OPCloud Version 6.1

Introduction Guide

March 2022

Table of Contents

1. Purpose of the Guide

The purpose of this Guide is to help you as an **Object Process Methodology OPM ISO 19450** modeler to create and evolve OPM models with **OPCloud 6.1.** The Guide focuses on OPM and the way it is implemented in OPCloud, with a basic example. It shows how to use OPCloud most common features while introducing some of the OPM methodology parts.

The software environment embodied in OPCloud supports OPM ISO 19450-based system development, evolution, lifecycle engineering, lifecycle management and knowledge management. This software package implements the bimodal expression of the OPM model along with many new model-based capabilities, including computations and automatic document generation. In addition to OPM-based systems modeling, the OPCloud platform will support such features as ,querying, argumentation, simulation and validation of the model, and design-structure matrix.

The Guide is divided into sections, each describing a certain aspect of OPM in OPCloud.

2. Brief Introduction to OPM

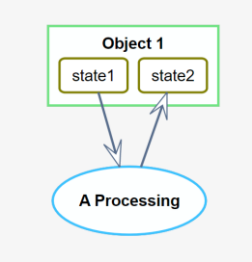


Figure 1. An Object-Process Diagram (OPD) showing the three OPM entities: Object, Process, and State, and the input/output procedural link pair, which expresses that Processing changes Object from state1 to state2.

Object Process Methodology (OPM) is a holistic approach for conceptual modeling of complex systems. The OPM model integrates the functional, structural, and behavioral aspects of a system in a single, unified view, expressed bi-modally in equivalent graphics and text with built-in refinement-abstraction mechanism. Allows a clear representation of the many important features of a system.

Two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related hierarchically organized Object-Process Diagrams (OPDs), showing portions of the system at various levels of detail, constitute the graphical, visual OPM formalism.

The OPM ontology comprises entities and links. Each OPM element (entity or link) is denoted in an OPD by a symbol, and the OPD syntax specifies correct and consistent ways by which entities can be connected via structural and procedural links, such that each legal entity-link-entity combination bears specific, unambiguous semantics (see Figure 1 for example).

There are three different types of entities: objects, processes (collectively referred to as "things"), and states (Figure 1). *Objects* are the (physical or informatical) things in the system that exist, and if they are stateful (i.e., have states); the *states* of an object are a specific situation of position, they are at some state or in transition between states. *Processes* are the things in the system that transform objects: they generate and consume objects or affect stateful objects by changing their state.

Links are graphical expressions of relations between things, in OPM, links connected processes with objects or their states, providing meaning to relationships among them. Link can be structural or procedural: Structural links express static, time-independent relations between pairs of entities. The four fundamental structural relations are aggregation-participation, generalization-

specialization, exhibition-characterization, and classification-instantiation. General tagged structural links provide for creating additional "user-defined" links with specified semantics. Procedural links connect processes with objects or object states to describe the behavior of a system.

System behavior is manifested in two ways: (1) a process can transform (generate, consume, or change the state of) one or more objects; (2) an object can enable one or more processes without being transformed by them, in which case it acts as an agent (if it is human) or an instrument (non-human). Accordingly, a procedural link can be a transformation link or an enabling link. A transformation link expresses object transformation, i.e., object consumption, generation, or state change. Figure 1 shows a transformation link, the input-output link pair. It expresses in OPL that **Processing** changes **Object** from **state1** to **state2**. An enabling (agent or instrument) link expresses the need for a (possibly state-specified) object to be present for the enabled process to occur. The enabled process does not transform the enabling object.

A procedural link from an object to a process can have a control modifier, which can be the letter e for event or c for condition next to the process end of the link. A procedural link with an event control modifier has the additional semantics that the object or state to which the process is connected invokes that process. A procedural link with a condition control modifier has the additional semantics that the process to which is the object or state is connected is skipped and not executed if the object does not exist or is not at that state.

The System Diagram, SD, is the topmost diagram in a model. It presents the most abstract view of the system, typically showing a single process as the main function of the system, along with the most significant objects that enable it and the ones that are transformed by it. The further from the root the OPD is, the more detailed it is. Each OPD, except for the SD, is obtained by refinement—in-zooming or unfolding—of a thing (object or process) in its ancestor OPD. This refined thing is described with additional details. The abstraction-refinement mechanism ensures that the context of a thing at any detail level is never lost, and the "big picture" is maintained at all times.

A new thing (object or process) can be presented in any OPD as a refinable (part, specialization, feature, or instance) of a thing at a higher abstraction level (a more abstract OPD). It is sufficient for some detail to appear once in some OPD for it to be true for the system in general even though it is not shown in any other OPD. Accordingly, copies of a thing can appear in other diagrams, where some or all the details (such as object states or thing relations) that are unimportant in the context of a particular diagram need not be shown.

The Object-Process Language (OPL) is the textual counterpart modality of the graphical OPD set. OPL is a dual-purpose language, oriented towards humans as well as machines. Catering to human needs, OPL is designed as a subset of English, which serves domain experts and system architects, jointly engaged in modeling a complex system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase that is generated automatically by OPCloud in response to the user's input. According to the modality principle of the cognitive theory of multimodal learning, this dual graphic/text representation of the OPM model increases the human processing capability. It has indeed been our experience that humans enhance their understanding of the model as they conveniently draw upon the graphic and/or the textual model representation to complement what they missed in the other modality.

A major problem with most graphic modeling approaches is their scalability: As the system complexity increases, the graphic model becomes cluttered with symbols and links that connect

them. The limited channel capacity is a cognitive principle which states that there is an upper limit on the amount of detail a human can process before being overwhelmed. This principle is addressed by OPM and implemented in OPCloud with three abstraction/refinement mechanisms. These enable complexity management by providing for the creation of a set of interrelated OPDs (along with their corresponding OPL paragraphs) that are limited in size, thereby avoiding information overload and enabling comfortable human cognitive processing.

The three refinement/abstraction mechanisms are: (1) unfolding/folding, which is used for refining/abstracting the structural hierarchy of a thing and is applied by default to objects; (2) in-zooming/out-zooming, which exposes/hides the inner details of a thing within its frame and is applied primarily to processes; and (3) state expressing/suppressing, which exposes/hides the states of an object.

2.1 OPM Basics: A Quick Summary

- OPM is a modeling language based on the paradigm that views processes and objects as equally important in the system model.
- OPM uses **Objects** and **Processes** in order to model the structural and behavioral aspects of a system.
- **Objects** are things that exist over time. Object can be **stateful** (i.e., have states).
- **Processes** are things that transform **Objects** by creating them, destroying them, or changing their states.
- **Procedural links** connect processes to objects to express these transformations.
- **Structural relations** connect objects to express static, long-term between them.

This Manual presents the basic concepts of OPM as both a language and a MSBE methodology, using OPCloud. For more details you can review the book Model-Based Systems Engineering with OPM and SysML, writing by Dov Dori, Professor of Technion, Israel Institute of Technology. And you also can search complementary information in ISO 19450:20015 specifications.

3. Starting to Model with OPCloud

The OnStar System, specified in ***Annex A*** in free text, is used as a running example throughout this guide. A detailed, step-by-step OPM-based model construction of the OnStar System is used to explain how to create an OPM model using OPCloud by describing almost each mouse-click, so you, as a new user, can closely follow each step and reproduce the model as it is being constructed in the pages below.

3.1 Starting to Work with OPCloud

Go to the following link: https://opcloud.systems/

NOTE: If you can experience and practice OPCloud, but do not have a user ID and password, you can use OPCloud Sandbox: https://sandbox.opm.technion.ac.il/ where credentials are not needed, so, but note that you will not be able to save your models.

3.2 Sign In

Logging in our system can be performed using 3 methods:

1. OPCloud Account – email and password
2. Microsoft Single Sign On (SSO) of supported organization
3. Google Single Sign On (SSO) of supported organization

Figure 2. OPCloud login page

Login using account details

Click the OPCloud Account button, as shown in Figure 2, and the sign in popup menu will be shown as seen in Figure 3.

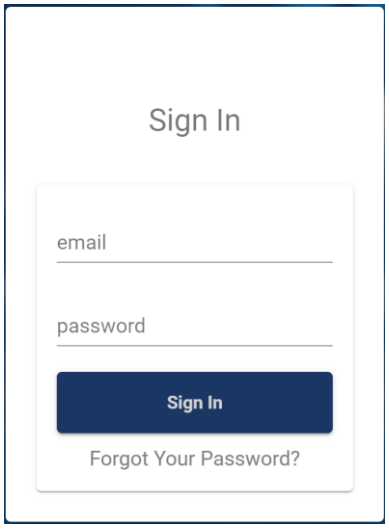


Figure 3. OPCloud sign-in via account details popup

Sign in using your email and password (login details). If you have forgotten your password, you can insert your account mail, and click on the "Forgot Your Password?" link at the bottom of the sign-in popup screen, and an email with instruction to reset your password will be sent. (Please note, that this is not optionable for SSO users, expaired users and unregistered users).

Login using SSO

First login: On the first login trial using the SSO you will be asked to enter your SSO's organizational credentials. if your organization is supported, the system will automatically forward you to the main system screen, directly to your organization. If your organization is not supported, a system admin will be informed about it automatically.
Regular login: After the first time you have logged in, if you are already logged in to your organization SSO, you will automatically be inserted into your account.

Multiple SSO accounts

If you have multiple SSO accounts with several organizations that has defined OPCloud organization, you may switch between the organizations.
First, if you are logged in and in OPCloud, logout from OPCloud by clicking on your name and sign out. Then, click on the Switch SSO Account button (Figure 2) and it will log you out from the current organization SSO. Please note that logging out from the organization SSO may transfer you to the organization SSO logout page.
Go back to OPCloud login page, and sign-in via the SSO button again. You will be promoted to select the desired SSO account to be logged in with. Once selected, and if needed logging in to the organization SSO, you will be automatically logged in into OPCloud

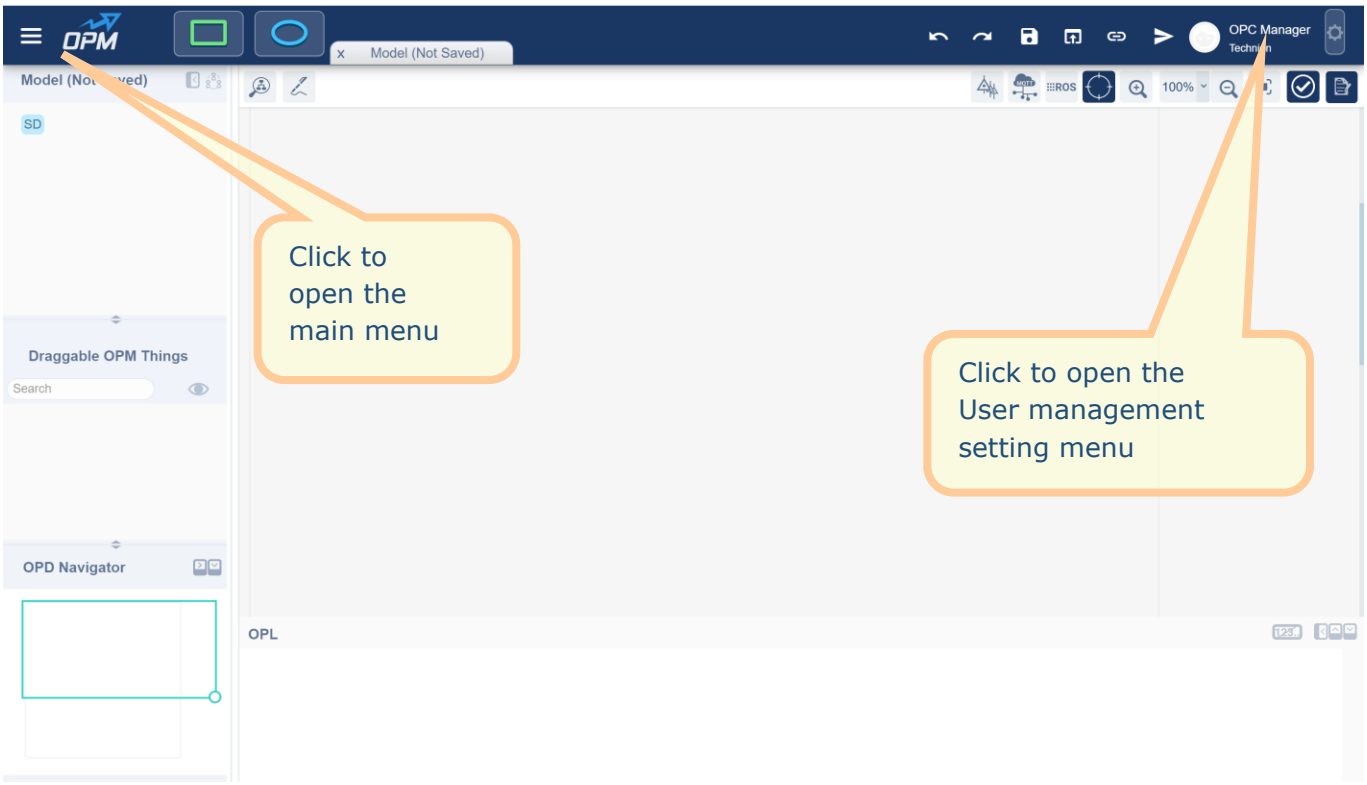


Figure 4. Main OPCloud screen after login (with your login name), user management setting menu button, and main menu.

3.3 Main OPCloud menu

Before we start to model the OnStar System, we need to know the essential parts of OPCloud. The first thing is the main menu, at the left side of the screen, specified by 3 horizontal lines (Figure 4) that clearly show to us labels like (Figure 5): New Model, you can create a model and save it for later; Load Model; Save and Save as (create a model name, give it a short description); Model Option which includes: System Map, Copy link – copy the model URL link, Model validation option for range validation policy, Compare model (see the difference between two models) and Mark Things (select model tings that will be colored in grey, also option to export the model tings list); Import Model, in this case, the input file must be .opx saved in OPCAT 4.2 (Figure 6); Exports, you have three way to export your model, these are: Export OPL (as HTML), Export Model OPD's (as JEPG) and Export Model to PDF; OPCloud Settings; finally, About, that show the OPCloud version and Help, a link to online manual in the setting screen.

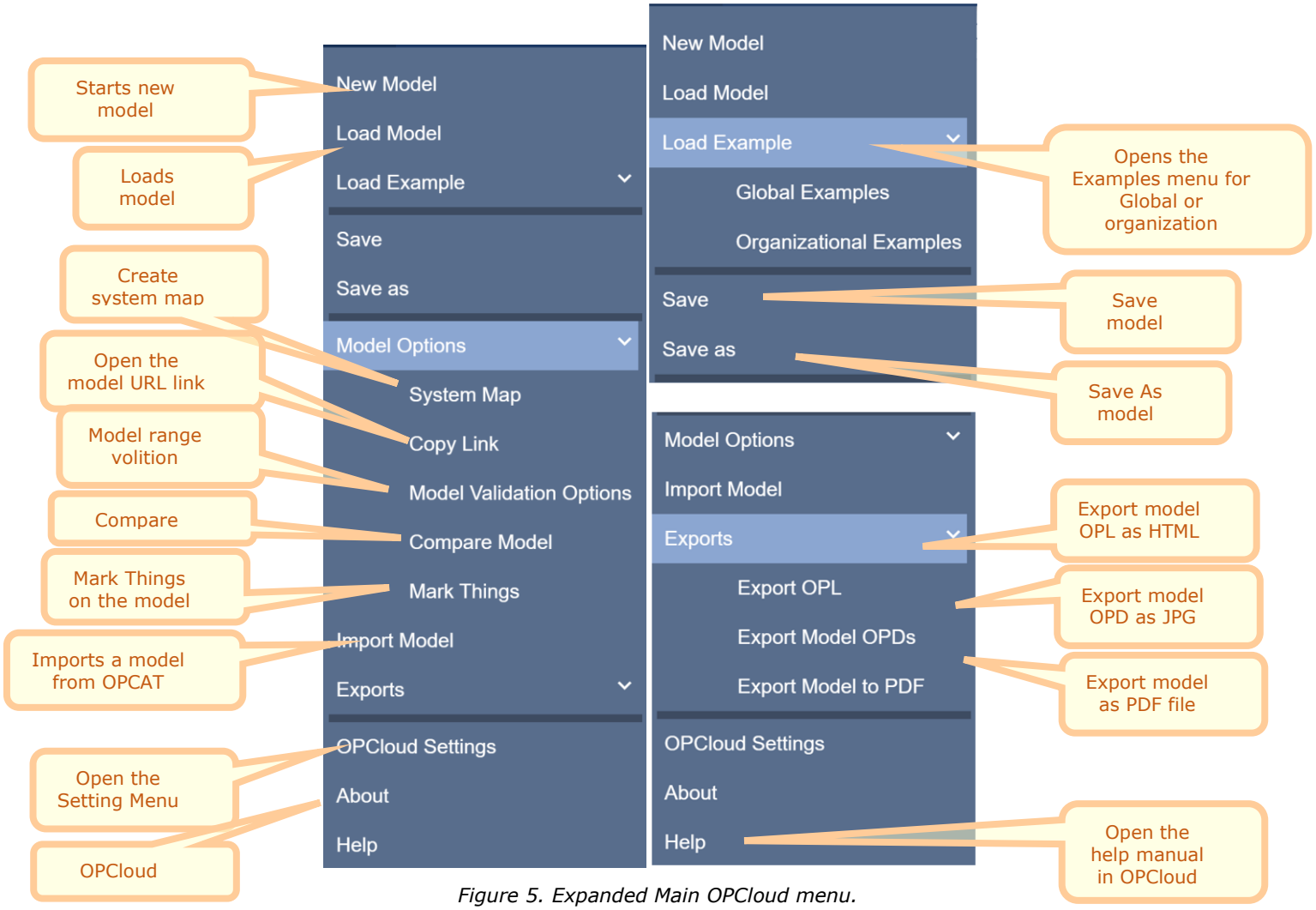


Figure 5. Expanded Main OPCloud menu.

