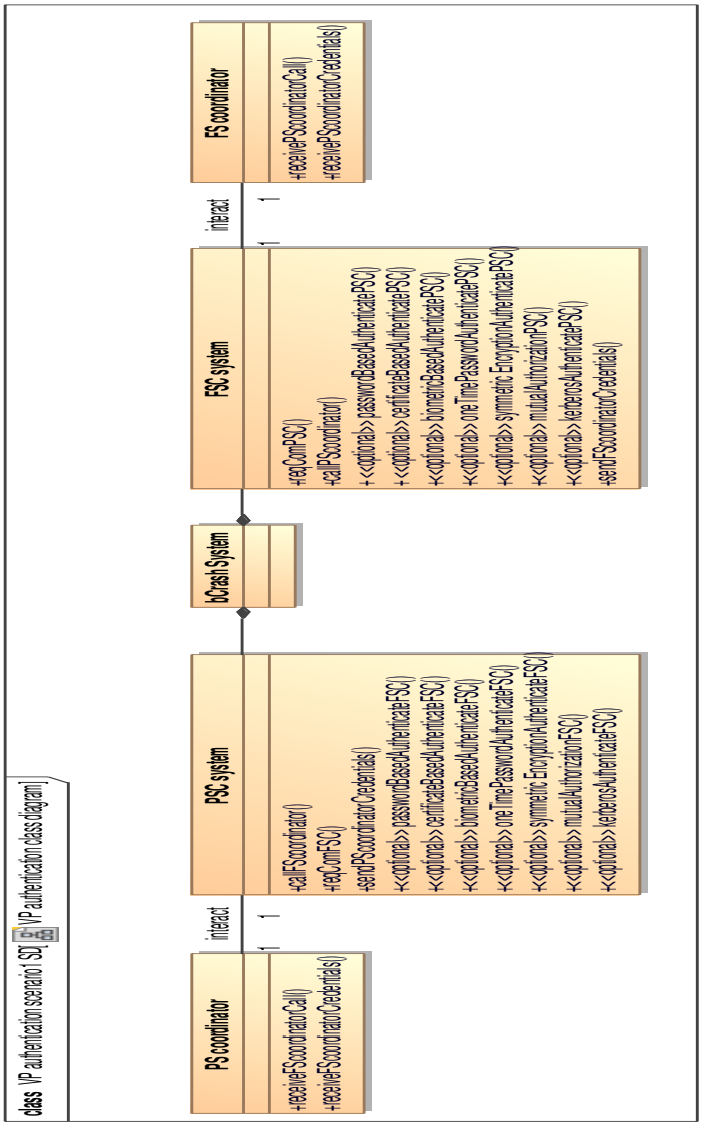


Figure 46: Authentication of system's users variation point.

- This variation point allows the users of the system to authenticate in different manners when working with the bCrash SPL system. In our case, this refers to the actual authentication of the PS coordinator to the FS coordinator and of the FS coordinator to the PS coordinator, when they establish communication and need to identify themselves.
- We start constructing the class diagram for this variation point starting from the class diagram of the reference variant
- The only thing that needs to be modified is to add in the *PSC system* and *FSc system* classes, new methods that perform the authorized identification of one actor to the other.
- For this purpose, we introduce one new method, tagged with the stereotype "optional", for each type of possible authorization defined (corresponding to each of the variants of the *Authentication* variation point)
- As an example, we add the following methods tagged "optional" to the *PSc system* class: *passwordBasedAuthenticateFSC(), certificateBasedAuthenticateFSC(), biometricBasedAuthenticateFSC(), oneTimePasswordAuthenticateFSC(), symmetricEncryptionAuthenticateFSC(), mutualAuthorizationFSC(), kerberosAuthenticateFSC().*

The resulting class diagram is presented in Figure 46.
Modelling the sequence diagrams:

- In the scenario described in Section 4 of the requirements document, the authentication of the PSC and FSC takes place at step 1. Therefore, this is the only step that will be affected by this variation point.
- We only need to model one sequence diagram for this variation point, describing scenario 1. For thsi, we start from the sequence diagram of scenario 1 corresponding to the reference variant.
- We identify that the messages *authPSC()* and *authFSC()* are the only ones in this scenario impacted by the variation point.
- We make use again of optional interaction fragments to capture the variability. We add seven such optional interaction fragments, corresponding to all the possible variants of the variation point. Each of these fragments has a guard condition evaluationg to true is the corresponding feature is selected.
- The optional interaction fragments contain only one message exchange, corresponding to the particular method that implements the particular type of authorization
- This procedure is applied for both the FSC and PSC authentication

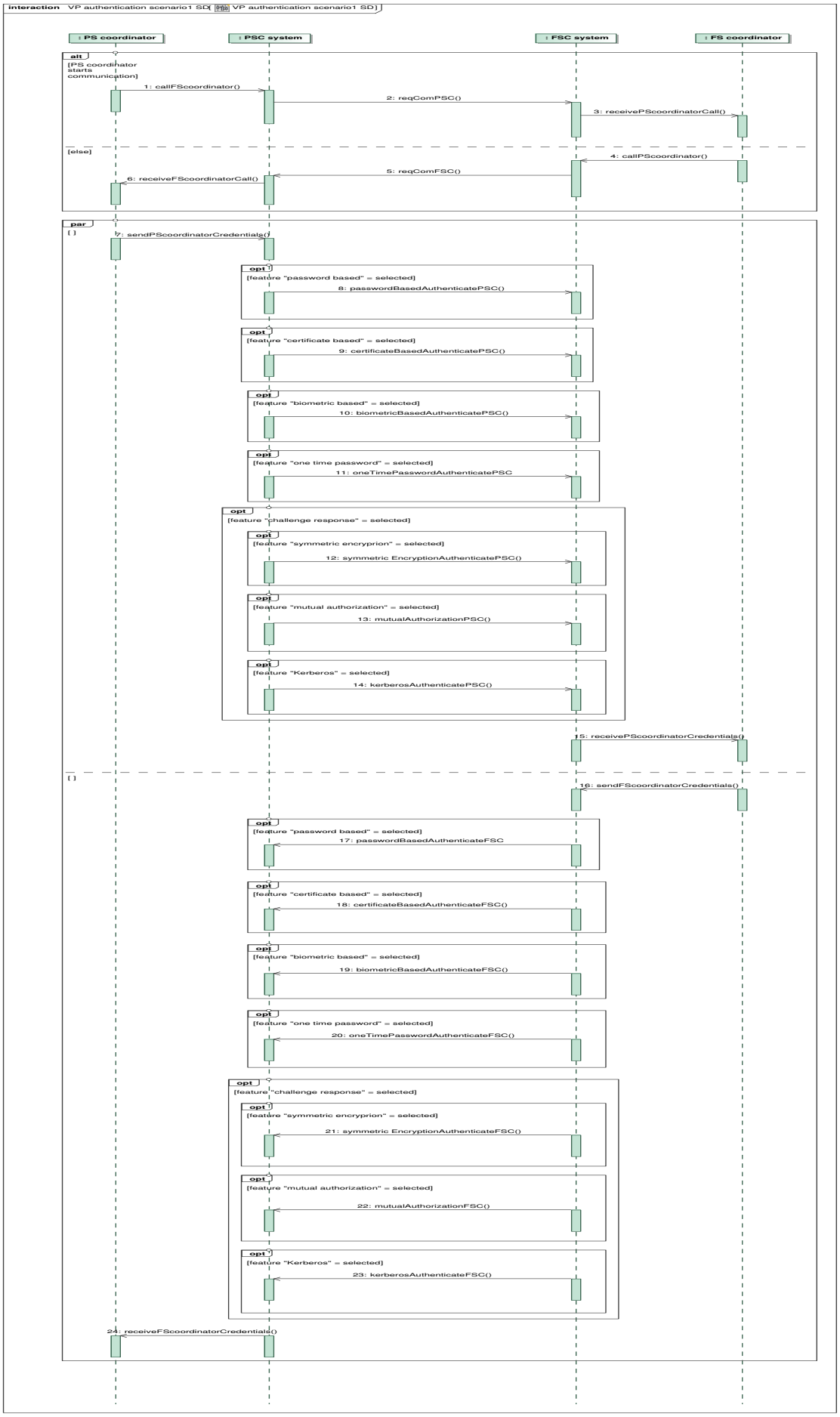


Figure 47: Authentication of system's users variation point - sequence diagram.