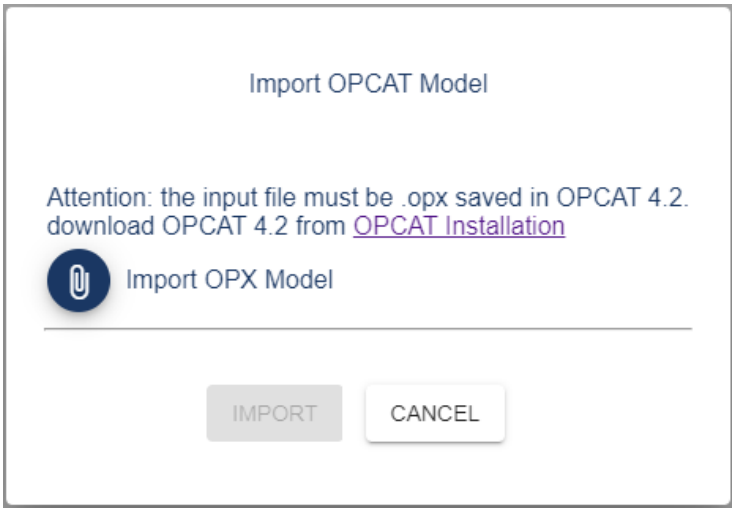


Figure 6. Import model in main OPCloud menu.

In Figure 7 we can see other buttons in the main toolbar in OPCloud, located in the upper right, from left to right are: Undo, Redo, Save, Load, Copy Link URL for current model, and Execute (Simulation).

Figure 7. OPCloud main toolbar actions.

For more details on each option, go to the second manual "OPCloud User Guide".

3.4 The System Diagram (SD)

The ***System Diagram*** (SD) is the first, top-level Object-Process Diagram (OPD) of our system, where we start any new OPM model in the opening main screen. SD expresses clearly the main system's function – the combination of the main system's process and object, typically specifying the function – what the modeled system is designed to do, the beneficiary, the operands – the objects that are transformed (consumed, created, and/or affected) by the main process, the main human and non-human enablers (agents and instruments) of the system, and the environmental things (objects and processes) – the things which interact with the things in our system – the systemic things.

3.5 Defining the system's main process

We now start to model the OnStar system. We do this by drawing model elements – processes, objects, and links – on the canvas—the main large pane. As we model, we inspect in real time the translation of our graphic editing into OPL – Object-Process Language – to make sure we have done the right thing.

The first thing we need to do when modeling a new system is to **define the system's major function**. This is going to be the pair of things: a main process and the main object which that process transforms.

To create a process (denoted as a blue ellipse), you drag it from the main toolbar (see Figure 8). Click on the Process icon (the blue ellipse on the left of the toolbar) holds the mouse button and drag it to the desired location, the approximate center of the canvas. You will get a process as shown in Figure 8. The default name of the first process is 'A Processing' (Figure 8. the letters will increase as we add processes until Z and then AA, AB …). The default process name is highlighted in the opened edit text popup menu, so to change its name just start typing. To update the process name at any time, double click on its name so the edit text popup menu opens, and the current value becomes highlighted, and type the name you want it to have, in our case, "Driver Rescuing". You should get the process as shown in Figure 9 and Figure 10.

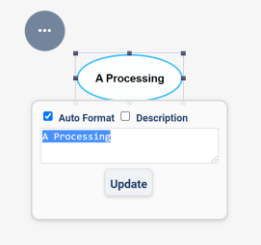


Figure 8. Creating a process

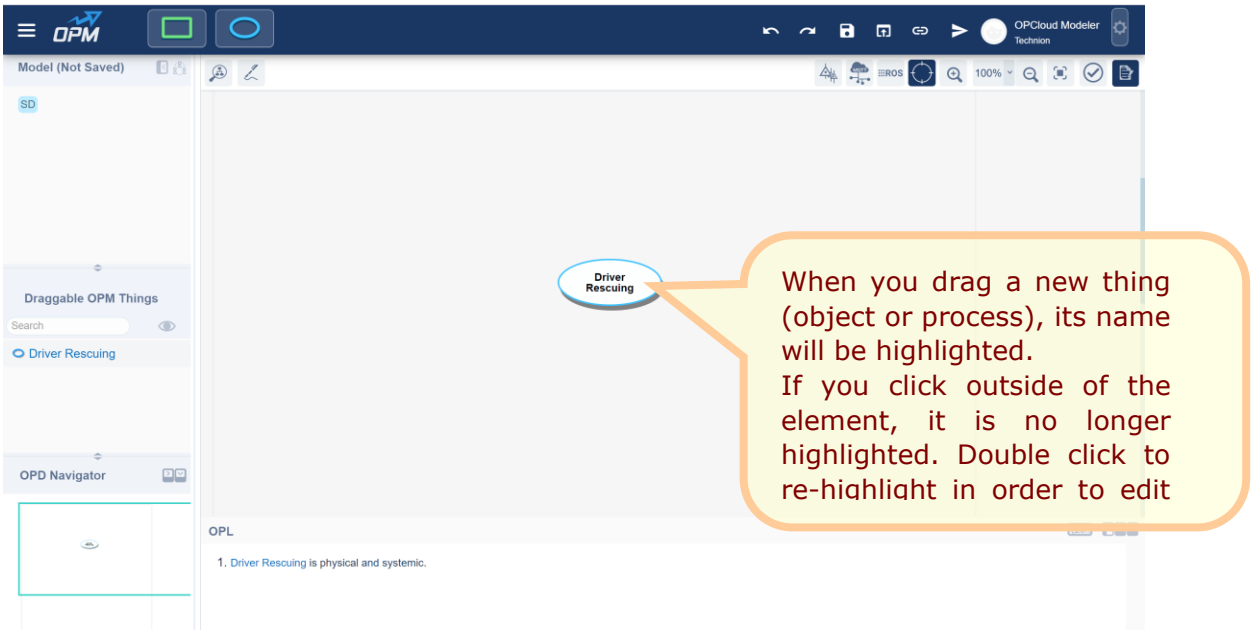


Figure 9. Updating a process name.

Figure 10. Process name updated.

3.6 Saving the model

Before we move on, we should save our model. If the model is not saved yet, the Save As popup screen will be opened. In the screen we name our model, with description, browse in the various folders, switch views, see previous model version and archived models and more. For more details on each option, go to the second manual "OPCloud User Guide". In our example we selected "OnStar System" as the model's name.

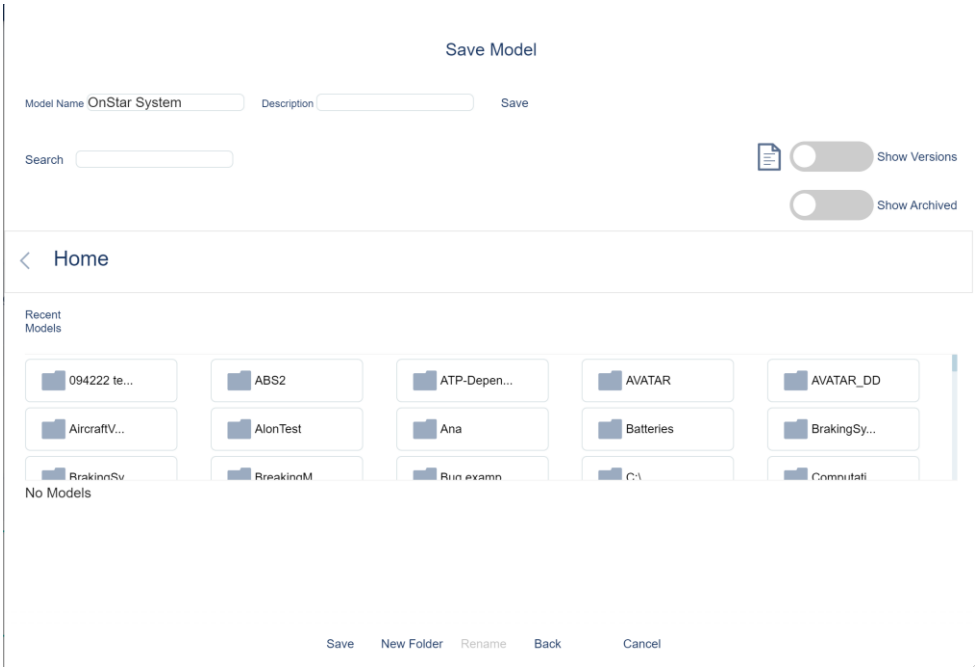


Figure 11. The Save As popup menu.

3.7 Creating objects and linking them

Having defined the major system process, we now turn to depicting the main objects in the system. We need to think first who the beneficiary of the system is, i.e., who gets value from the system. Obviously, it is the driver, so we insert **Driver** as a physical object. To insert a new object, as we did for process, drag the Object icon (the green rectangle on the left of the toolbar) to the desire location on the canvas. You will get an object with the default name **Object 1** as shown in Figure 12. For more details on each option, go to the second manual "OPCloud User Guide".

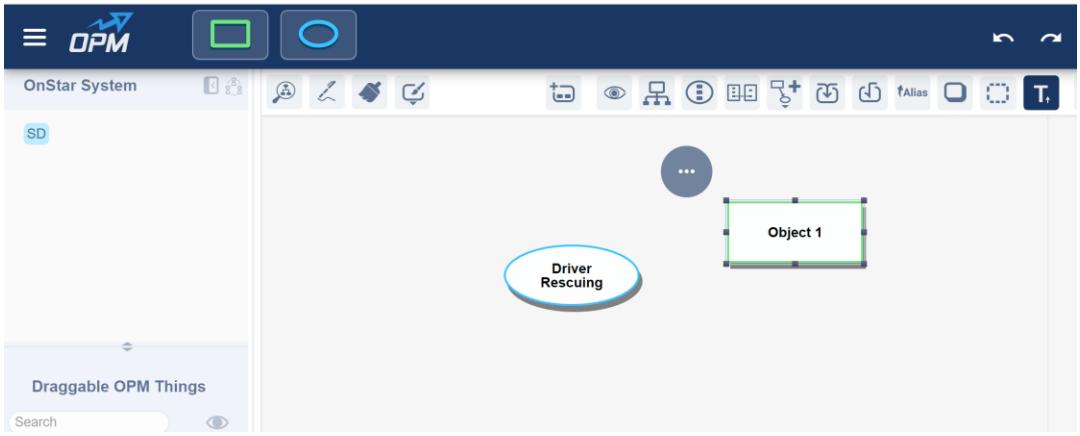


Figure 12. Object 1 is added to the canvas by dragging it from the green object icon on top left.

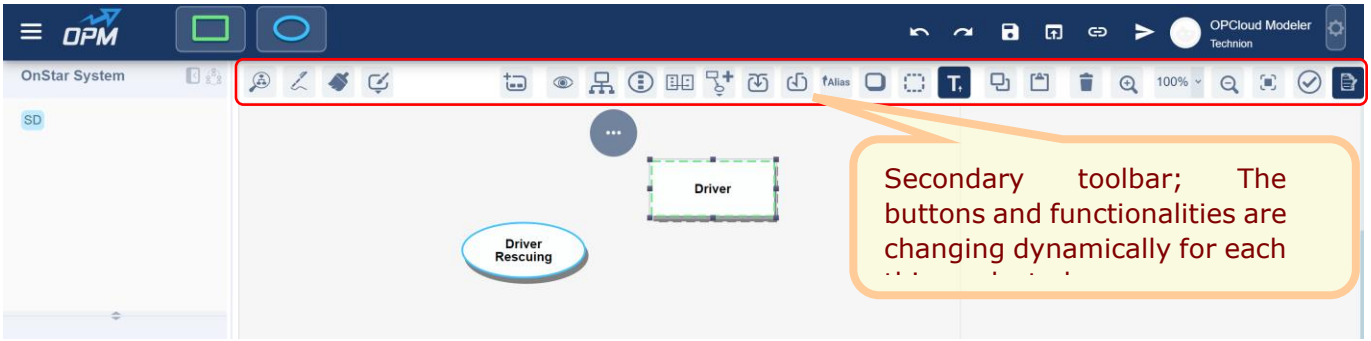


Figure 13. Object 1 is changed to Driver and made environmental.

The default name of an object in OPCloud is "**Object**" +<**running number**>. The object name is highlighted, so to update it, just type the new name, **Driver**. If you clicked outside, the name is no longer selected, so double click on the name. Please note that each new thing in OPCloud is toggled with the Auto Format name option. This option will automatically capitalize each word first letter and will lower case the rest of the word. In order to use capital letters in other places in the word, you will need to remove the selected option checkbox (Figure 14).

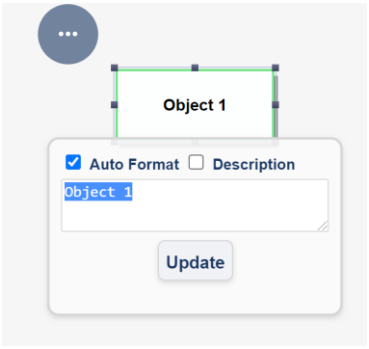


Figure 14. Object 1 name update and name Auto Format.

Since the driver is not part of the system, but interacts with it, its **Affiliation** is **environmental** rather than **systemic.** Changing the object affiliation is done via the Expansion menu, which shows up for a thing when it is selected (see Figure 13). Once we do this, the OPL sentences in the OPL pane are updated.

Another vital object in the model is the **OnStar System**, which we also add as a physical object. Figure 15 shows **SD**, the **System Diagram**, after the objects **Driver** and **OnStar System** were added and moved around. To move a thing, place the mouse or trackpad over its name so that the cursor changes to compass, and then drag it.

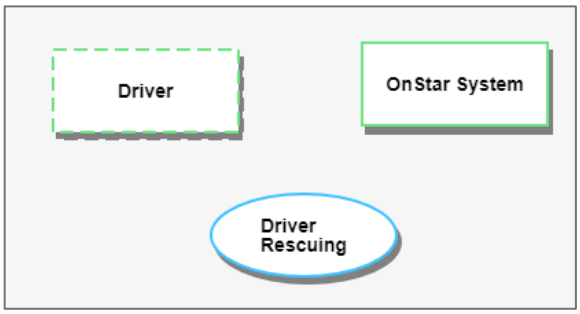


Figure 15. SD after adding OnStar System.

3.8 Thing's Essence

Any OPM thing (object or process) has an attribute called **Essence** (Figure 16), which can have one of two values: **physical** or **informatical.** A physical thing is shaded (tangible; three-dimensional), while an informatical one is not shaded (non-tangible; two-dimensional).

The default essence of things in a model in OPCloud is physical (but can be changed by the modeler or the organization admin), so our process is shaded upon its creation. Since **Driver Rescuing** process is physical, we don't need to change its essence. Please note that the default of the things' essence may vary. Some organization may define the default as informatical while other as physical. Each modeler may change the default for its own liking at the setting menu page.

3.9 Thing's Affiliation

Another attribute of any OPM thing is its **Affiliation** (Figure 16), which can have one of two values: **systemic** (part of the system, the default; denoted by a solid contour) or **environmental** (part of the system's environment; denoted by dashed contour). Our process is systemic, not environmental, so the following OPL sentence appeared in the bottom right pane, titled OPL (Figure 17)

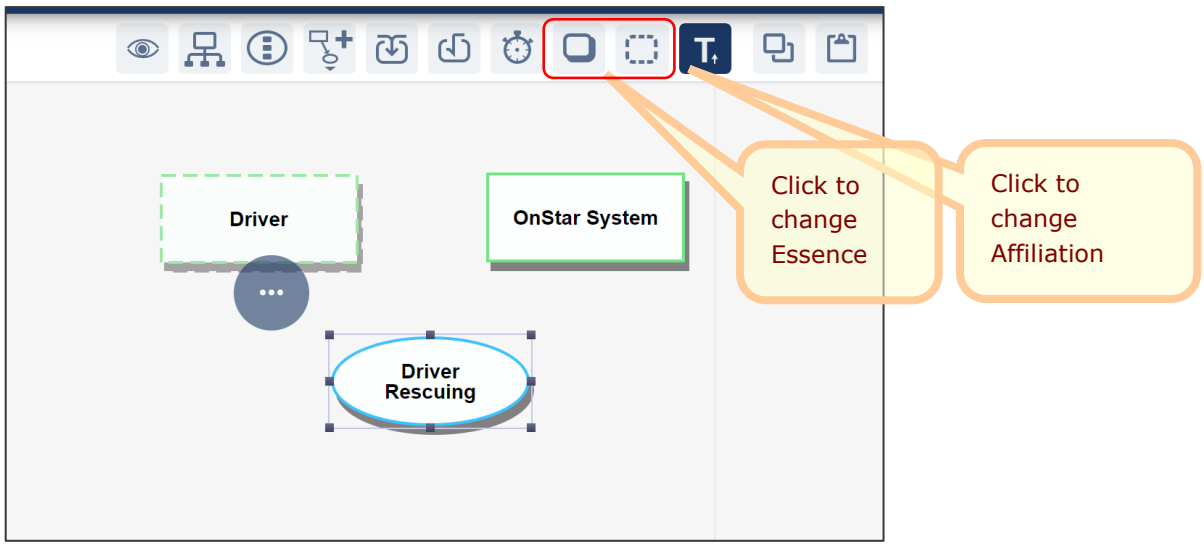


Figure 16. The Change Essence and Change Affiliation buttons marked in red. Things Group in Secondary Toolbar in orange.

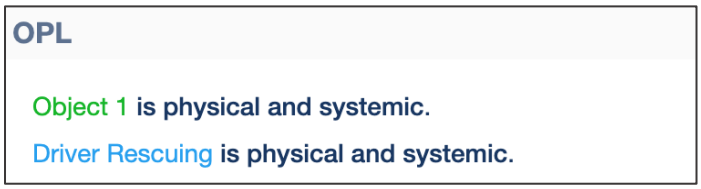


Figure 17. The OPL sentence that appears after adding a process (blue) and an object (green).

To help distinguishing between objects and processes in OPL, the process name in the text is in blue, as is the process ellipse in the OPD. Similarly, object names will be green – the same color of their OPD counterparts (Figure 17)

3.9.1 Changing Essence or Affiliation

We can control the Essence or Affiliation of Elements using the Things menu at the element options toolbar – the white toolbar beneath the main one (Figure 16). The essence of each thing can be changed by the user as needed by highlighting the thing (by clicking on its name) and then clicking the shaded **Change Essence** button on the top right (Figure 16) to toggle the thing's Essence. The default Essence can be changed in the setting screen.

The affiliation of each thing can also be changed by the user as needed by highlighting the thing and then clicking the dashed **Change Affiliation** button on the top right (Figure 16) to toggle the thing's Affiliation

3.10 Creating a link

As the beneficiary, **Driver** is affected by the **Driver Rescuing** process. To denote this, we draw a link from **Driver Rescuing** Process to the **Driver** Object. Please note that Effect Link can be also drown from the Object to the Process.

In order to draw a link from the process **Driver Rescuing** to the object **Driver**, place the mouse inside the process but not on its name, so that the cursor becomes a crosshair. Press the mouse button and drag it over to inside the **Driver** box. A dashed line appears, and the object will be highlighted in orange (see Figure 18).

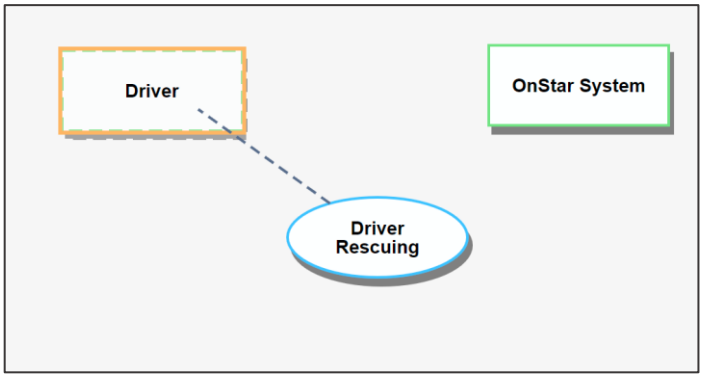


Figure 18. Dragging a link from the Process to Object

A table with all the possible links from an object to a process pops up and you can move it by dragging it from the top, so it does not hide the link being created. The table shows the link symbol and name. For each link, the table shows the OPL sentence that will be added to the OPL paragraph if you select that link.

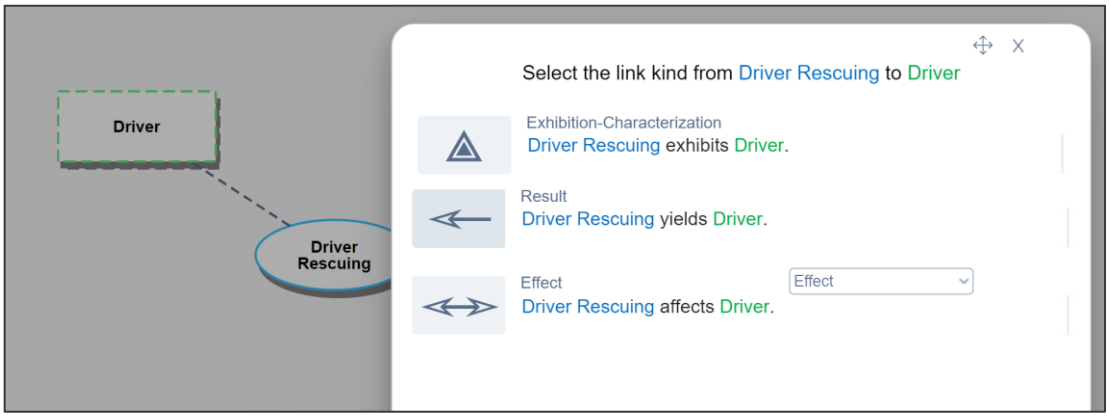


*Figure 19. The links table that pops up and moved to the right after dragging a link from **Driver Rescuing** to **Driver.***

Reading the sentence helps you select the correct link. Since we wish to express that " Driver Rescuing affects Driver. ", we select the **effect link** Figure 19, so it is added to the OPD along with the new OPL sentence at the bottom (Figure 20)

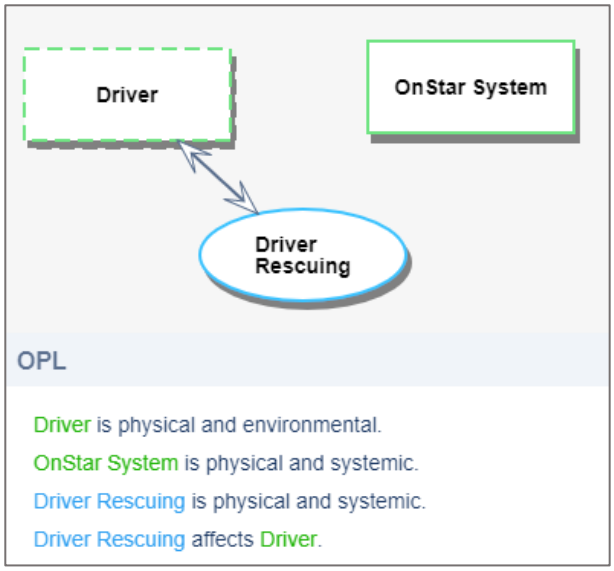


Figure 20. SD after linking Driver and Driver Rescuing with an effect link.

The semantics of this sentence is that the **Driver**'s state is changed by the **Driver Rescuing** process, but at this abstract level of detail, the states themselves are not specified, so the effect link abstracts this expression. At a later stage, we will describe the relationship between the **Driver** and **Driver Rescuing** in more detail. Next, in Figure 21, we want to add the **OnStar System** as an instrument to the **Driver Rescuing** process. To this end, we draw a link from the object **OnStar System** to **Driver Rescuing**.

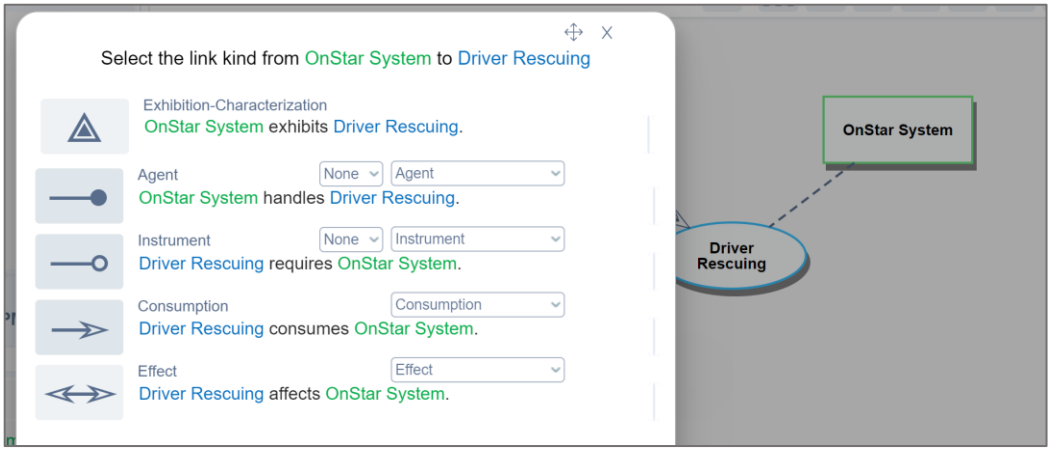


Figure 21. Adding an instrument link from the OnStar System to Driver Rescuing.

Selecting the **instrument link**, we get in Figure 22 the updated OPD with the new OPL sentence.

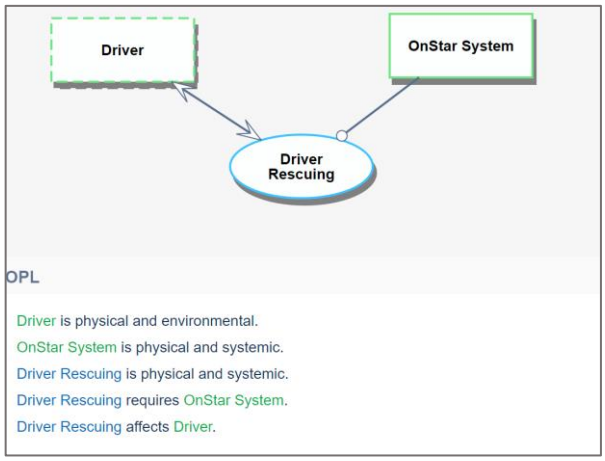


Figure 22. The OnStar System added as an instrument to Driver Rescuing.

3.10.1 Structural links

The two links we used so far, the **effect link** and the **instrument link**, are *procedural links*, as they connect an object and a process, and therefore their sematic is *procedural*. Beside procedural links we also have *structural links*, the second kind of links. Structural links model the *structure* of the system by connecting an object to one or more objects or a process to one or more processes. We are now going to use the first kind of structural link – **the Aggregation-Participation** (whole-part) link.

Reading the OnStar System description, specified in **Annex A**, we find that "*At its most basic, OnStar consists of four different systems: cellular phone, voice recognition, GPS and vehicle telemetry*."

We add these four parts to SD one by one. The location of each object is determined by where we drag the object from the toolbar (see Figure 23). To express the model fact that **Cellular Network** is part of **OnStar System**, we create a link from **OnStar System** (the whole, or aggregate) to **Cellular Network** (the part), as shown in Figure 24.

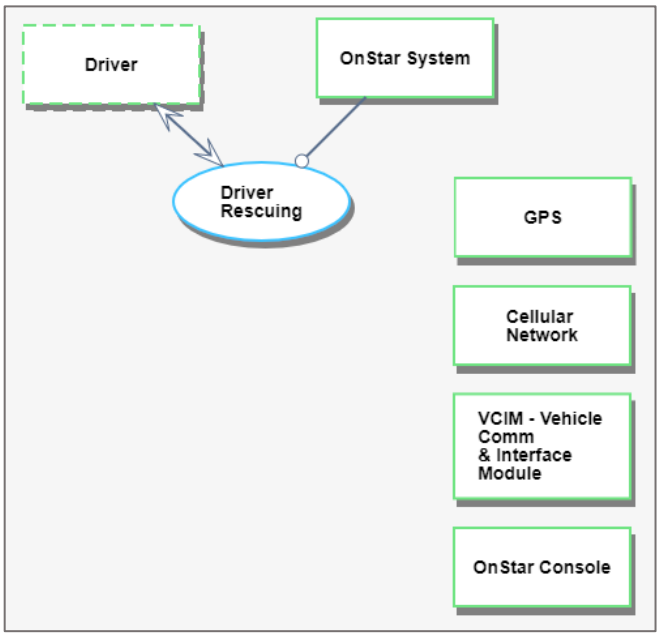


Figure 23. Adding the four parts of OnStar System

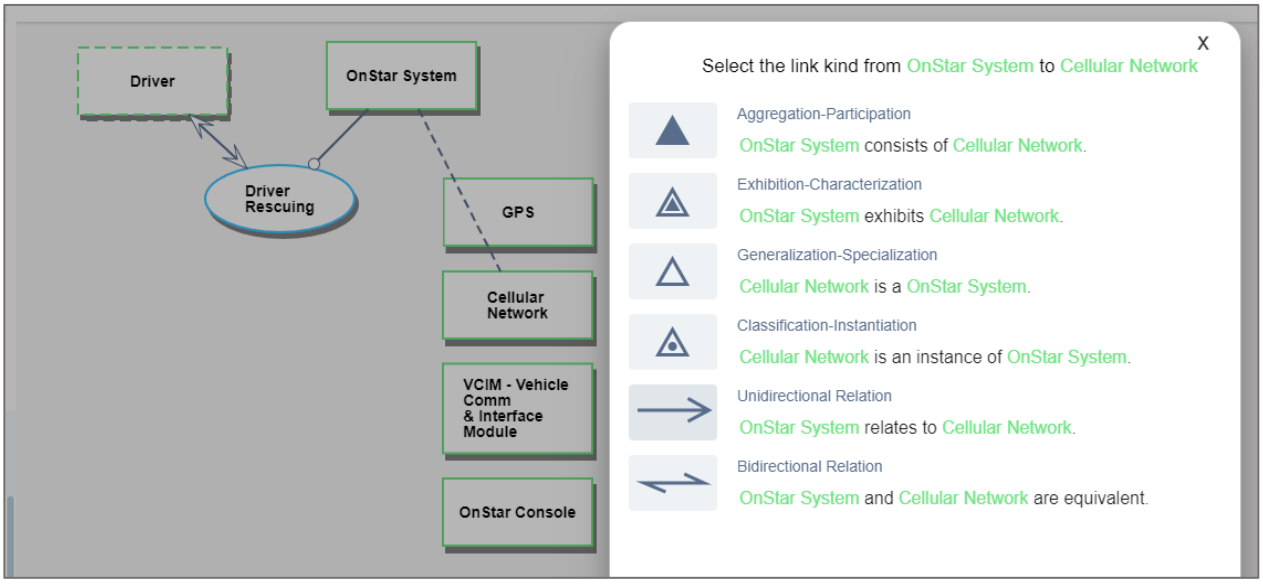


Figure 24. Linking Cellular Network as part of the whole OnStar System and the links table after linking the whole OnStar System to Cellular Network.

The links table that pops up as shown on the left-hand side of in Figure 24. Having read the OPL sentence candidates, we select the aggregation-participation link at the top (the black triangle), and the link is created. We can rearrange the link layout by clicking on the triangle and moving it to the desired location (Figure 25)

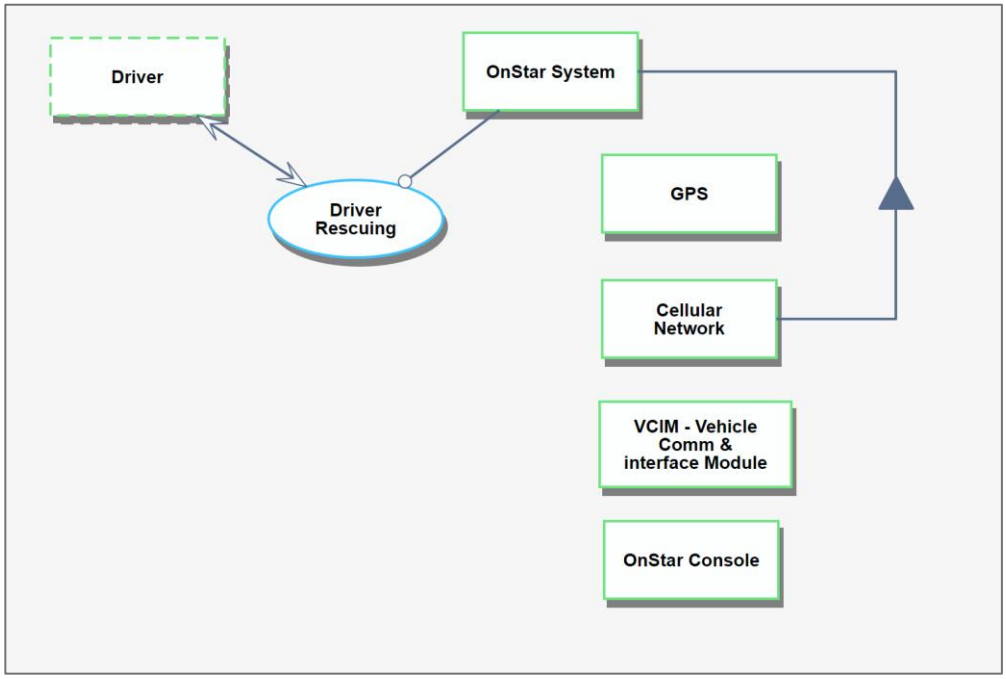


Figure 25. The Aggregation-Participation link is created after selecting it from the links table and moving the black triangle to the right.

Since we have several objects, we can link each object *separately* as we did for **Cellular Network**. In Figure 26 **GPS** is added as part of **OnStar System**.

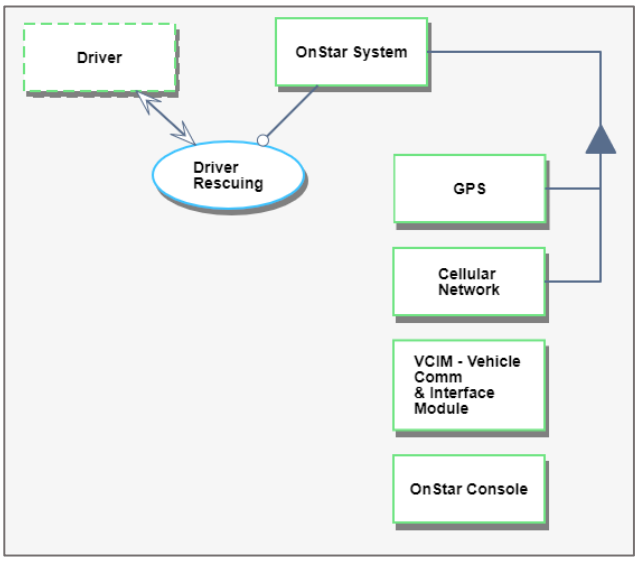


Figure 26. GPS is added as part of OnStar System.

Another option to link many parts (or other refinees) at once is *multiple linking*, which can be done in two ways:

(1) **Multiple-select**: While holding the Ctrl (Control) key, select each one of the objects to be linked, they will be highlighted in orange. Now draw a link to one of the things in the selected group, the result is presented in Figure 27.