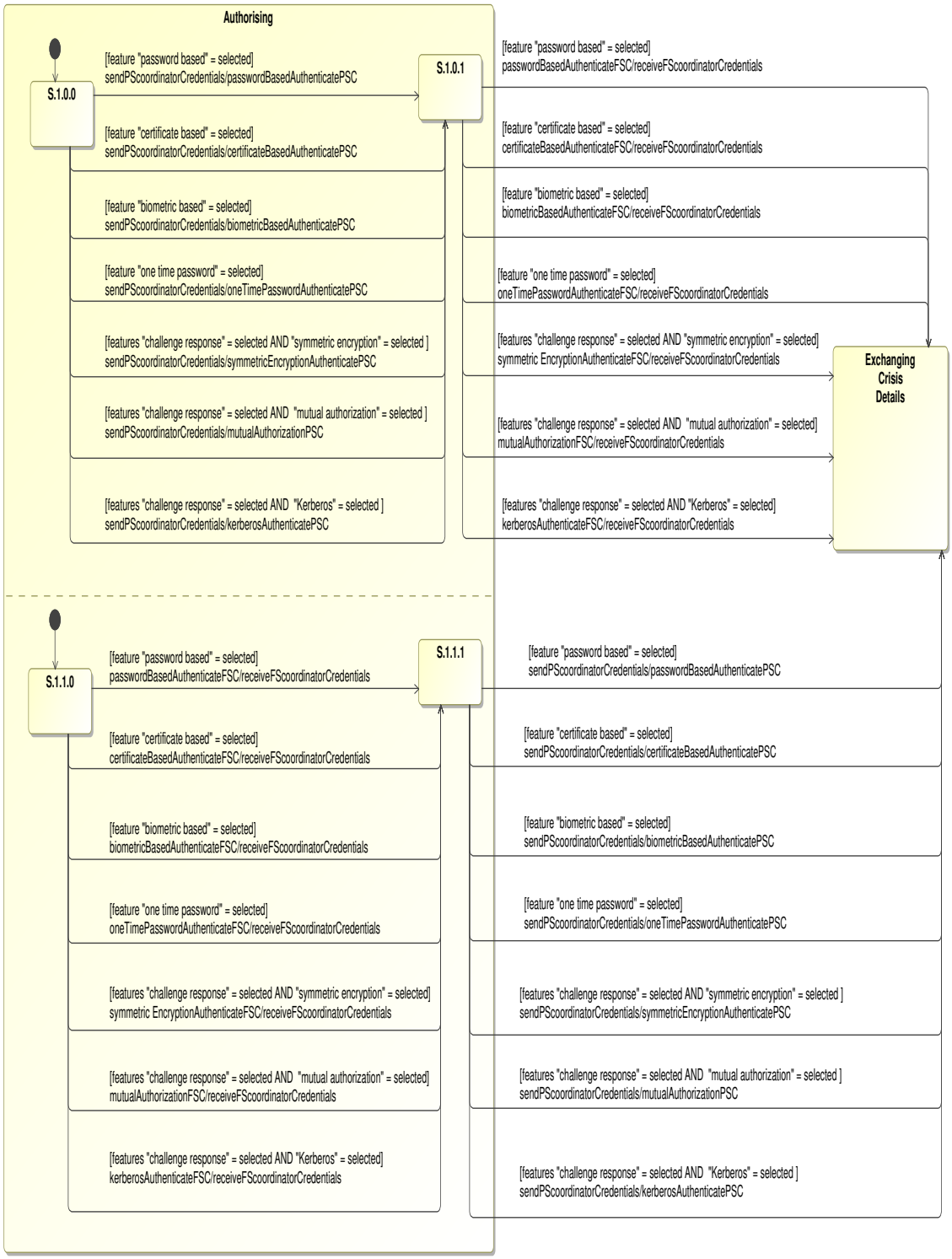


Figure 48: Authentication variation point - state machine for the PSC system.

The resulting sequence diagram is presented in Figure 47.

Modelling the state machine diagrams:

In the following, we briefly explain how the state machine diagrams are created for the *Authentication* variation point. They are created starting from the state machine diagrams previously done for the reference variant and the sequence diagrams done for this variation point, which were just presented. This variation point impacts the state machines of the PSC system and FSC system. The state machine for the PSC system of the reference variant is presented in Figure 15. The changes introduced by the *Authentication* variation point are at the level of the *Authorizing* and *Exchanging crisis details* states and the transition between them. We also analyse the sequence diagram for this variation point from Figure 47. In this diagram, several new messages, corresponding to the different possible authentication variants, are introduced in optional interaction fragments. New transitions, corresponding to these messages, need to be introduced therefore in the state machine for this variation point. The resulting state machine for the PSC system is presented in Figure 48. Following a similar procedure, we create also the state machine for the FSC system, which is shown in Figure 49.

Creating the architectural model:

Also for this variation point, the P2P architectural style is used. The architectural model corresponding to this variation point is presented in Figure 50.

3.6 Deriving a product of the bCMS SPL system

We conclude the section on SPL modelling of bCrash SPL by presenting how a specific individual product can be obtained from all the models created so far. The product we want to obtain should have the following features: describe the case of a single police station and fire station; have a vehicle management functionality that allows both police and fire stations to send messages to their respective vehicles; the communication between the police/fire stations and their respective vehicles will be done using a SOAP communication protocol; the system can handle a single crisis at a time; communication is done through a proprietary communication layer; system supports a certificate based authentication mechanism; data communication confidentiality is ensured by having encrypted communication. This derivation process corresponds to the Application Engineering phase of the SPL engineering process we follow throughout this paper.

The first activity that needs to be performed is to *configure the feature diagram*. This means that all the variation points defined in the initial feature diagram have to be resolved. The resolution of the variation points is done based on the specific choices made before:

- *Police and Fire station multiplicity*: variant *"One PS and FS"* is selected. The *XOR* feature relation automatically excludes the other possible variant *"Multiple PS and FS"*.
- *"Vehicles management"*: variant *"PSC and FSC send"* is selected.
- *"Vehicle management communication protocol"*: variant *"SOAP"* is selected.

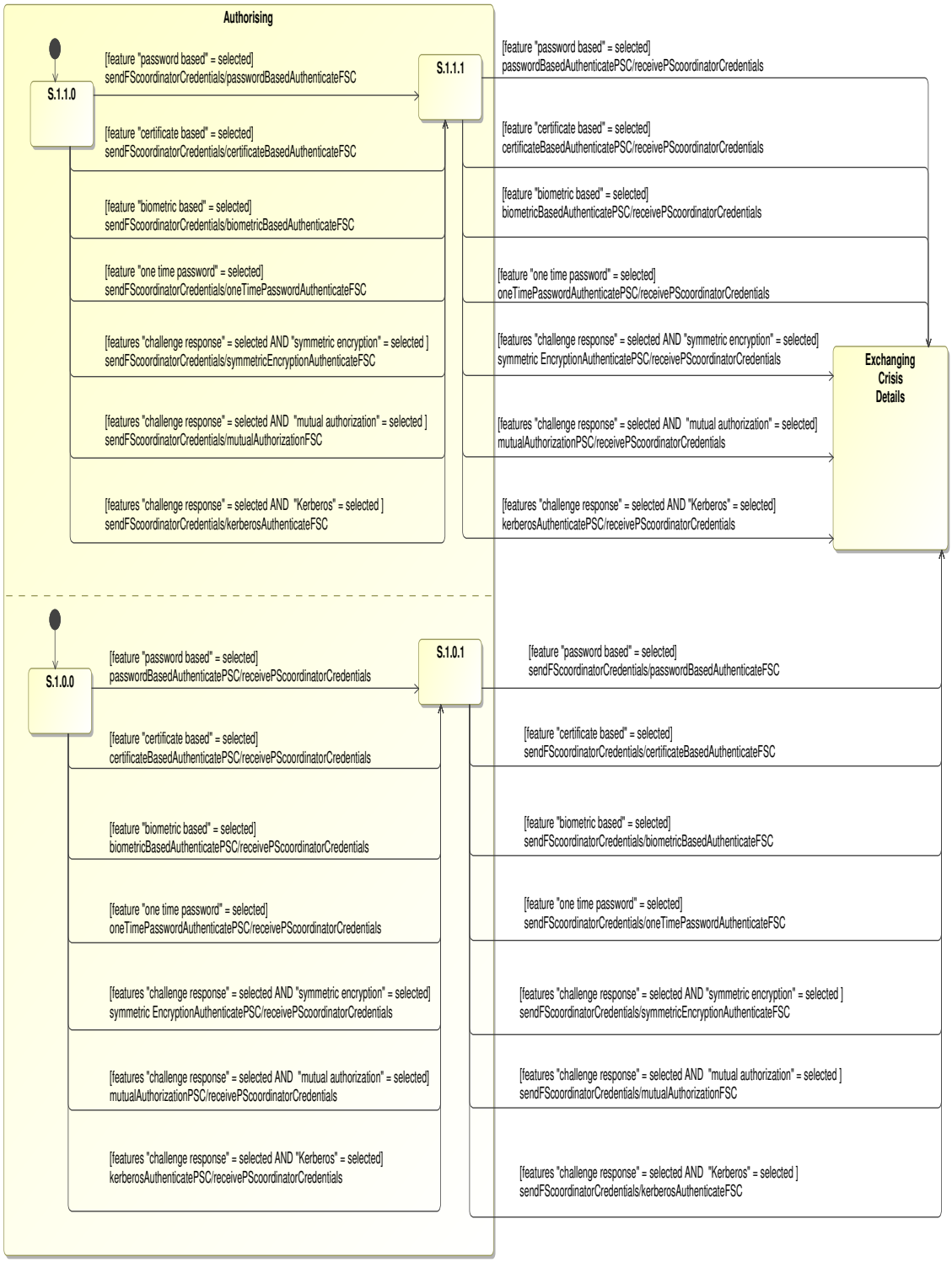


Figure 49: Authentication variation point - state machine for the FSC system.

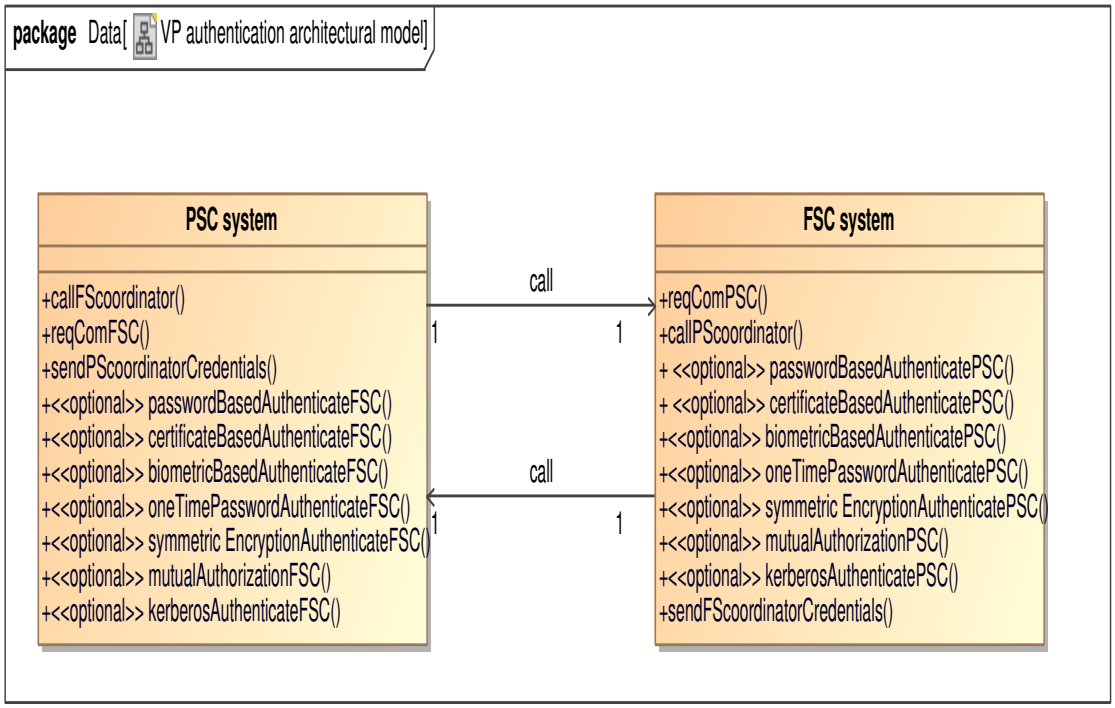


Figure 50: Authentication variation point - architectural model

- *"Crisis multiplicity"*: variant *"Single"* is selected. The *XOR* feature relation automatically excludes the other possible variant *"Multiple"*.
- *"Communication layer"*: variant *"Proprietary"* is selected. The *XOR* feature relation automatically excludes the other possible variant *"Other"* and implicitly all of its children.
- *"Authentication"*: variant *"Certificate based"* is selected.
- *Data communication confidentiality*: variant *"Encrypted"* is selected. The *XOR* feature relation automatically excludes the other possible variant *"Not encrypted"*.

These are all the variation points present in the feature diagram. All the other features that remain are *mandatory features*, so they need tobe selected for the product that we are deriving. The result of the feature diagram configuration process is presented in Figure 51.

The next step to be performed in the derivation process is to resolve the class diagram models for each variation point. This is done based on the choices previously made during the feature model configuration step. Based on those choices, several adaptations will be made to class diagram models of each variation point:

- *Police and Fire station multiplicity*: even though this variation point was not modelled, the selection of the variant *"One PS and FS"* corresponds to the reference