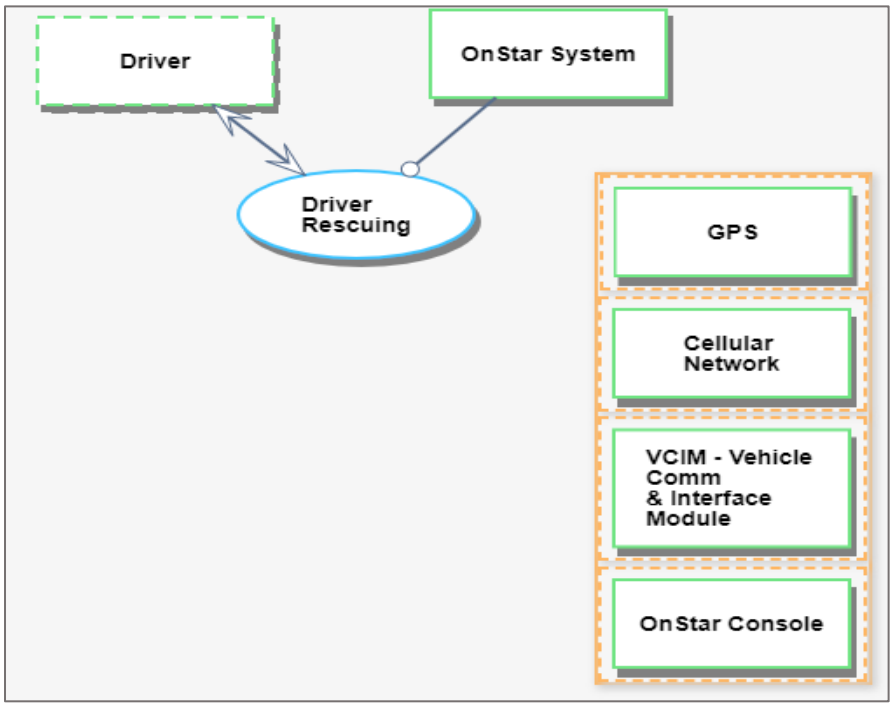


Figure 27. Multiple selection is done by holding the Ctrl key and clicking on the elements one by one.

(2) **Lasso**: Hold the Shift key, select all the four-part objects by a lasso (Figure 28), and draw a link from the **OnStar System** to one of the things in the selected group.

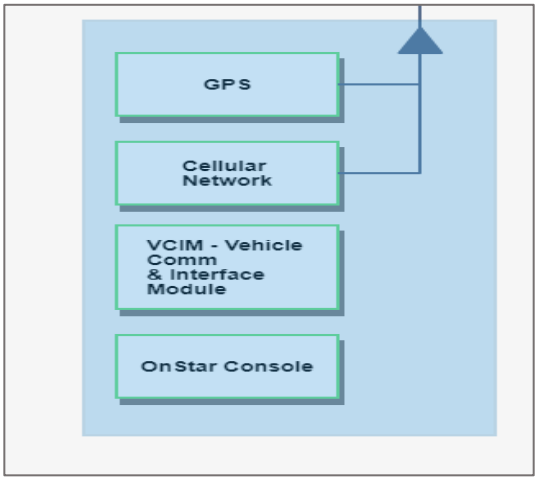


Figure 28. Lasso selection is done by holding the Shift key and dragging the mouse over the area containing the elements to be selected.

After linking the whole to any one of the four-part objects, all the selected objects are connected by the aggregation-participation link (Figure 29). The resulting OPL sentences are shown in the OPL panel, where the object names ordered alphabetically.

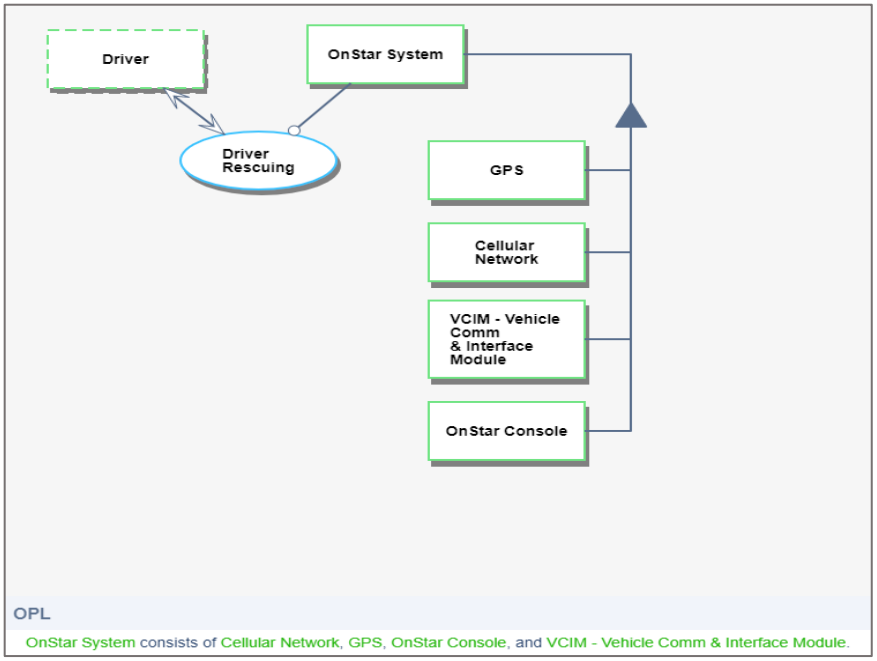


Figure 29. The four parts are now linked to the whole OnStar System.

In order to specify that **Driver** communicates via **OnStar Console**, we draw a link from **Driver** to **OnStar Console** and select the unidirectional tagged link. The result (after rectifying the link) is shown in Figure 30.

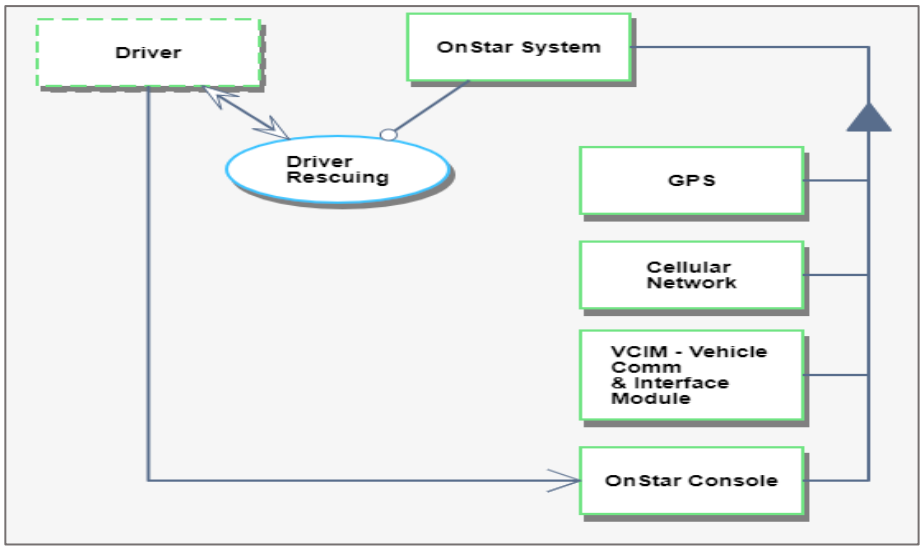


Figure 30. Specifying the relation between the Driver and OnStar Console.

Inspecting the newly generated OPL sentence we see that it reads as follows:

Driver relates to **OnStar Console**.

In order to be more specific, we wish the sentence to be:

Driver communicates with **OnStar Console**.

To do this, we right-click on the tagged link, and a label dialog box opens, as shown in Figure 31. Another option is to double click on the OPL sentence itself on the link name (Figure 32). This will

open the link properties popup menu. (Doing so on a thing name will open the thing's name edit text popup menu).

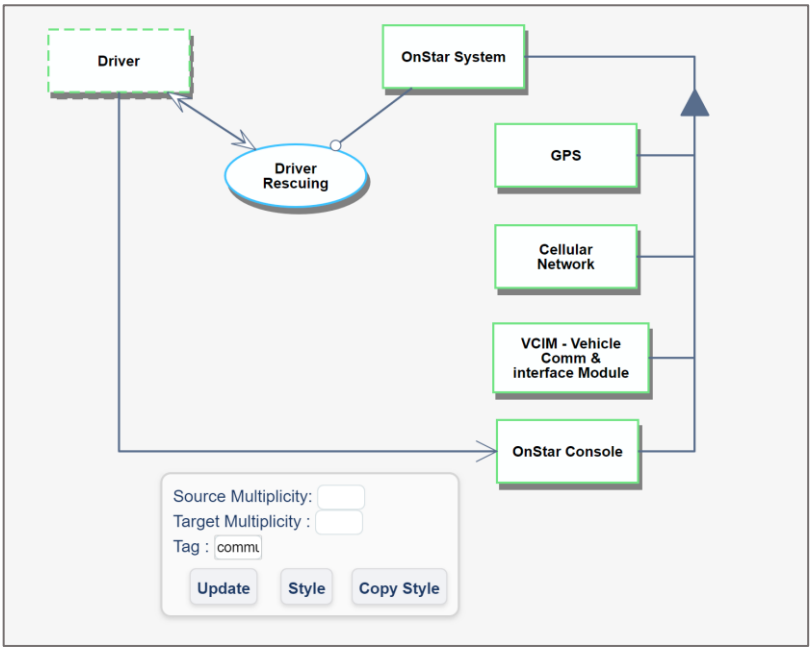


Figure 31. Right clicking on the tagged link opens the label dialog box.

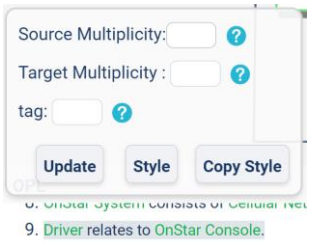
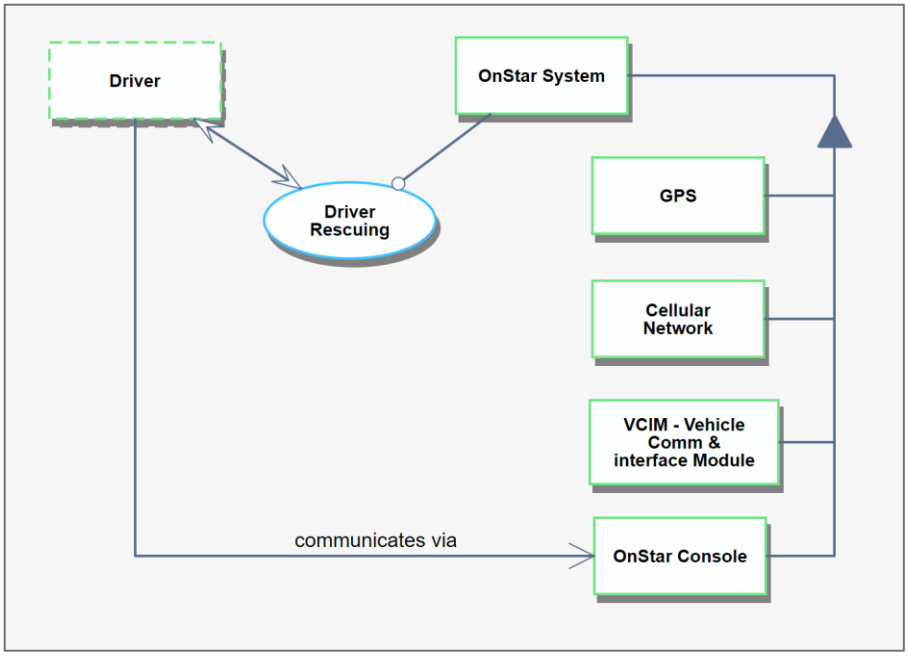


Figure 32. Edit link properties from the OPL module

After typing "communicates via" in the box and clicking update, we get the OPD and the updated OPL sentence shown in Figure 33.



Driver communicates via OnStar Console.

Figure 33. SD after the label "communicates via" is created and repositioned.

3.11 Quick Summary

- The first **OPD** (the **SD –** System Diagram) is the top-level OPD, the root of the OPD tree, which expresses the main function of the system – the main process and the objects involved in this process.
- We start modeling by depicting in SD the ***main process*** and the ***main object*** which that process transforms. This is the ***function*** of the system.
- **Things** (**objects** and **processes**) have an **Essence** attribute and an **Affiliation** attribute.
- The default **Essence** is **physical** (shaded), and the default **Affiliation** is **systemic** (solid line); This is depending on the organization default.
- **Things** which are not physical (such as datum, file, command, message, algorithm) will be marked as informatical.
- **Things** which are not **systemic** – not part of the system (but interact with it) – will be marked as **environmental.**
- Environmental objects interact with our system, but we have no influence over their design.
- We add the main objects in the system, denoting, if needed, their **Essence** as **physical** and their **Affiliation** as **environmental**.
- Each part of the system which will be modeled later is some refinement (specification of parts, specializations, features or instances) of the **Things** in the **SD.**

4. Zooming into the main process

At this point, the System Diagram, SD, is already quite crowded, but we barely scratched the surface of specifying the OnStar System. How are we going to keep on modeling while maintaining the OPD clear and readable?

To the rescue comes the OPM built-in in-zooming mechanism. In-zooming is a refinement mechanism. It enables starting a new Object-Process Diagram (OPD), in which a thing (process or object) is copied from the ancestor OPD, and it is blown-up in order to enable specifying its sub processes (Figure 34).

Along with the in-zoomed thing, things that were attached to it are brought along to maintain the consistency across the entire OPD set (Figure 35)