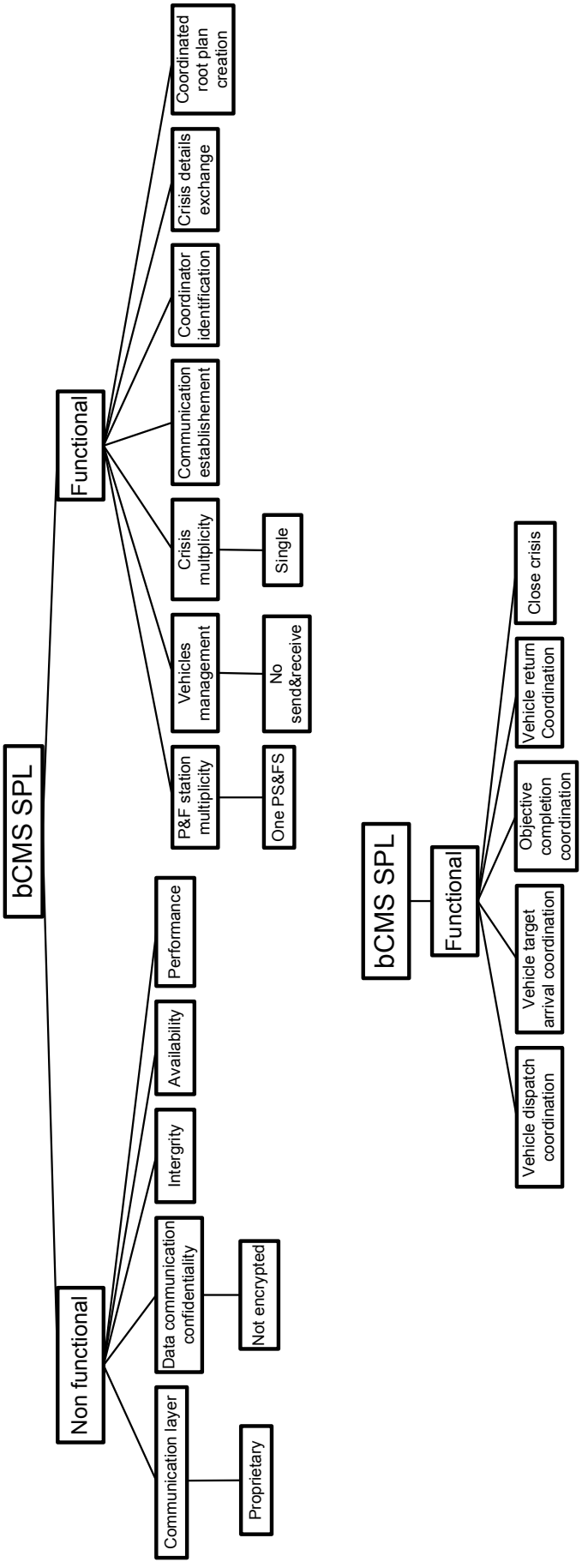


Figure 51: Deriving a specific product - configured feature diagram.

variant initially modelled using object-oriented approaches. Therefore, this feature will not add anything new to the original class diagram model of the reference variant.

- *"Vehicles management"*: variant *"PSC and FSC send"* was selected. The class diagram modelled for this variation point is changed in the following way: we analyse the sequence diagrams corresponding to this variation point and for all of them we extract the message exchanges that correspond to the selection of the *"PSC and FSC send"* variant; these message exchanges correspond to the methods that need to be kept in the configured class diagram for this variation point. The result of this process is the class diagram presented in Figure 52.

- *"Crisis multiplicity"*: variant *"Single"* was selected. The configured class diagram for this variation point does not bring anything new to the class diagram of the reference variant.

- *"Vehicle management communication protocol"*: this optional feature was selected and variant *"SOAP"* was selected for it. Several changes are necessary to configure the class diagram for this variation point. For classes *PS coordinator* and *FS coordinator*, all the methods tagged *optional* become normal methods. For classes *PSC system*, *FSC system*, *Dispatch service*, *Police car*, *Fire truck*, *Citizen vehicle*, we make the following changes: all methods tagged *optional* are kept and become normal methods; the classes are not abstract any more and their *abstract* methods are replaced withe the concrete SOAP version of of each method. The result is the class diagram from Figure 53.

- *"Communication layer"*: even though this variation point was not modelled, the choice made, variant *"Proprietary"*, corresponds to the reference variant modelled initially using object-oriented approaches. Therefore, the models for this feature do no bring anything new to the original OO models of the reference variant.

- *"Authentication"*: variant *"Certificate based"* was selected for this variation point. To configure the class diagram of this variation point, the following changes need to be made: in the *PSC system* and *FSC system* classes, from all the methods tagged *optional*, only keep the one corresponding to the certificate based authentication, which will also become a normal method. The resulting class diagram is presented in Figure 54.

  Data communication confidentiality: variant *"Encrypted"* is selected. We can notice that in the class diagram for this variation point presented in Figure 44, all the classes, except *bCrash System*, are *abstract classes*. When we configure this class diagram, the abstract classes will become concrete one, so their abstract methods are replaced by concrete methods corresponding to the version with *encryption*. The result of this process is the class diagram presented in Figure 55.

To obtain the final class diagram describing the particular product that we are deriving, we need to compose all the configured class diagrams obtained above with the