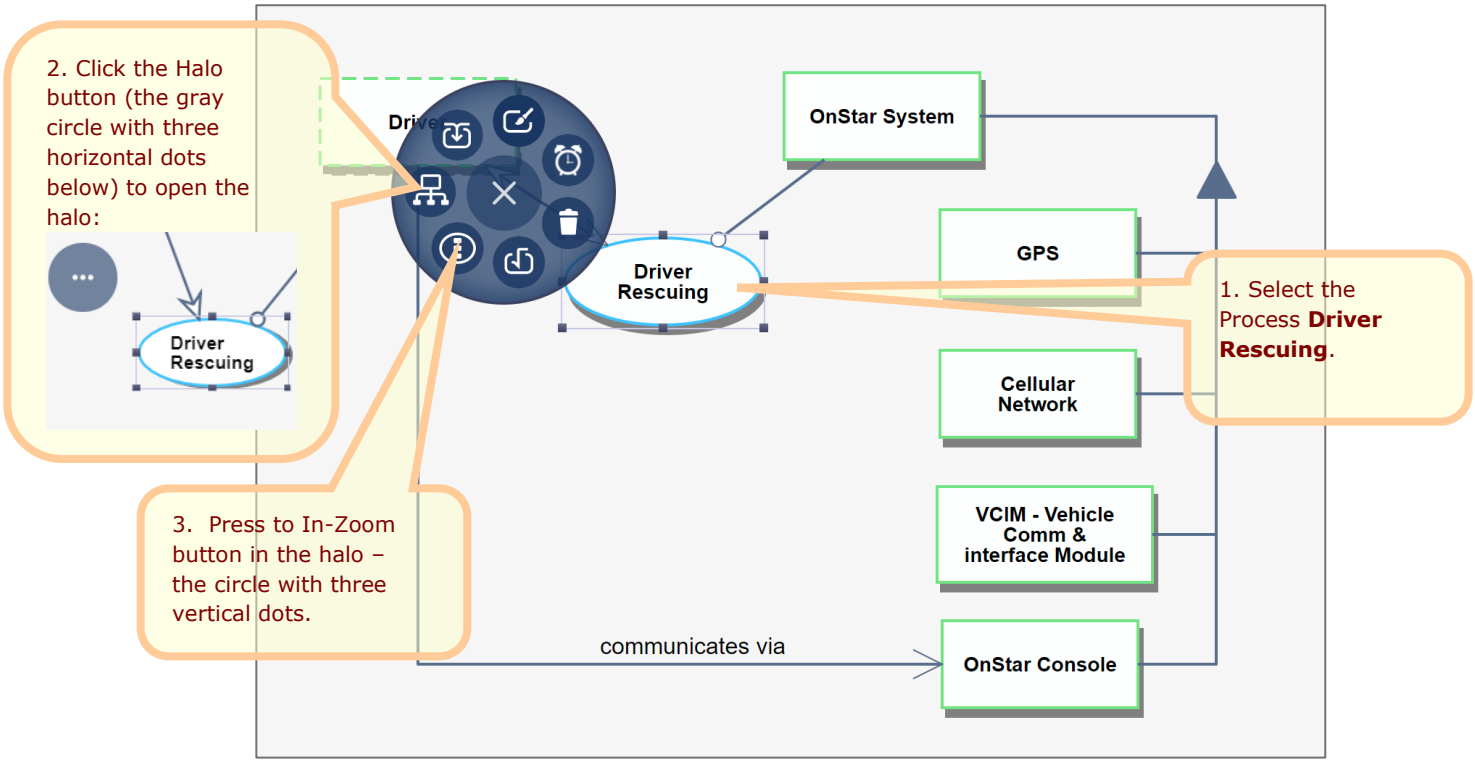


Figure 34. In-Zooming thing into the Driver Rescuing process

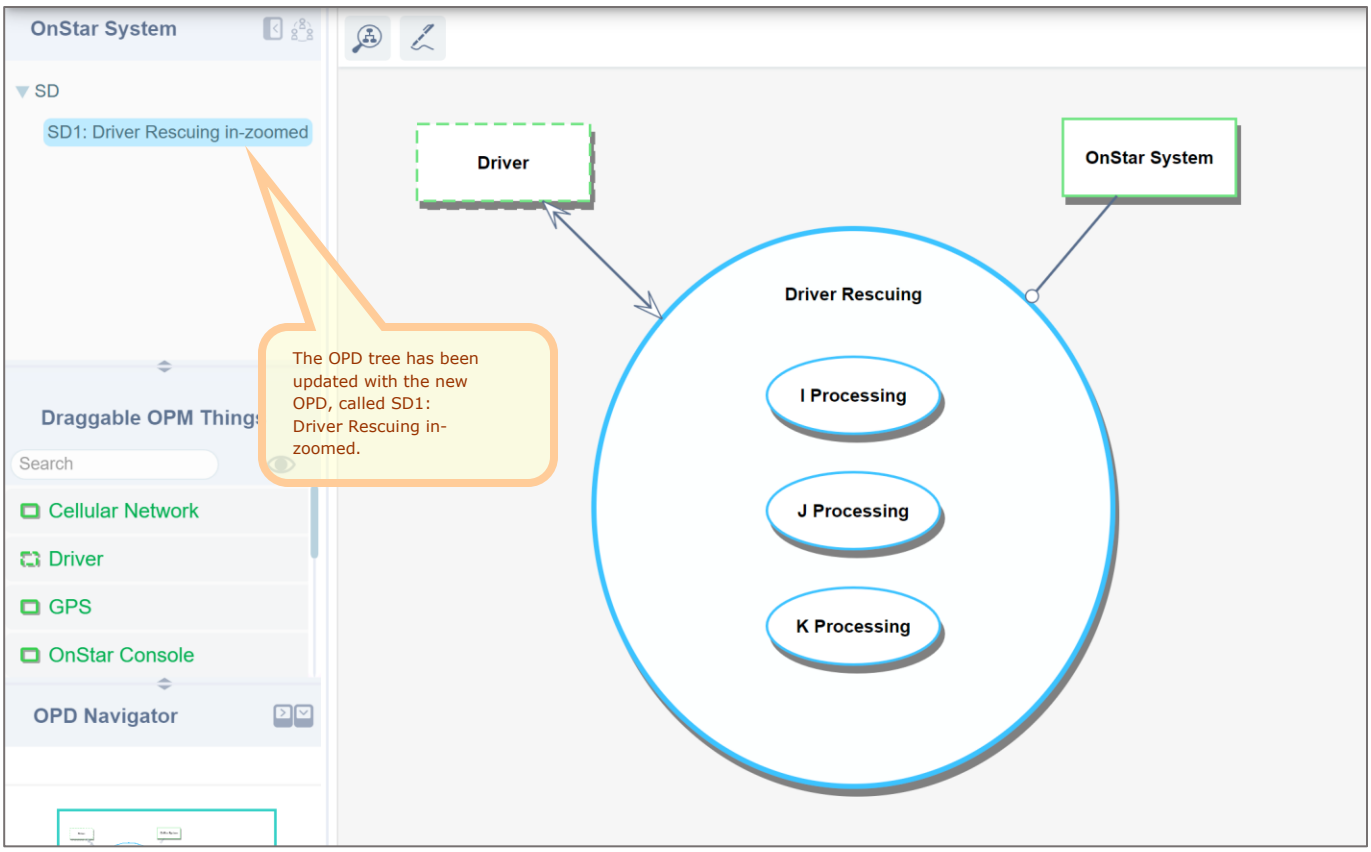


Figure 35. The Driver Rescuing process in-zoomed

As we have seen, in SD we first selected the main function of our system, **Driver Rescuing**, and then added the main involved objects and the main result or effect of the system's function. We now continue to refine the specification of the system using the subprocesses of **Driver Rescuing** as the skeleton for decomposition.

Following the "5 plus or minus 2" law of human cognitive capacity (Shiffrin, 1994), OPCloud adds by default three nested subprocesses, each of which, in turn, can be further in-zoomed (or unfolded) into new subprocesses in the next detail level, SD1.1., SD1.2, etc. SD1 should thus describe the main subprocesses comprising the **Driver Rescuing** process by the **OnStar System**. We can continue this refinement until we are satisfied with the level of detail of the system.

For each process, we will describe its preconditions and its result or effect. We do this by adding objects or object states and linking them to the subprocesses. We continue this till the level of detail required to describe the preconditions and results of the nested subprocesses is sufficient; more details about these additional objects can possibly be added when we zoom into the sub-subprocesses of each subprocess.

Remember that OPM uses detail-level decomposition, i.e., the lower levels present detailed descriptions of OPDs in upper levels. Consistency must be maintained between any two OPDs in the OPD set, so that overall, the entire OPD set is consistent. For additional explanation about this topic, please consult the OPM ISO 19450:2015 specification.

When we edit the names of an in-zoomed process and objects related to it, OPCloud automatically updates the thing names in all the OPDs in which that thing appears.

To place a call in the **OnStar System**, you say outloud a phone number, or a previously stored name associated with a phone number. Modeling this statement presents us with an opportunity to recommend an effective OPCloud design methodology.

The first subprocess of **Driver Rescuing** is **Call Making**. In Figure 36, we edit the name of the topmost subprocess within the enclosing **Driver Rescuing** process.

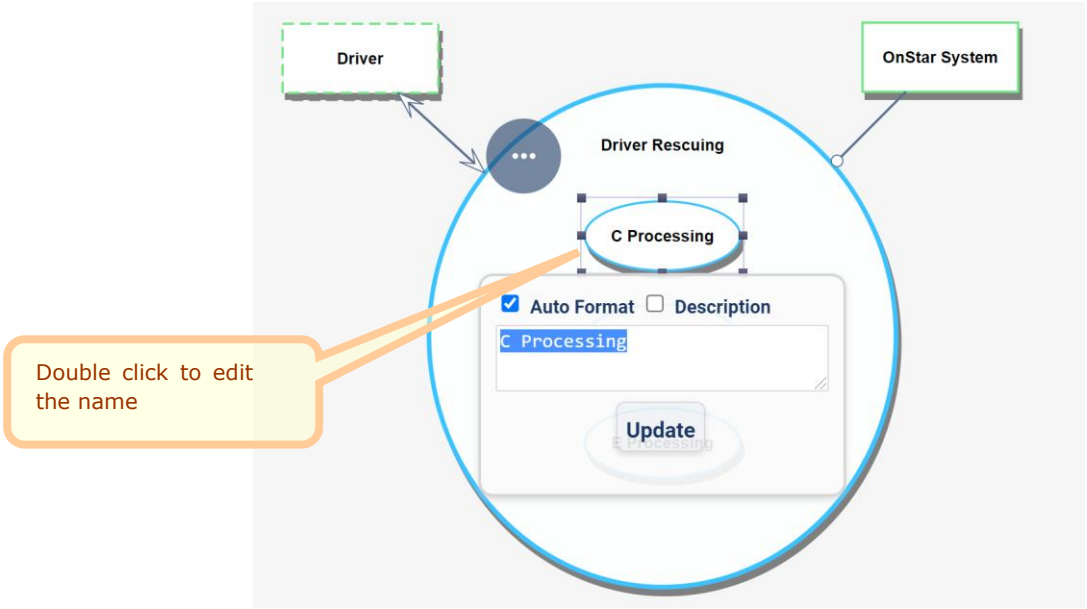


Figure 36. Call Making is the first subprocess within the in-zoomed Driver Rescuing process

Call Making requires the **Driver** and the **OnStar Console**. In order to express this, we need to bring **OnStar Console** to this OPD. Since **OnStar Console** is not connected directly to the process, it was not included in SD1, the OPD of the in-zoomed Process.

In order to do so, as Figure 37 shows, select **OnStar System**, Open the halo, and click on the "Bring Connected Things" Icon.

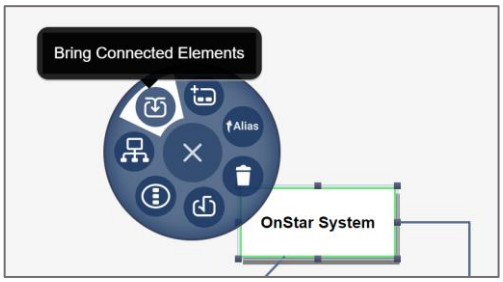


Figure 37. The Bring Connected Things button for OnStar System

Now it's time to specify that the **Driver** and **OnStar Console** are required for **Call Making**. **OnStar Console** is required for **Call Making**, and therefore in Figure 38 is connected with an *instrument link*. **Driver** is also required for **Call Making**, but since **Driver** is a human, he is connected with

an *agent link* – the other kind of enabling link. The agent link denotes that **Driver** is a human required for the **Call Making** process.

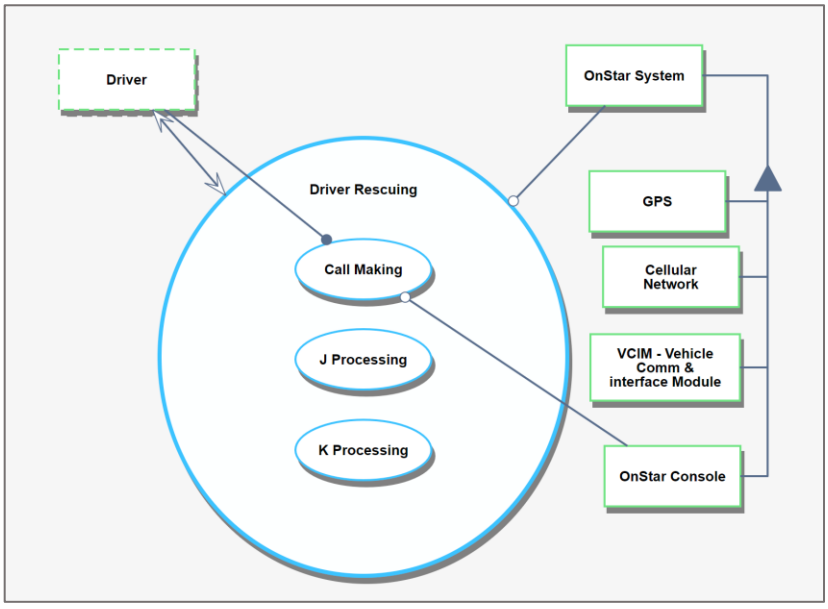


Figure 38. Driver and OnStar Console connected to Call Making

Now we need to ask ourselves what the result or effect of **Call Making** is. In this case the result is **a call**.

We will need to add a new object, **Call**, and insert it into the process. To create the new object inside the in-zoomed process, drag a new object directly to the in-zoomed process. The name is already highlighted (Figure 39), so just type **Call** (see Figure 40). You can rearrange the things within the in-zoomed process for better readability, but make sure that you do not inadvertently alter the vertical order of the states, as this determines their order of execution!

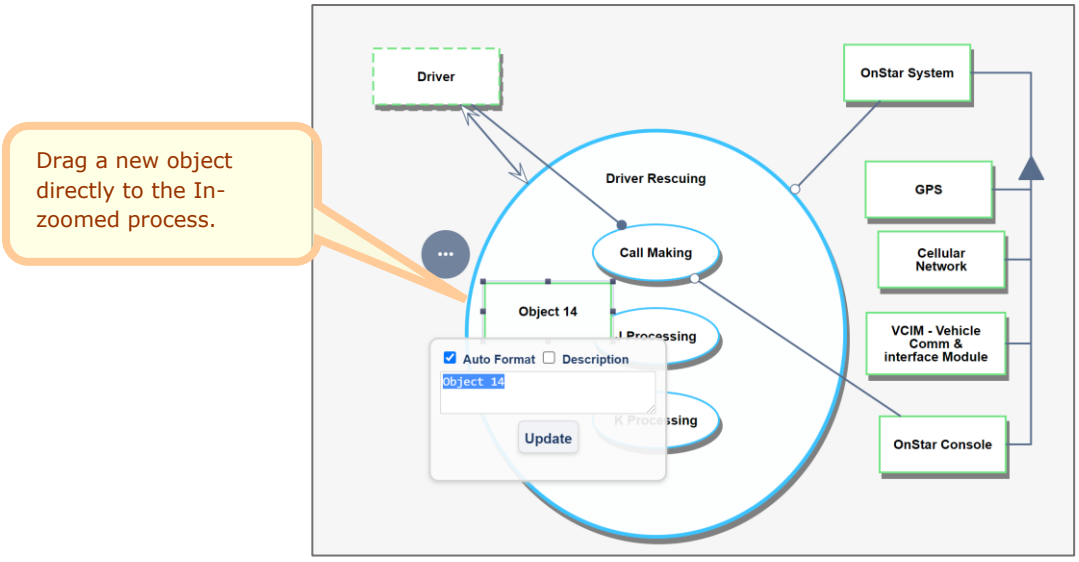


Figure 39. Placing a new object inside the in-zoomed process

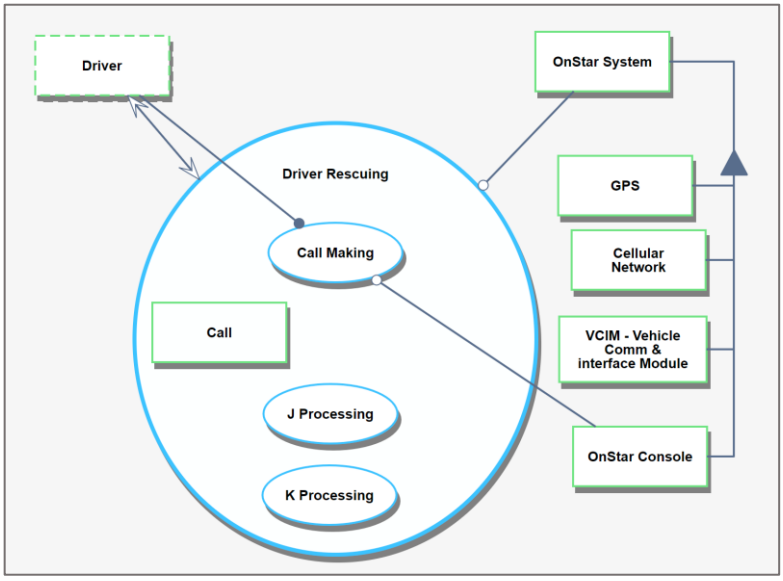


Figure 40. The object Call added into the in-zoomed process

Now create a **result link** from **Call Making** to **Call** by dragging the link from **Call Making** to **Call** and selecting the result link from the table, as shown in Figure 41, Figure 42, and Figure 43.

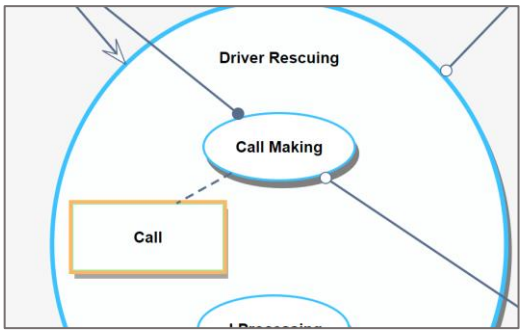


Figure 41. Connecting Call Making to Call

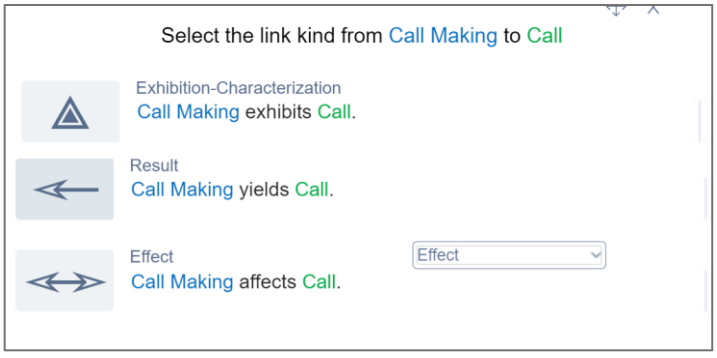


Figure 42. The Links table that pops up after linking Call Making to Call. We select the Result link

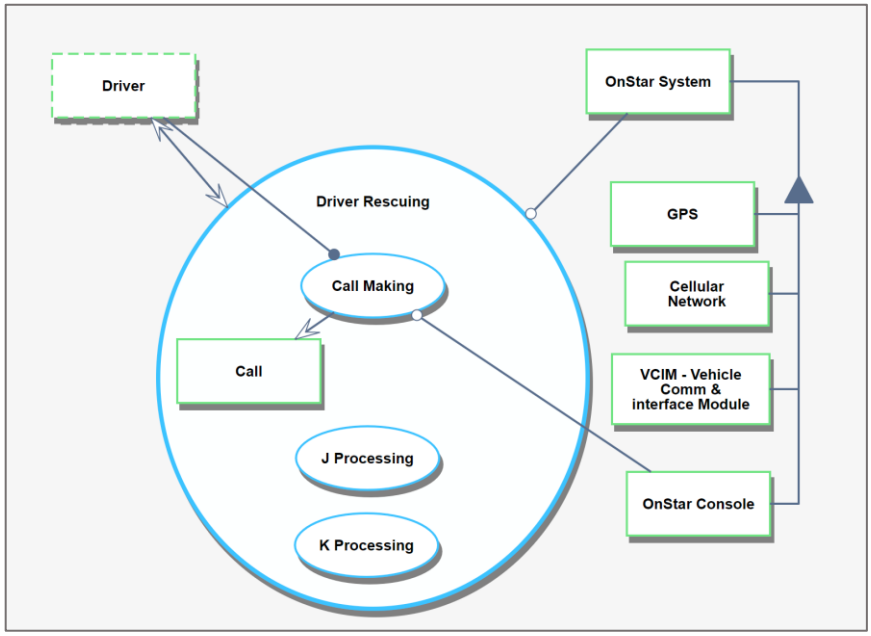


Figure 43. A Result link is created from Call Making to Call

Once we have a **Call**, the next subprocess is **Call Transmitting**. Subprocesses within an in-zoomed process happen sequentially from top down. If we wish to model processes happening in parallel, we need to position them, so their ellipse topmost points are at the same height. For design of asynchronous processes that are triggered without a pre-set timeline, please refer to the OPM ISO 19450. In Figure 44 we make sure that the next subprocess is placed below **Call** and rename it **Call Transmitting**.