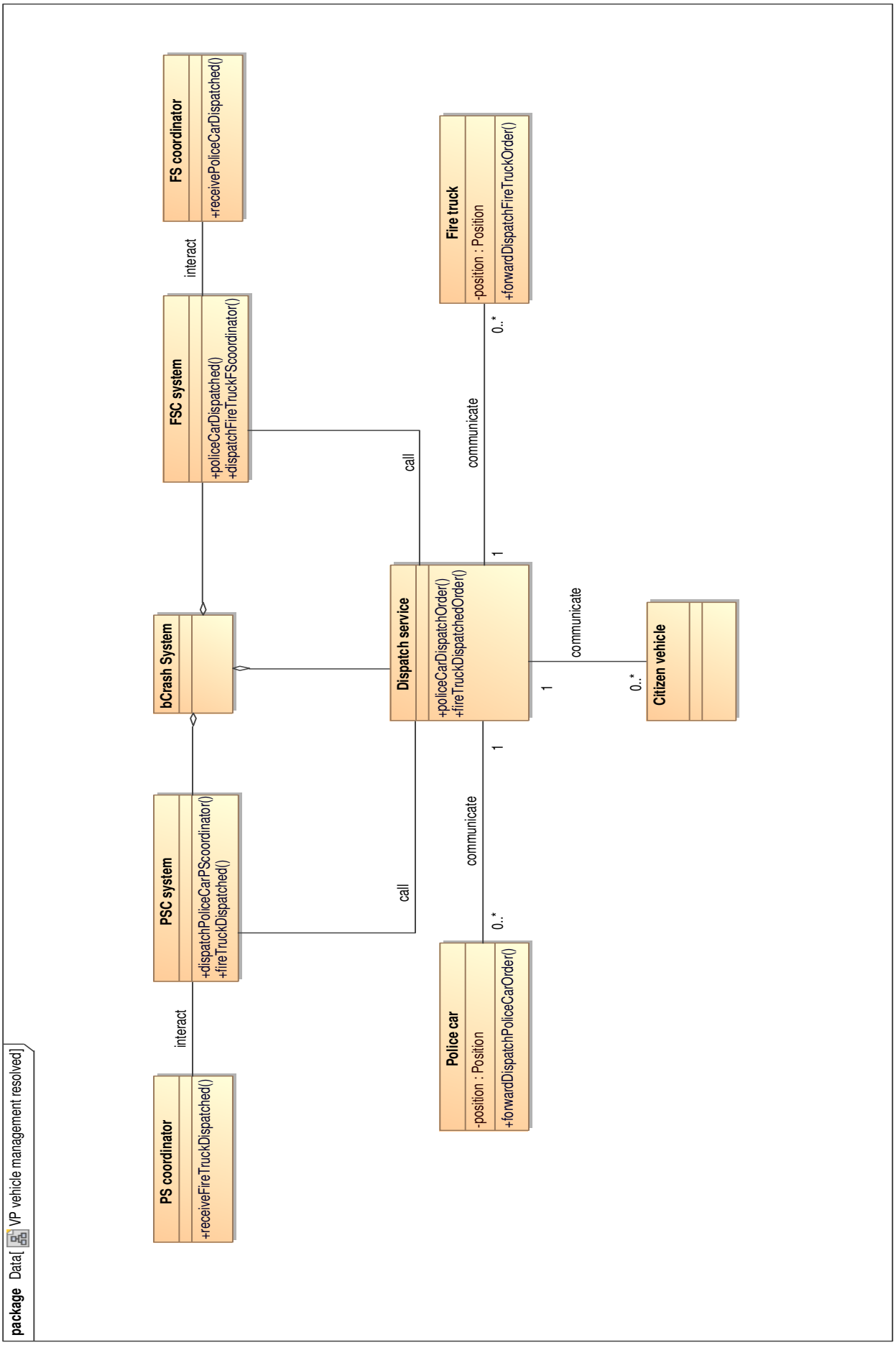


Figure 52: Variation point Vehicle Management - resolved

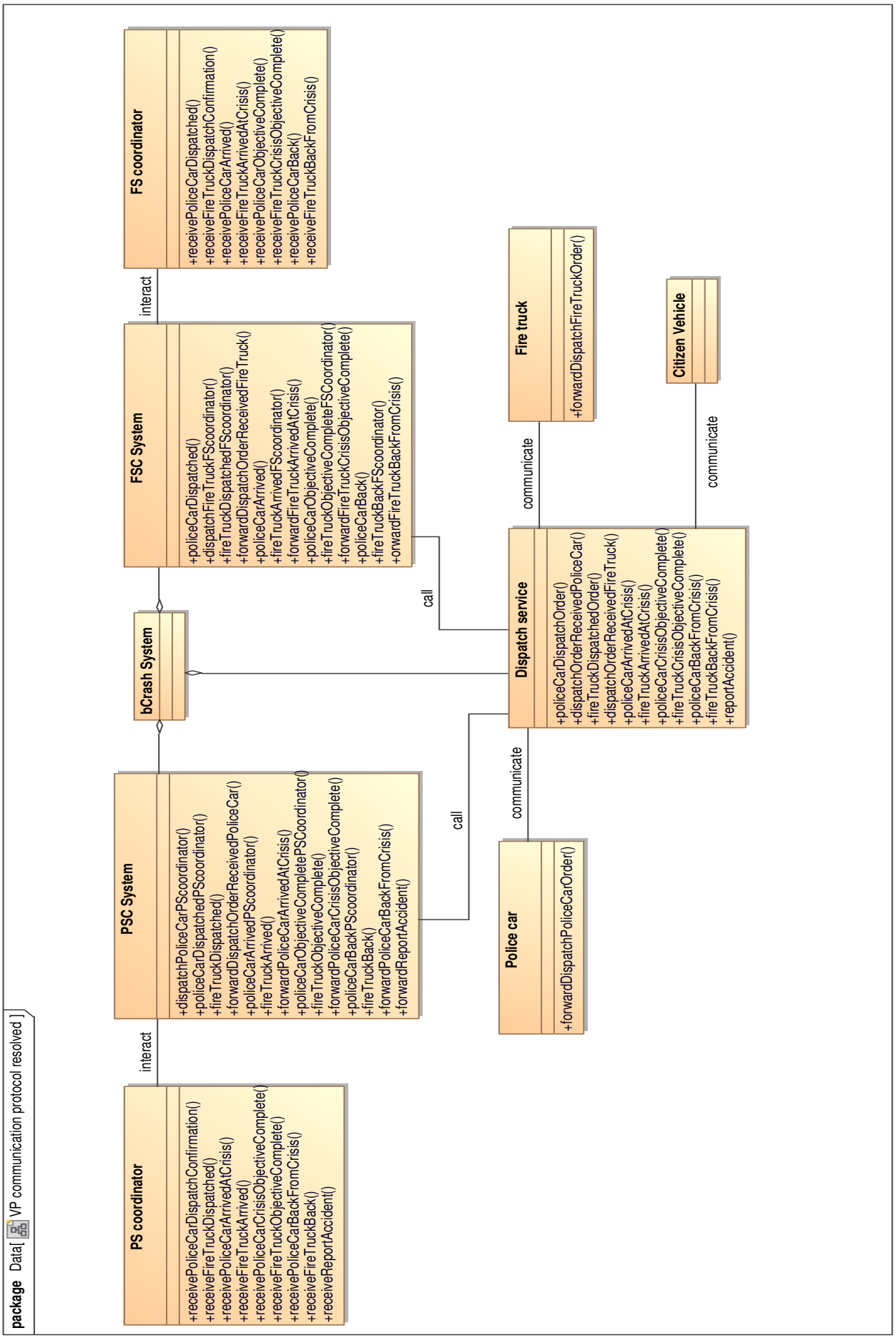


Figure 53: Variation point Communication Protocol - resolved

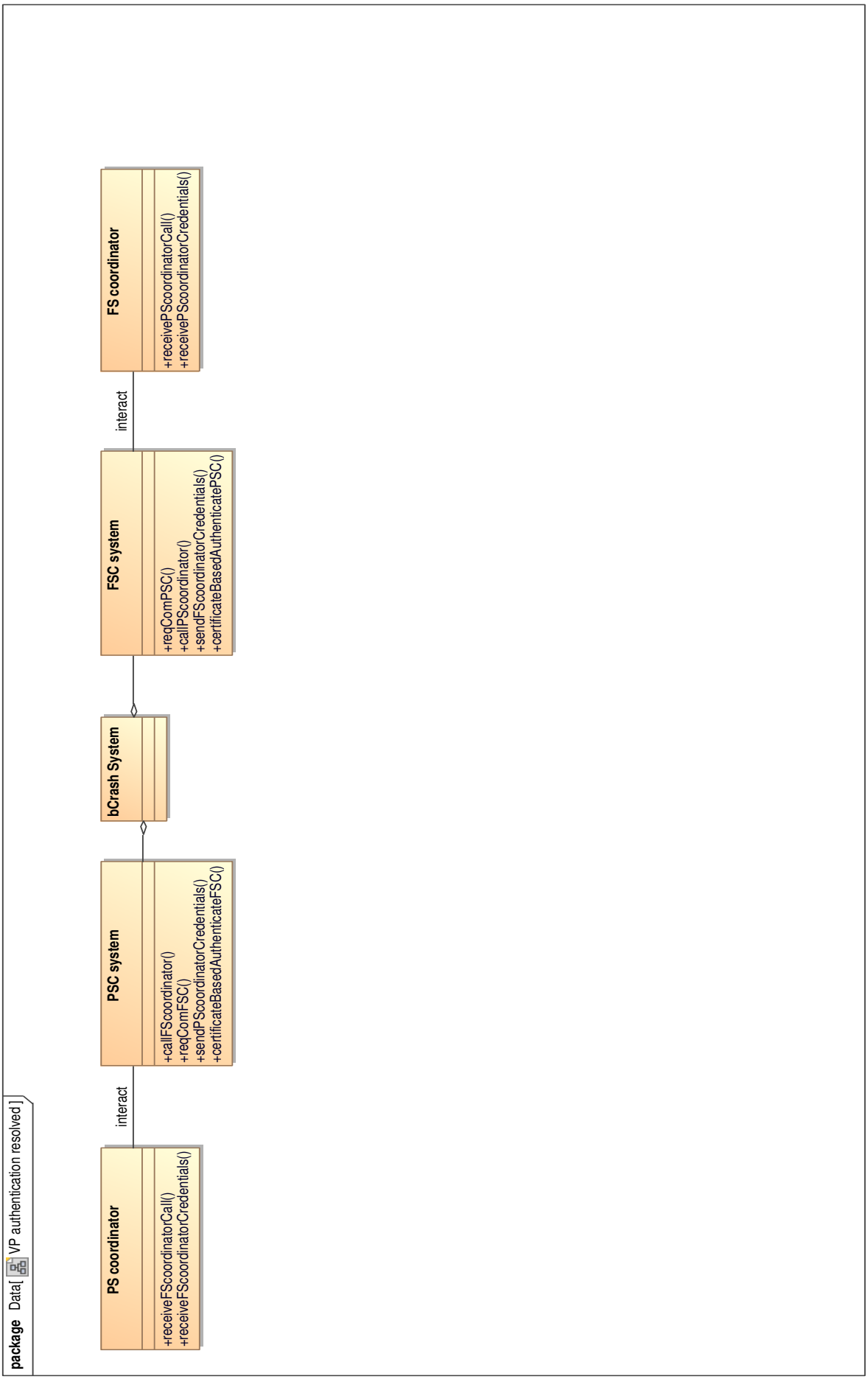


Figure 54: Variation point Authentication - resolved

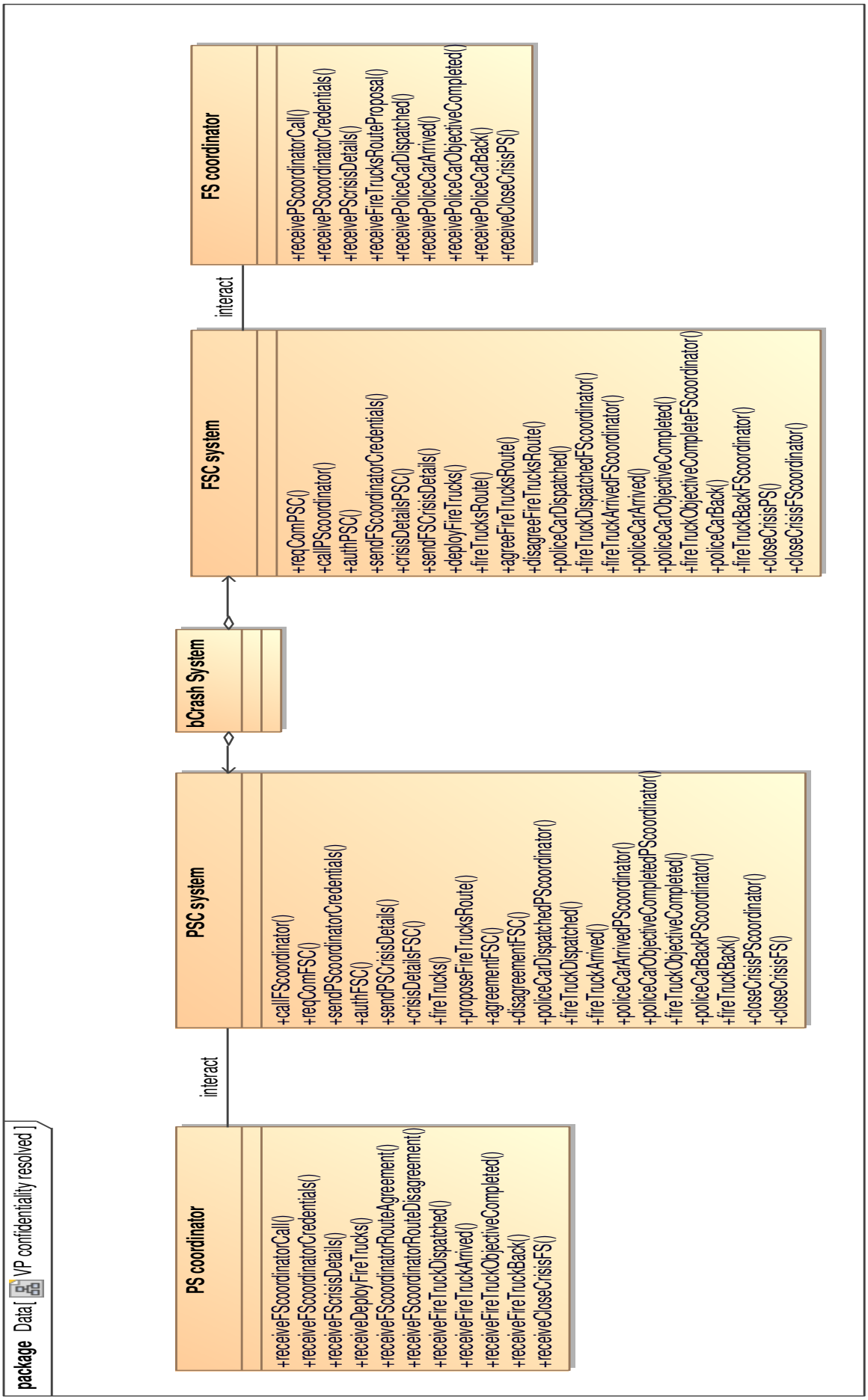


Figure 55: Variation point Data communication Confidentiality - resolved

original class diagram of the reference variant. The type of composition used is a UML class diagram merge. The result of the composition process is presented in Figure 56.

References

[1] Software product line conference - hall of fame. `http://splc.net/fame.html`.

[2] Bachmann, F., and Bass, L. Managing variability in software architectures. *SIGSOFT Softw. Eng. Notes 26*, 3 (2001), 126–132.

[3] Bachmann, F., and Clements, P. Variability in software product lines. Technical report cmu/sei-2005-tr-012, Software Engineering Institute, Pittsburgh, USA, 2005.

[4] Bosch, J. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[5] Capozucca, A., Cheng, B. H., Georg, G., Guelfi, N., Istoan, P., and Mussbacher, G. Requirements definition document of focused case study. http://cserg0.site.uottawa.ca/cma2011/CaseStudy.pdf, June 2011.

[6] Coplien, J., Hoffman, D., and Weiss, D. Commonality and variability in software engineering. *IEEE Software 15*, 6 (1998), 37–45.

[7] Harel, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program. 8*, 3 (1987), 231–274.

[8] Harel, D., and Naamad, A. The statemate semantics of statecharts. *ACM Trans. Softw. Eng. Methodol. 5*, 4 (1996), 293–333.

[9] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon University Software Engineering Institute, November 1990.

[10] Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng. 5* (1998), 143–168.

[11] Maccari, A., and Heie, A. Managing infinite variability in mobile terminal software: Research articles. *Softw. Pract. Exper. 35*, 6 (2005), 513–537.

[12] Northrop, L. A framework for software product line practice. In *Proceedings of the Workshop on Object-Oriented Technology* (1999), Springer-Verlag London, UK, pp. 365–376.

[13] Northrop, L. M. Sei's software product line tenets. *IEEE Softw. 19*, 4 (2002), 32–40.

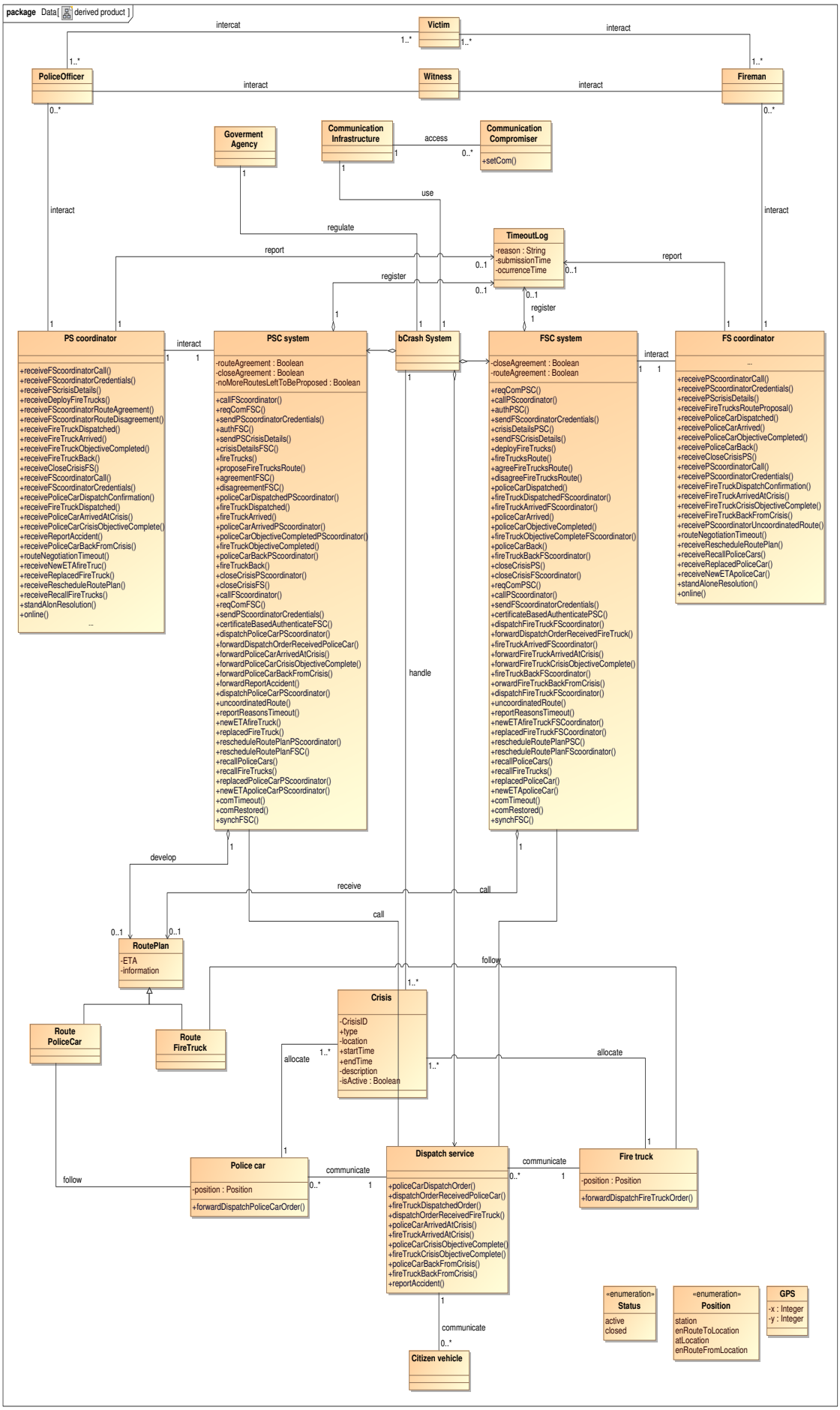


Figure 56: Class diagram of derived product

[14] Perrouin, G. *Architecting Software Systems using Model Transformation and Architectural Frameworks*. PhD thesis, University of Luxembourg (LASSY) / University of Namur (PReCISE), September 2007.

[15] Pohl, K., Böckle, G., and Linden, F. J. v. d. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[16] Warmer, J., and Kleppe, A. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[17] Weiss, D. M., and Lai, C. T. R. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[18] Ziadi, T., and Jezequel, J.-M. Software product line engineering with the uml: Deriving products. In *Software Product Lines*. 2006, pp. 557–588.

Appendix: Data Dictionary for OOM Domain Model

Element Name	Description
bCMS	Overall system for managing a crisis involving a Fire State Coordinator (FS coordinator) and a Police Station Coordinator (PS coordinator) via their respective software interfaces (FSC System and PSC System).
Relationships	This element aggregates into the PSC System and FSC System to "handle" a Crisis. It is regulated by a Government Agency and uses the Communication Infrastructure.
UML Extensions	

Element Name	Description	
FS coordinator	Human being in charge of coordinating the activities on the fire station side, and communicating the decisions made to the PS coordinator.	
Attributes		
	-credential:Credential	FS coordinator's credential to be used when identifying with the PS coordinator
Operations		
	+receivePScoordinatorCall()	Call to request a communication with a PS coordinator.
	+receivePScoordinatorCredentials(Credential)	Call to pass the PS coordinator credentials.
	+receivePScrisisDetails(Crisis)	Call to give the crisis details as got by the PC coordinator.
	+receiveFireTrucksRoute Proposal(RoutePlan)	Call to pass the route plan for fire trucks.
	+receivePoliceCarDispatched()	Call to inform a police car has been dispatched to the crisis location.
	+receivePoliceCarArrived()	Call to inform a police car has arrived at the crisis location.
	+receivePoliceCarObjective Completed()	Call to inform a police car has completed its mission.
	+receivePoliceCarBack()	Call to inform a police car has returned to its station.
	+receiveCloseCrisisPS()	Call to request the closing of the mission.
	+receiveDeployPoliceCars(Integer)	Call to inform the number of police cars to be deployed.
	+receivePScoordinator UncoordinatedRoute()	Call to inform the route plan will not be coordinated.
	+routeNegotiationTimeout()	Call to inform the route plan negotiation has passed its deadline.
	+receiveRescheduleRoutePlan()	Call to create a new route plan as more vehicles are required due to the higher

Appendix: Data Dictionary for OOM Domain Model

		severity of the crisis.
	+receiveRecallPoliceCars(Integer)	Call to inform the recalling of certain number of police cars due to the lower severity of the crisis.
	+receiveReplacedPoliceCar()	Call to inform a police car has been replaced by another one.
	+receiveNewETApoliceCar()	Call to inform the ETA of a police car.
	+standAloneResolution()	Call to inform the PSC is in stand-alone mode.
	+online()	Call to inform the PSC is in collaborative mode.
Relationships	The FS coordinator interacts with the FSC system to deal with the crisis in a collaborative manner. When dealing with the crisis the FS coordinator interacts with Fireman to manage the FireTrucks efficiently. In case the negotiation with the PS coordinator takes longer than expected, the FS coordinator has to report the reasons why the negotiation exceeded the predefined timeout.	
UML Extensions		

Element Name	**Description**	
FSC System	Software system that manages the interactions between the FS coordinator and the PSC System (that manages communication from the PS coordinator)	
Attributes		
	-routeAgreement : Boolean	The PS coordinator and FS coordinator have reached agreement about the route plan
	-closeAgreement : Boolean	The PSC and FSC have reached agreement to close the crisis
Operations		
	+fireTrucksRoute(RoutePlan)	Call from PSC to inform the proposed route plan for fire trucks
	+policeCarDispatched()	Call from PSC to inform a police car was dispatched to the crisis location
	+policeCarArrived()	Call from PSC to inform a police car has arrived to the crisis location
	+fireTruckObjectiveComplete FScoordinator()	Call from FS coordinator to inform a fire truck has completed its mission
	+reqComPSC(Credential)	Call from PSC to establish communication
	+authPSC(Credential)	Call from PSC to authenticate the PS coordinator

Appendix: Data Dictionary for OOM Domain Model

	+callPScoordinator(Credential)	Call from FS coordinator to establish communication
	+sendFScoordinatorCredentials (Credential)	Call from FS coordinator to pass his credentials
	+sendFSCrisisDetails(Crisis)	Call from FS coordinator to inform the crisis details
	+crisisDetailsPSC(Crisis)	Call from PSC System to inform the crisis details
	+deployFireTrucks()	Call from FS coordinator to deploy fire trucks.
	+agreeFireTrucksRoute()	Call from FS coordinator to agree on route for fire trucks
	+disagreeFireTrucksRoute()	Call from FS Coordinator to disagree on route plan for fire trucks
	+fireTruckDispatched FScoordinator()	Call from FS coordinator to inform a fire truck has been dispatched to the crisis location.
	+fireTruckArrived FScoordinator()	Call from FS coordinator to inform a fire truck has arrived at the crisis location.
	+policeCarObjectiveCompleted()	Call from PSC to inform a police car completed its mission.
	+fireTruckBackFScoordinator()	Call from FS coordinator to inform a fire truck has returned to the station.
	+policeCarBack()	Call from PSC to inform a police car has returned to the station.
	+closeCrisisFScoordinator()	Call from FS coordinator to request the closing of the crisis
	+closeCrisisPS()	Call from PSC to request the closing of the crisis
	+policeCars(Integer)	Call from PSC to inform the number of police cars to be deployed
	+uncoordinatedRoute()	Call from PSC to inform the route plan is not going to be coordinated
	+reportReasonsTimeout()	Call from FS coordinator to create description of reasons for Timeout
	+newETAfireTruck FSCoordinator()	Call from FS coordinator to inform the new ETA of a fire truck
	+rescheduleRoutePlanPSC()	Call from PSC to create a new route plan as more vehicles are required due to the higher severity of the