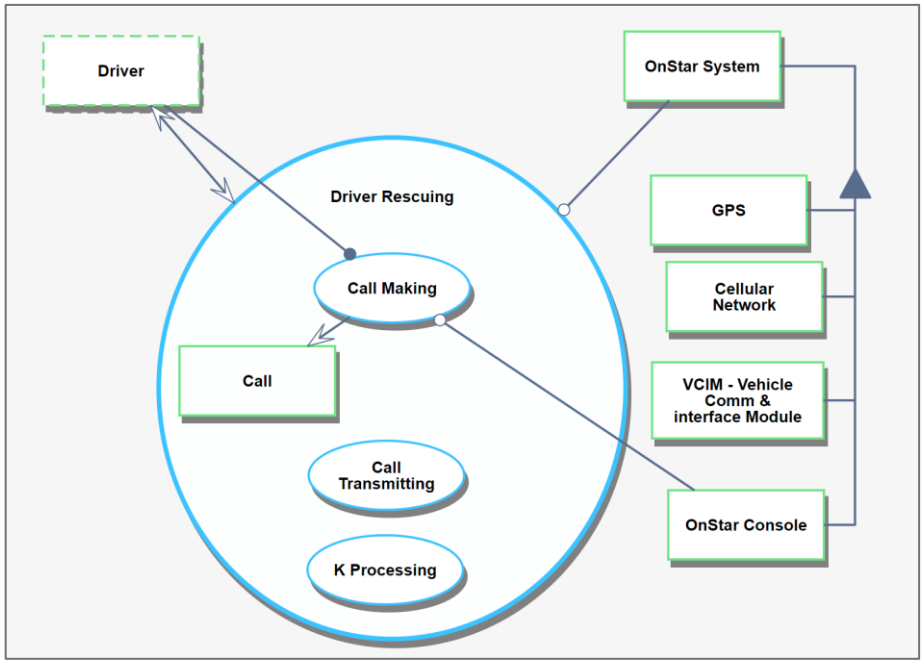


Figure 44. Call Transmitting is renamed from D Processing

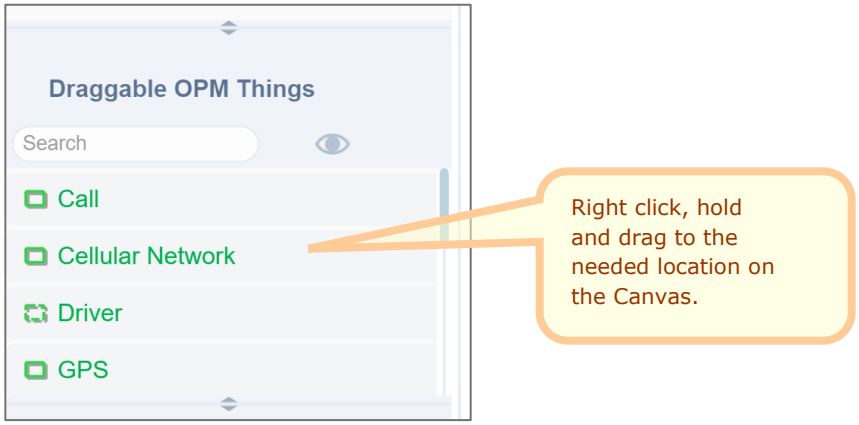


Figure 45. The Draggable OPM Things menu

Call Transmitting uses the **Cellular Network** and affects the **Call**. If **Cellular Network** were not drawn in SD1, we would have needed to bring it. You could have dragged it from the '**Draggable OPM Things**' menu in the left pane to the canvas (see Figure 45)
In Parallel, **Vehicle Location Calculating** uses the **GPS** to produce the **Vehicle Location**. You can add the new objects and links as instructed in previous steps to get the OPD in Figure 46.

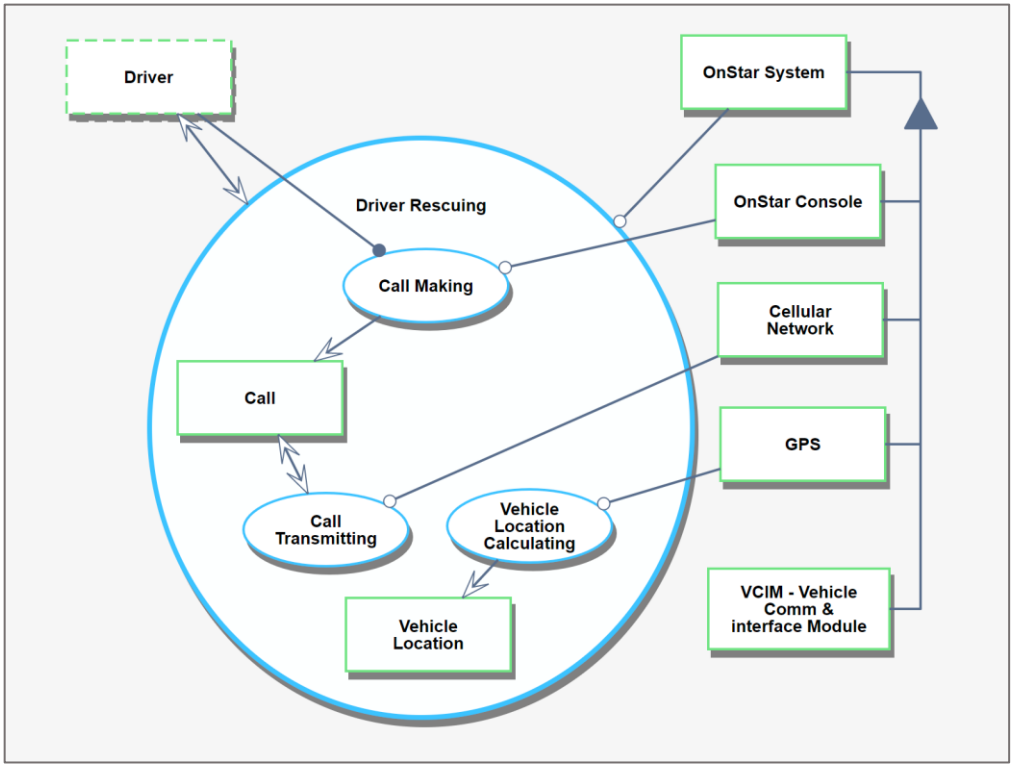


Figure 46. Adding Vehicle Location Calculating and other links and objects

4.1 Quick Summary

- We in-zoom into the main process and then specify its different subprocesses.
- The process activation order within an in-zoomed process follows the timeline, which is from top down. Parallel processes will be positioned vertically next to each other such that their topmost ellipse points are at the same height (y axis).
- Asynchronous processes, i.e., processes with no specific execution order, will be modeled outside the in-zoomed process. For more information about this please refer to OPM ISO 19450:2015.
- In parallel to specifying the subprocesses, we describe the objects involved in these subprocesses. Each object will be connected to the relevant process with the appropriate procedural link. Objects that are required as instrument for all the subprocesses or are agents who handle all these subprocesses can be connected with one link to the outer contour of the entire in-zoomed process.

5. Basic Conditional Flow

As explained previously, the effect link between **Call Transmitting** and **Call** might be replaced in lower OPDs by a set of one or more links with more specific semantics. We want to add states to **Call** the two states **requested** and **online** and specify that **Call Transmitting** changes **Call** from **requested** to **online**. This fact is also represented by the corresponding OPL sentence.

5.1 Adding States

In order to add states, we select the object **Call** and click on the 'Add States' button  in the second tool bar or in the halo (Figure 47). By default, two states are added. Each click on the 'Add States' button after the first two are added shall add a single state. The default names for the first two states are "**state1**" and "**state2**". States added later added will have the default names "**state3**", "**state4**", etc.
Once created, the text of **state1** is selected, and can be directly edited. Update the text, and when done, click 'Enter'. The second state text will be automatically selected and ready to be renamed. Each click on Enter will highlight the next state if its name starts with "**state**".

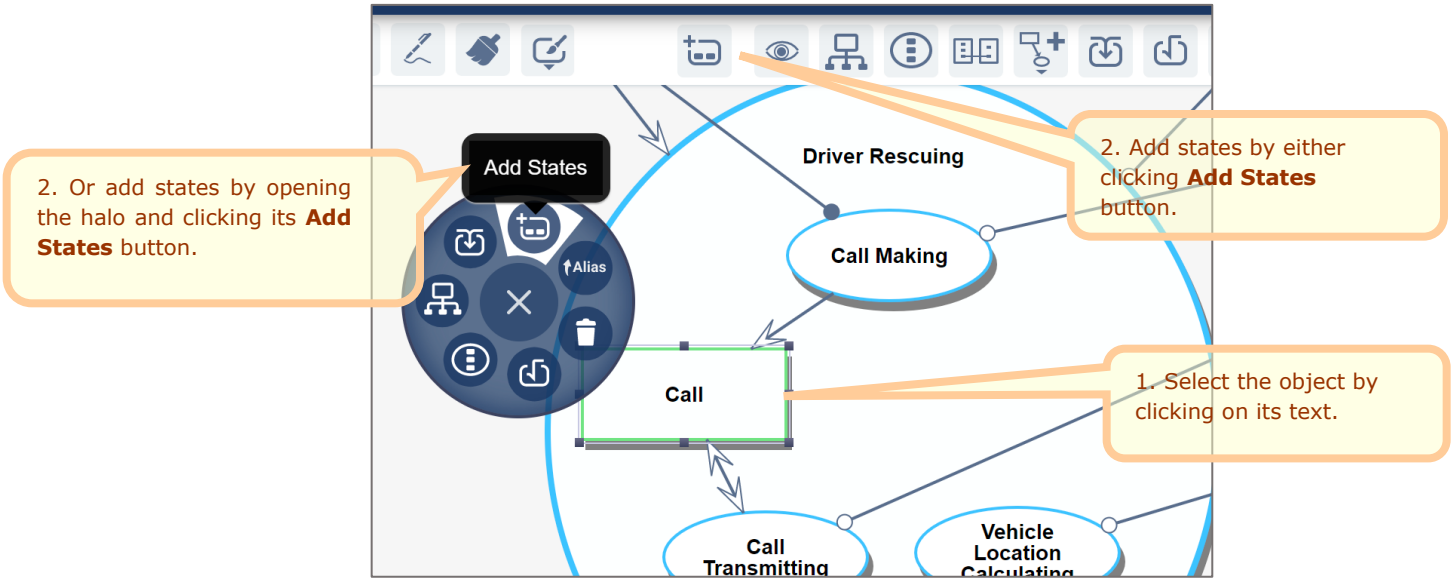


Figure 47. Selecting the Call object and 'Add States' in Halo or toolbar to add states to Call

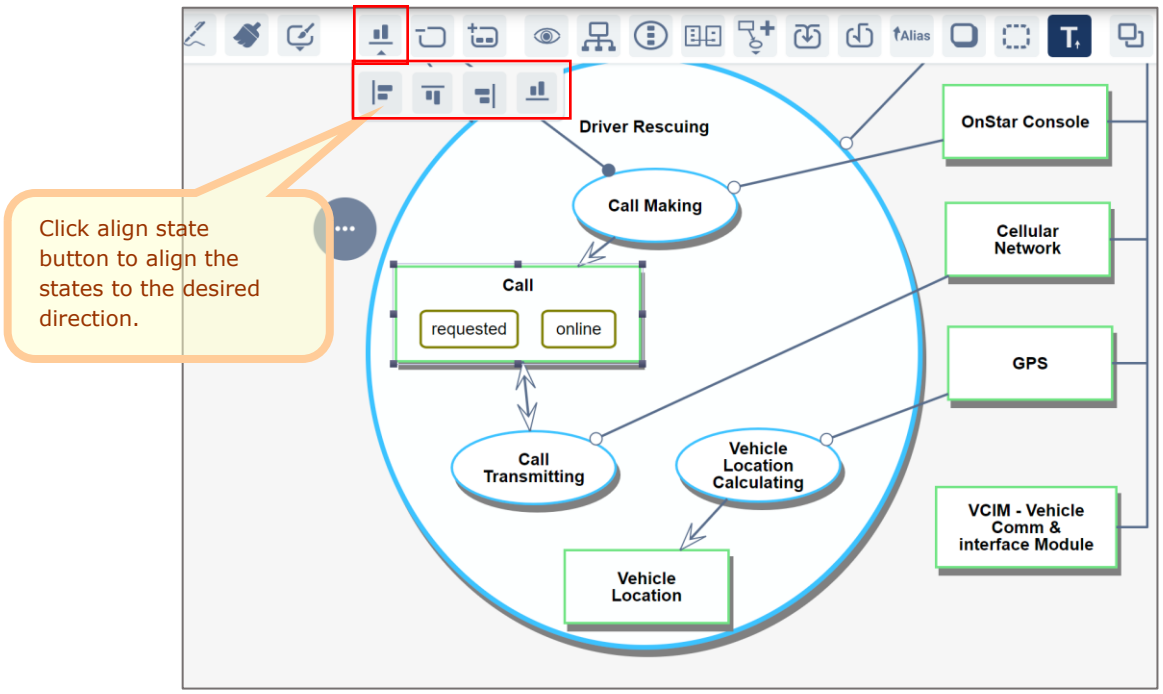


Figure 48. Call with its updated state names requested and online, and the four states alignment buttons

The states **requested** and **online** was created (see Figure 48, and Figure 49). If we want to align the state, we can use one of the four alignment buttons on the second toolbar (Figure 48).
In order to replace the **effect link** with **in-out link pair** we can delete and recrate the link or use the auto-switch of the effect link. If we select the first option, we need to delete the effect link so we can replace it with an **in-out link pair**. Hover over the effect link, see the red X and click on it (Figure 49), The delete element popup menu will be opened (Figure 50). There you will be able to select whether to delete the element locally (only visually if it has more than one visual representation in the model or logically – from the entire model). Once selected the link will be deleted.

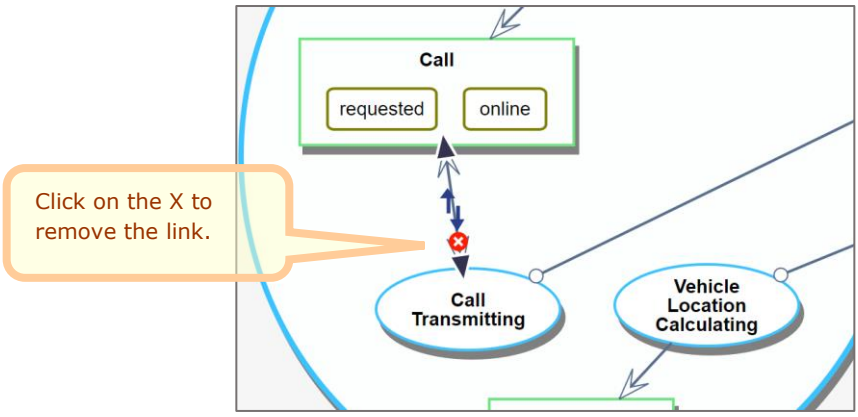


Figure 49. Delete the link icon and switched link icon

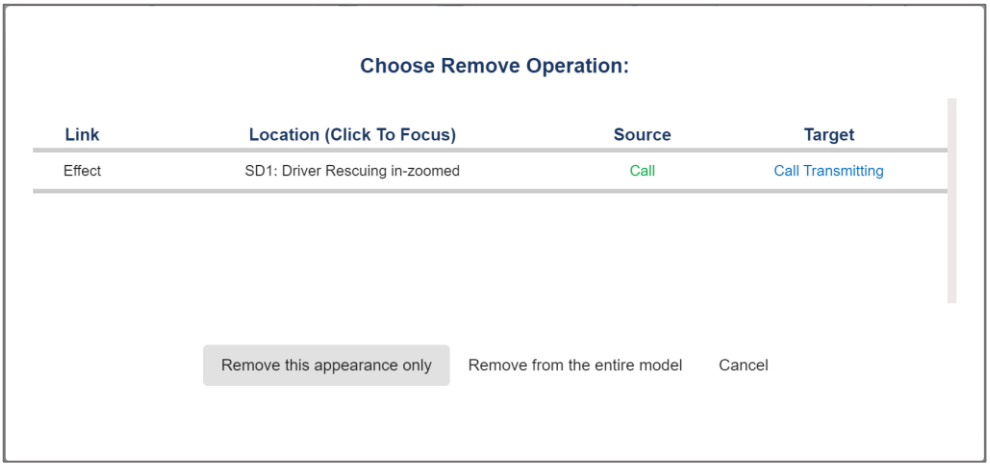


Figure 50. Delete element popup menu

If you didn't update the states name, you need to update them now. Do it by double clicking state1 and state2 and typing the names **online** and **requested**, respectively. We now drag a link from the state **requested** to the process **Call Transmitting**. The link menu will open, and we can select the **in/out link pair** (see Figure 51, Figure 52, and Figure 53).

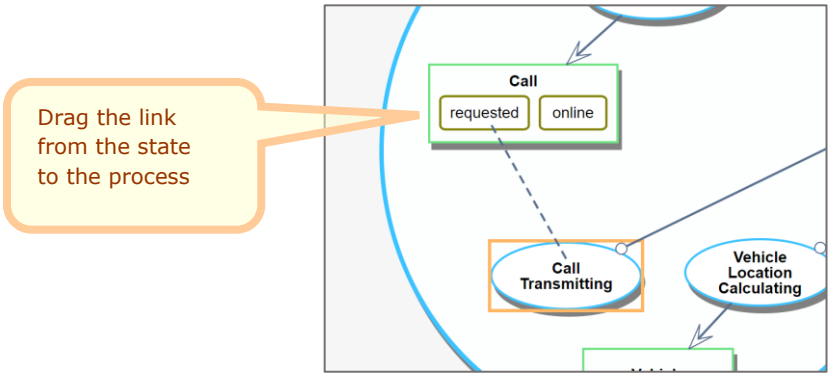


Figure 51. In-out link pair creating

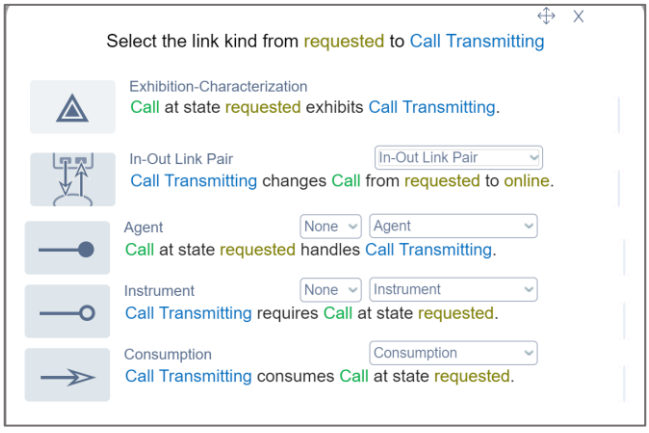


Figure 52. Connecting a state and a process with In-out link pair

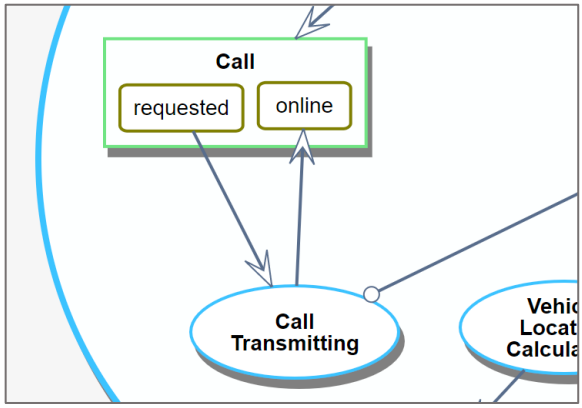


Figure 53. In Out link pair was created

If we want to use the second option for creating the in-out link pair, we can click on the auto-switch icon when we hover over the link and click on it (Figure 49). We will get the switched in-out link (Figure 54).

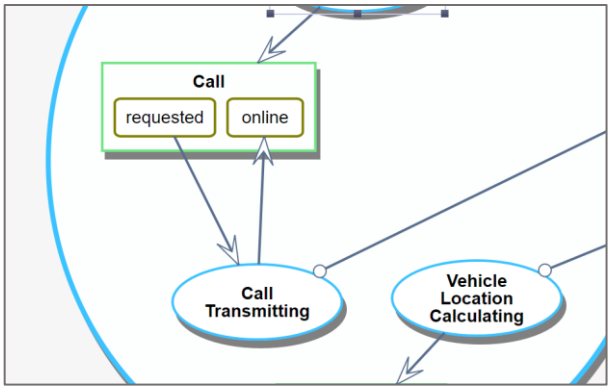


Figure 54. Switched In-Out link pair

Figure 55 express an alternative design in which **Vehicle Location Calculating** is occurring only when **Call** is **online**. We also added an *Instrument link* from **Call** to **Vehicle Location Calculating**.

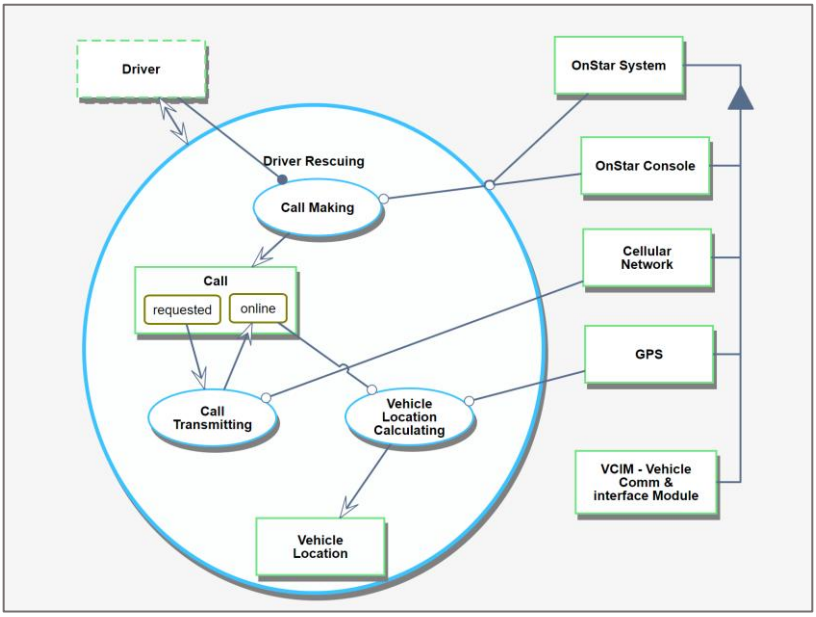


Figure 55. Conditional flow

Additional information about conditional flows can be found in the OPM ISO 19450:2015.

6. Consistency

Let's now continue to the final subprocess of **Drive Rescuing**. According to the **OnStar System** description at this stage an **OnStar Advisor** handles the call. Let's add **OnStar Advisor** to our design (Figure 56). We will now add another process (as we did before, by dragging a process from the upper toolbar) directly into the in-zoomed process. And name it **Call Handling** (**Error! Reference source not found.**)

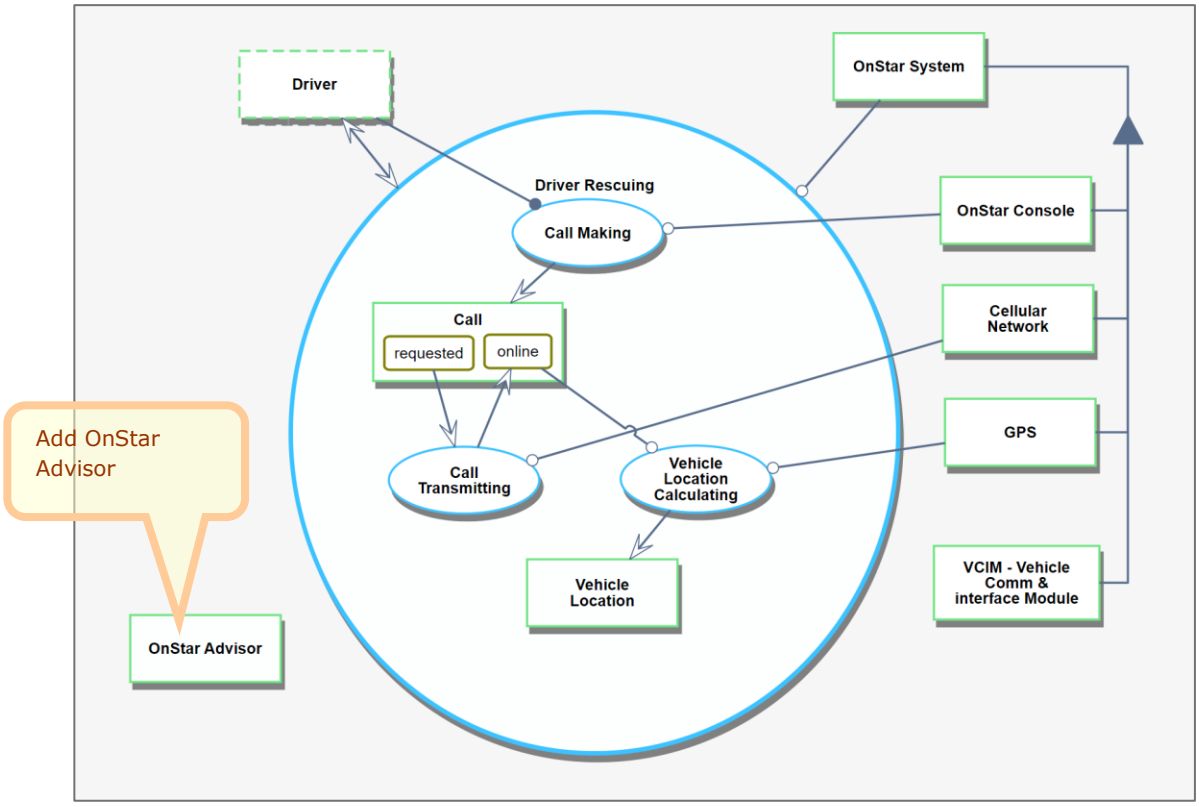


Figure 56. Adding OnStar Advisor

In OPCloud, embedding an object or a process inside another process, as we just did (Figure 57), is available only for an in-zoomed process, and only directly from the upper toolbar.

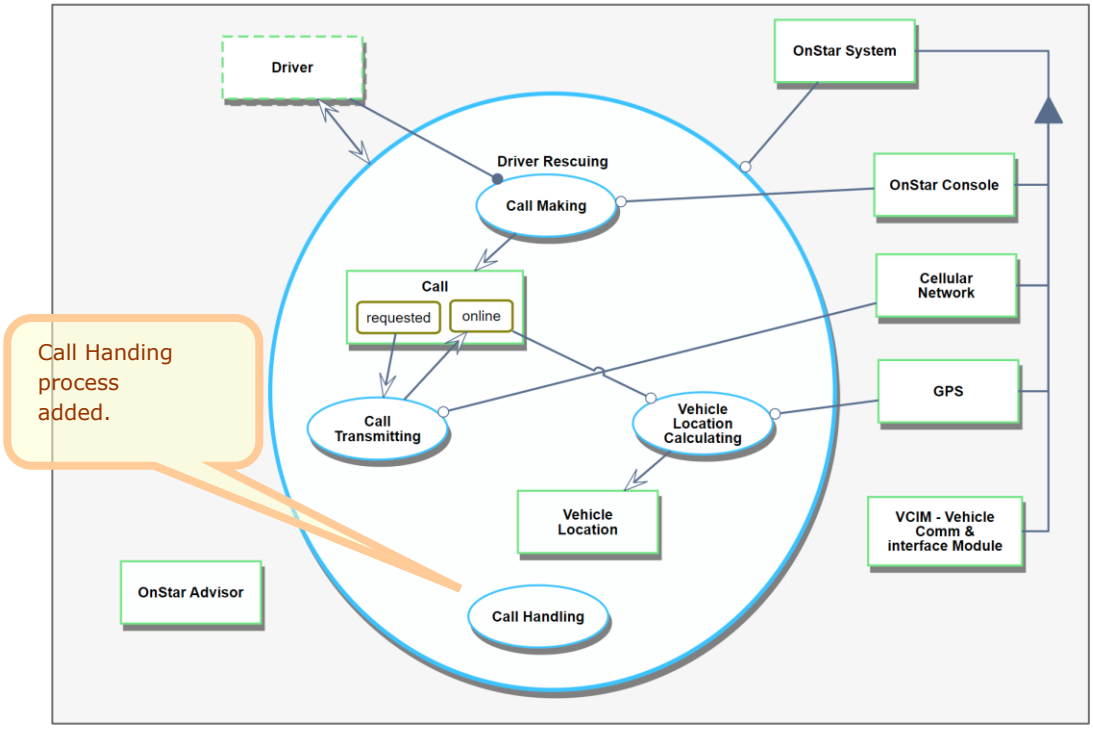


Figure 57. Call Handling process added to the in-zoomed process

OnStar Advisor is added outside the process, while **Call** was previously added inside the processes. The rule here is that any object that is linked only to subprocesses of a process **P** and does not exist or is not required to exist when execution of **P** is over, will be positioned inside **P**, so it is recognized only within the scope of **P**. Otherwise the object must be positioned outside the process.

As noted, a thing at a lower level, detailed OPD, refines a thing that is already modeled at an upper level OPD. In our case, is the **OnStar Advisor** part of the **OnStar System** or any other object at the upper level? The answer is yes, so we add it also in SD.

In later versions OPCloud will draw our attention to the need for consistency.

Go to the SD by clicking on the SD name in the OPD tree on the left. Go to the draggable OPM things menu and drag the **OnStar Advisor** to the canvas. Connect the **OnStar Advisor** to the main process with *Agent link* (Figure 58)

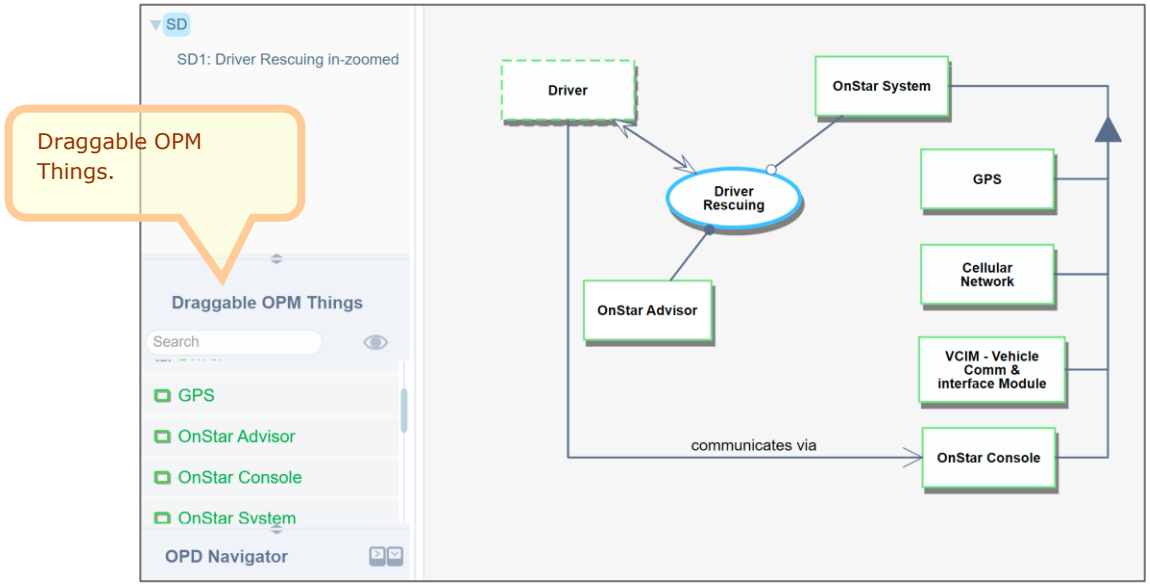


Figure 58. Adding the OnStar Advisor in SD

Since the **OnStar Advisor** is a human, we connect the **OnStar Advisor** using an *agent link* to **Driver Rescuing** at SD. At SD1 we will connect an agent link to **Call Handling**, the process for which **OnStar Advisor** is agent (see Figure 58).

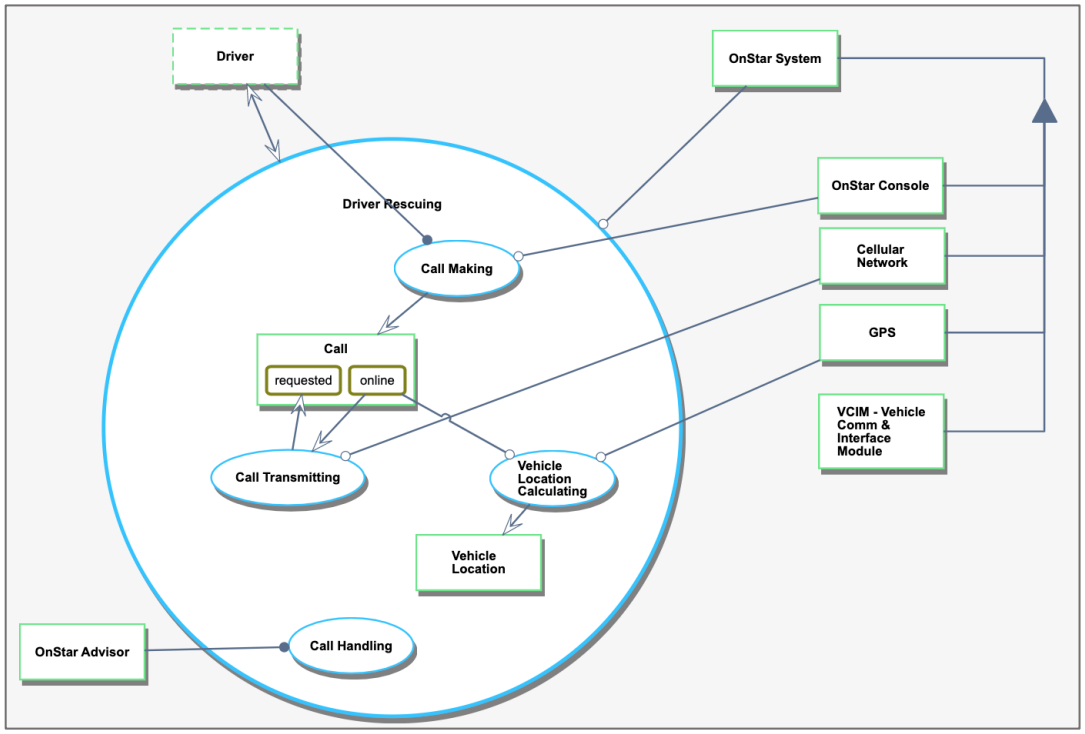


Figure 59. Adding OnStar Advisor link to Call Handing in SD1

In addition, for **Call Handling** to happen, we need the **Driver** as another agent and the **Vehicle Location** as instrument (Figure 59 and Figure 60).