Appendix: Data Dictionary for OOM Domain Model

		crisis.
	+rescheduleRoutePlan FScoordinator()	Call from FS coordinator to create a new route plan as more vehicles are required due to the higher severity of the crisis.
	+recallPoliceCars(Integer)	Call from PSC to inform the recalling of certain number of police cars due to the lower severity of the crisis.
	+recallFireTrucks(Integer)	Call from PS coordinator to inform the recalling of certain number of fire trucks due to the lower severity of the crisis.
	+replacedPoliceCar()	Call from PSC to inform a police car has been replaced by another one.
	+newETApoliceCar()	Call from PSC to inform the ETA of a police car
	+comTimeout()	Call to inform that the communication between FSC and PSC is down.
	+comRestored()	Call to inform that the communication between FSC and PSC has been restored
	+synchPSC()	Call from PSC to inform its current state
Relationships	This element interacts with the FS coordinator, and it is expected to receive a RoutePlan. It also may register a TimeoutLog in case a time out happens while negotiating a route plan.	
UML Extensions	None	

Element Name Element	**Description**	
Crisis	The vehicular crisis to be handled by the collaboration of the Police and Fire department.	
Attributes		
	-id : Integer	Each crisis has a unique integer identifier.
	-location : GPS	Each crisis has a satellite-based location
	-startTime: TimeExpression	A 24-hour clock time is used to denote the initiation of a crisis (when the crisis is reported to the bCMS system)
	-endTime : TimeExpression	A 24-hour clock time is used to denote when the crisis considered closed by both FSC and PSC coordinators
	-status : Enum (“Active”, “Closed”)	Enumerated type for status of crisis (active or closed)
	-description : String	Text-based description of crisis, including nature of crisis (single car accident, multiple

Appendix: Data Dictionary for OOM Domain Model

		car accident, fire hazard, number of victims involved, etc.)
Relationships		
UML Extensions	This is a variation point in the product line, with 2 different variations.	

Element Name	**Description**	
Status	Enumerated data type	
Attributes		
	active:String	The crisis is still ongoing
	closed: String	The vehicle has been already solved
Relationships	Enumerated data type is used in Crisis class to define its current status.	
UML Extensions	NA	

Element Name	**Description**	
RoutePlan		
Attributes		
	-crisisID : Integer	Each crisis has a unique integer identifier.
	-information : String	Text-based description of crisis, including nature of crisis (single car accident, multiple car accident, fire hazard, number of victims involved, etc.)
Relationships	Comprises RoutePoliceCar and RouteFireTruck for routes for police and fire trucks, respectively. In addition, it is an aggregate of the PSC System that is responsible for developing the routes. Similarly, the route is sent to the FSC System.	
UML Extensions		

Element Name	**Description**	
TimeoutLog		
Attributes		
	-crisisID : Integer	Each crisis has a unique integer identifier.
	-submissionTime: TimeExpression	A 24-hour clock time is used to denote when the communication was initiated (between the coordinators)
	-ocurrenceTime : TimeExpression	24 hour clock denotes when the communication failed (i.e., timeout occurred)
	-reason : String	Text-based description for reason of timeout (e.g., problem on one or both coordinator sites, or faulty communication infrastructure)
Relationships	Element is reported by FS Coordinator and/or PS Coordinator	
UML Extensions		

Element	**Description**

Appendix: Data Dictionary for OOM Domain Model

Name		
Route	Route for crisis management vehicles.	
Attributes		
	-ETA: TimeExpression	24 hour clock denotes the estimated amount of time it will take for emergency vehicle to reach crisis site.
	-path : String	A string of directions starting with origination and ending at crisis site, where each direction comprises a heading (e.g., north, south, east, west, etc.), street name, distance to travel on that street).
Relationships	Class is subclassed to routes for Fire Truck and Police Car, respectively	
UML Extensions	NA	

Element Name	Description
RoutePoliceCar	Route for a police car
Relationships	Each instance of this class is associated with the PoliceCar vehicle expected to follow such a route.
UML Extensions	It extends from Route class

Element Name	Description
RouteFireTruck	Route for a fire truck
Relationships	Each instance of this class is associated with the FireTruck vehicle expected to follow such a route.
UML Extensions	It extends from Route class

Element Name	Description	
Position	Enumerated data type	
Attributes		
	station:String	The vehicle is at its station
	inRouteToLocation: String	The vehicle is going to the crisis location
	atLocation: String	The vehicle is at the crisis place
	inRouteFromLocation: String	The vehicle is returning to its central station.
Relationships	Enumerated data type is used in PoliceCar and FireTruck classes.	
UML Extensions	NA	

Element Name	Description	
FireTruck	Fire truck requested to solve the vehicular crisis	
Attributes		
	-position: Position	Current position of the fire truck
Relationships	Each fire truck is allocate to the current ongoing crisis. The use of the fire truck is managed by the FS coordinator.	

Appendix: Data Dictionary for OOM Domain Model

UML Extensions	NA

Element Name	**Description**	
PoliceCar	Police car requested to solve the vehicular crisis	
Attributes		
	-position: Position	Current position of the police car
Relationships	Each police car is allocate to the current ongoing crisis. The use of the police car is managed by the PS coordinator.	
UML Extensions	NA	

Element Name	**Description**	
GPS	Satellite-based location	
Attributes		
	-x:Float	Latitude coordinate
	-y:Float	Longitude coordinate
Relationships	Used in the Crisis class to indicate the location of the crisis	
UML Extensions	NA	

Element Name	**Description**
Fireman	Firemen involved in the solving of the vehicular crisis
Relationships	A firemen interacts with the FS coordinator that gives him orders, and with the Victims of the crisis. He also might interact with some of crisis' Witnesses
UML Extensions	NA

Element Name	**Description**
PoliceOfficer	Police officer involved in the solving of the vehicular crisis
Relationships	A police officer interacts with the PS coordinator that gives him orders, and with the Victims of the crisis. He also might interact with some of crisis' Witnesses
UML Extensions	NA

Element Name	**Description**
Victim	Person(s) involved in crisis
Relationships	Provide information (interact) to the fire and police personnel
UML Extensions	NA

Element Name	**Description**
Witness	Person(s) who witnessed the crisis before, during, or after its occurrence
Relationships	Provide information (interact) to the fire and police personnel
UML Extensions	NA

Appendix: Data Dictionary for OOM Domain Model

Element Name	Description
Government Agency	Entity responsible for regulating the police and fire stations and protocols.
Relationships	Serves to regulate the policies realized by the bCMS system
UML Extensions	NA

Element Name	Description
Communication infrastructure	The means by which fire personnel correspond with the FS Coordinator, the police with the PS Coordinator, and how the FS and PS Coordinators communicate amongst themselves and with their respective software systems.
Relationships	The bCMS system uses the Communication Infrastructure to supports all of its communication needs.
UML Extensions	NA

Element Name	Description
Communication compromiser	This is an entity that might compromise the communication, including security threats, overloading of networking, etc.
Relationships	Potentially accesses Communication Infrastructure
UML Extensions	NA