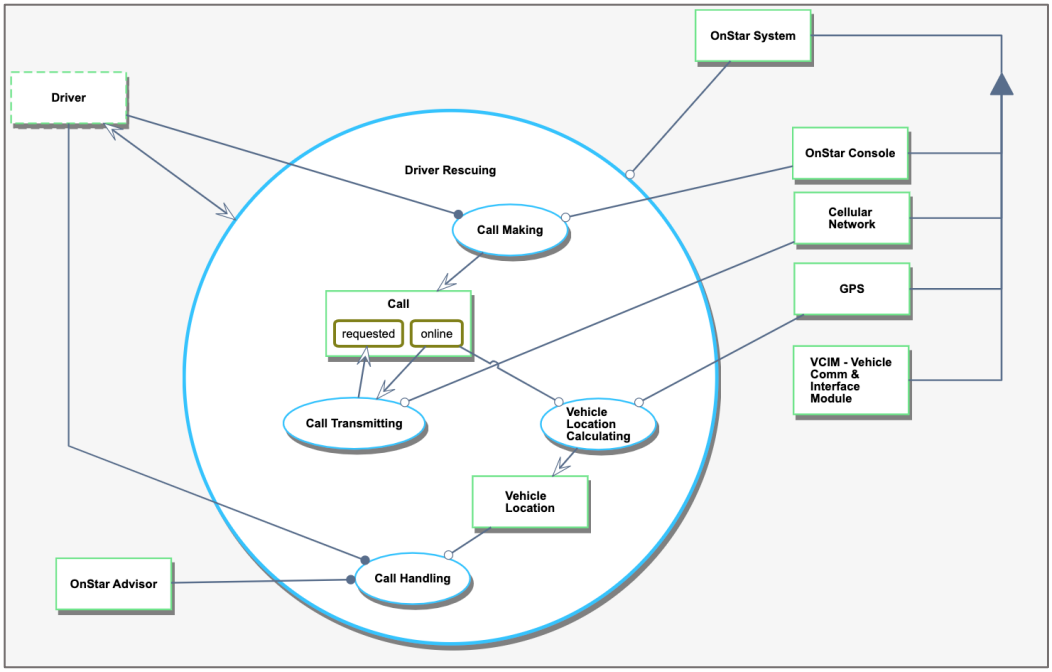


Figure 60. Connecting Driver and Vehicle Location

Eventually, the **Call Handling** process changes the **Driver**'s **Danger Status**, an attribute of **Driver**, from **endangered** to **safe**.

We will now add the new Object with its states, Figure 61, as we created and added above (see Figure 47 and Figure 48).

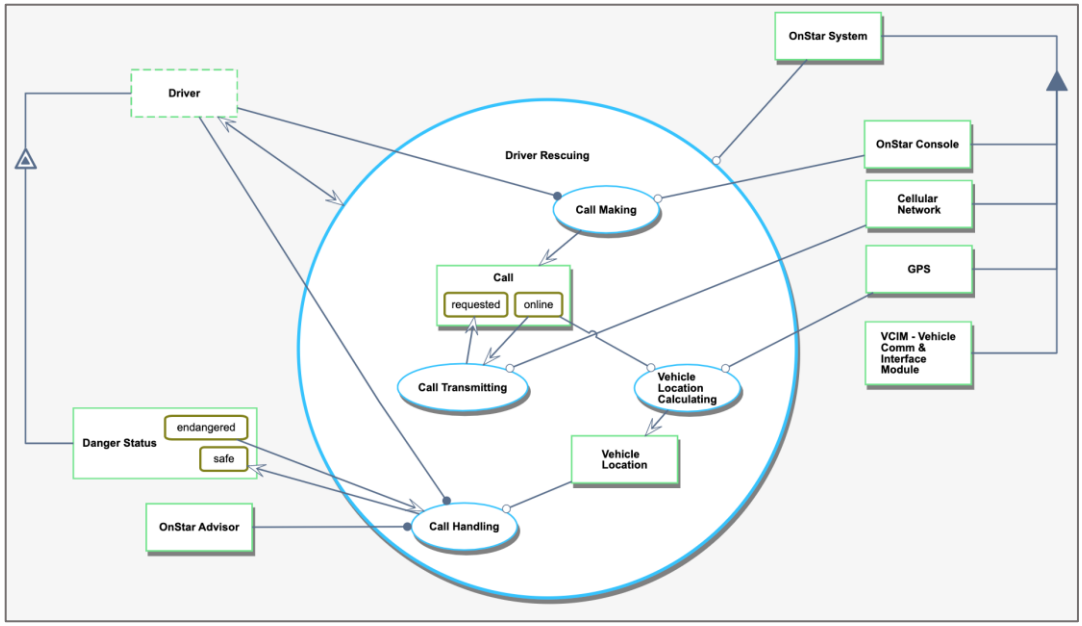


Figure 61. Call Handling changes the state of Danger Status from endangered to safe

7. Marking Initial States

At this stage, we may want to emphasize the fact that when the system starts to operate, the **Danger Status** is in the state **endangered**. We may also want to specify that **Call Making** creates the object **Call** in the state **requested**. We can do this by marking these states as initial (Figure 62).

To do so, select the state **requested**, and in the second toolbar you will "define state type" menu. Select the "initial" state type (Figure 63), and the same for state **endangered,** select the "final" state type (Figure 64)

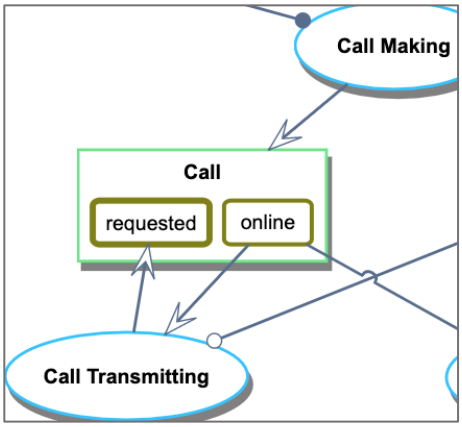


Figure 62. State marked as initial

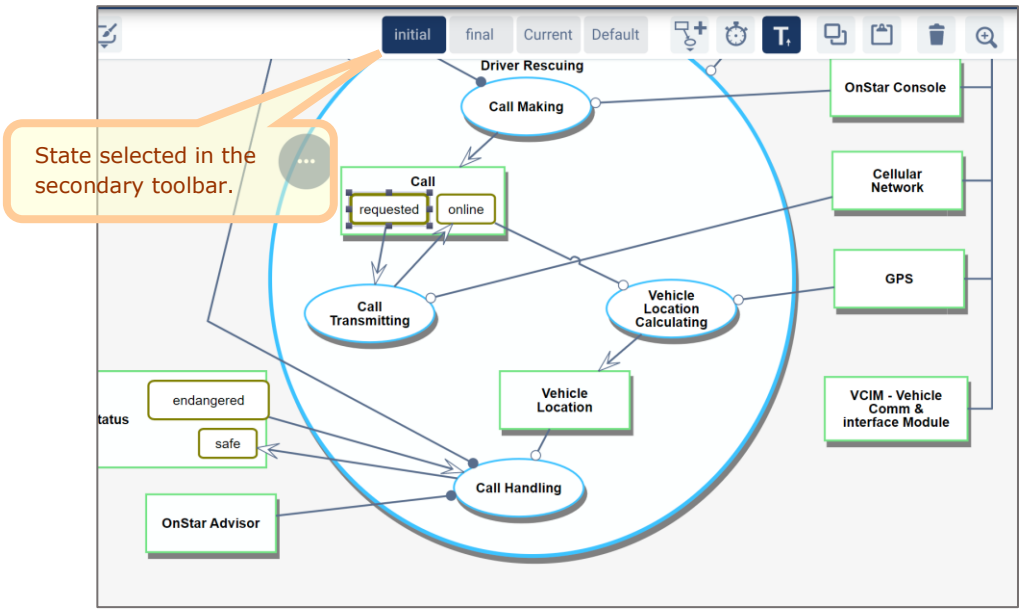


Figure 63. State selected

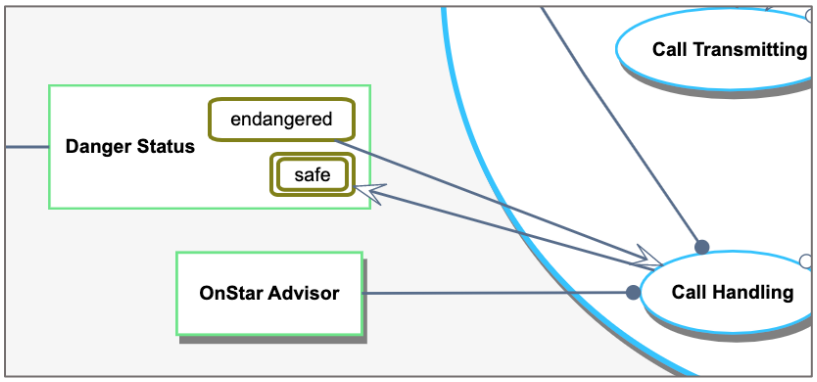


Figure 64. State marked as final

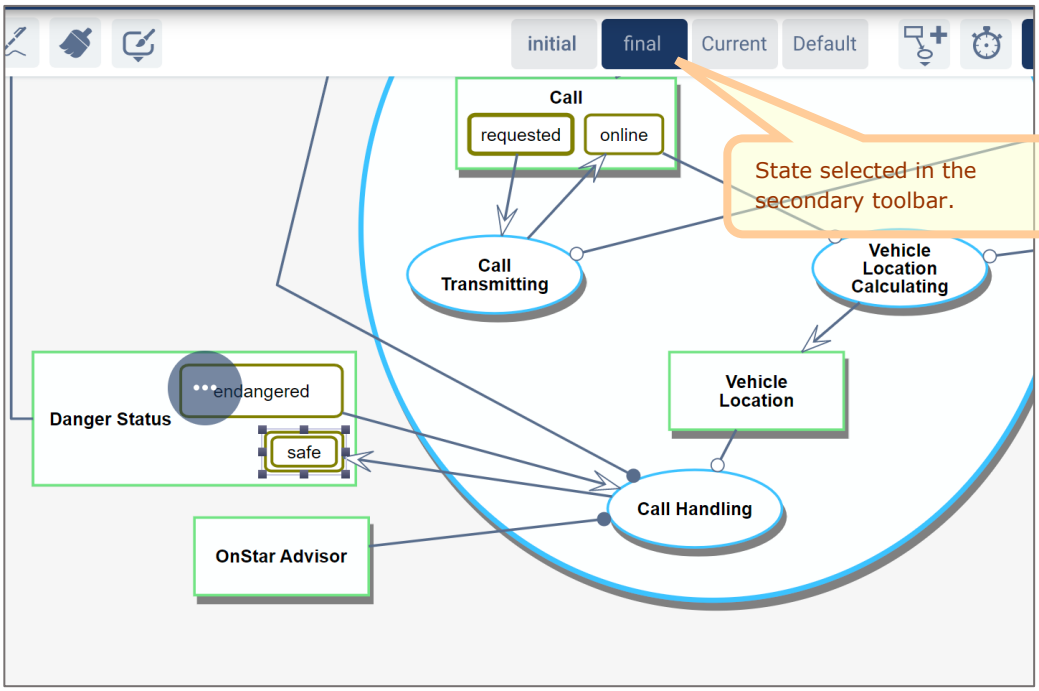


Figure 65. Safe final status selected

Finally, Figure 65, and Figure 66 shows the OPD´s and OPL respectively for the final **Driver Rescuing** SD, and SD1 In-zoomed.

We can remove the un-needed links from the **OnStar System** and **Driver**, as a link connected to the in-zoomed process is connected to all subprocesses. We have also refined the links to the specific subprocesses, so that also makes the higher links un-needed.

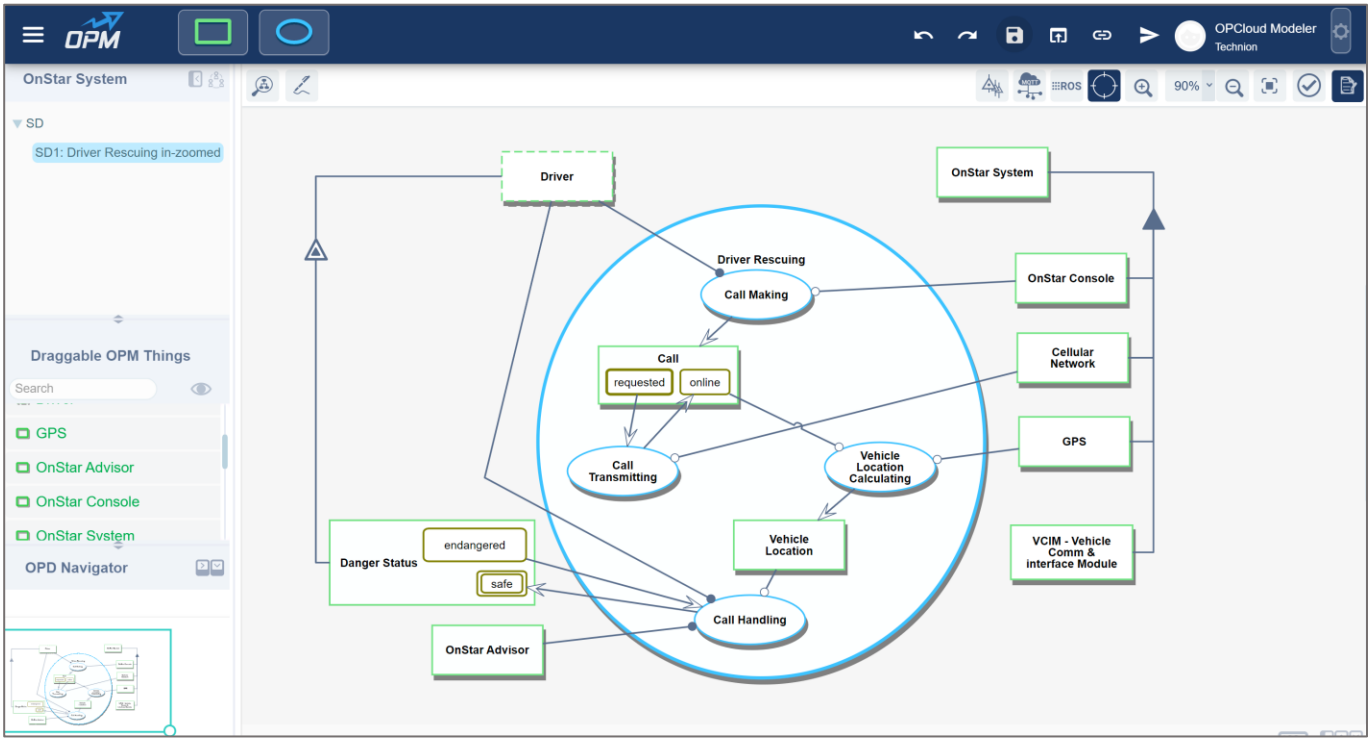


Figure 66. Final Driver Rescuing OPD's, with SD and In-zoomed SD1

OPL spec.
Driver is physical and environmental.
OnStar System is physical and systemic.
GPS is physical and systemic.
Cellular Network is physical and systemic.
VCIM - Vehicle Comm & interface Module is physical and systemic.
OnStar Console is physical and systemic.
OnStar Advisor is physical and systemic.
OnStar System consists of **Cellular Network**, **GPS**, **OnStar Console** and **VCIM - Vehicle Comm & interface Module**.
Driver communicates via **OnStar Console**.
Driver Rescuing is physical and systemic.
OnStar Advisor handles **Driver Rescuing**.
Driver Rescuing requires **OnStar System**.
Driver Rescuing affects **Driver**.
Driver Rescuing from SD zooms in SD1 into **Call Making**, **Vehicle Location Calculating**, **Call Transmitting**, and **Call Handling**, as well as **Call** and **Vehicle Location**.
Driver is physical and environmental.
OnStar System is physical and systemic.
Cellular Network is physical and systemic.
GPS is physical and systemic.
VCIM - Vehicle Comm & interface Module is physical and systemic.
OnStar Console is physical and systemic.
OnStar Advisor is physical and systemic.
Danger Status of **Driver** is informatical and systemic.

Danger Status of **Driver** can be **endangered** or **safe**.
State **safe** is final.
Call is physical and systemic.
Call can be **requested** or **online**.
State **requested** is initial.
Vehicle Location is physical and systemic.
OnStar System consists of **Cellular Network**, **GPS**, **OnStar Console** and **VCIM - Vehicle Comm & interface Module**.
Driver exhibits **Danger Status**.
Driver Rescuing is physical and systemic.
Call Making is physical and systemic.
Driver handles **Call Making**.
Call Making requires **OnStar Console**.
Call Making yields **Call**.
Call Transmitting is physical and systemic.
Call Transmitting changes **Call** from **requested** to **online**.
Call Transmitting requires **Cellular Network**.
Vehicle Location Calculating is physical and systemic.
Vehicle Location Calculating requires **GPS** and **Call** at state **online**.
Vehicle Location Calculating yields **Vehicle Location**.
Call Handling is physical and systemic.
Call Handling changes **Danger Status** of **Driver** from **endangered** to **safe**.
Driver and **OnStar Advisor** handle **Call Handling**.
Call Handling requires **Vehicle Location**.

8. In-zooming into the next level

At this stage, when our **Driver** is finally safe, we can drill further into the details of each of the subprocesses of **Driver Rescuing**.

When zooming into a subprocess details, we must remember that each child OPD refines thing in its father OPD. For instance, **Call Making** is uses **OnStar Console** and requires **Driver**, and it yields a **Call**. Therefore, at the next OPD level down, we can only describe the different refinees (parts, attributes, specializations, or instances) of the **OnStar Console**. If we found that something is missing, then it should be added to both the current OPD and its father OPD, as demonstrated above with the **OnStar Advisor**. If you go back to the requirements described at the beginning of this guide you will see that there is a possibility of automatic **Call Making** by using a "small panel located in the rearview mirror". If this panel is part of the **OnStar Console,** it will be described at the next level. If not, it should be added to SD1 and marked as instrument to **Call Making**.

It may be also a good time to revisit SD, the top-level OPD, and decide whether we need this amount of details at the top level. We may decide for example that the parts of the **OnStar System** will be specified only at lower levels.

9. Summary

This guide presents the basic concepts of OPM as both a language and a MBSE methodology, using OPCloud. You are more than welcome to complete the design of the OnStar System according to the requirements stated at the beginning of this guide.

10. References

Dori, D. (2016). *Model-based systems engineering with OPM and SysML* (pp. 1-411). New York: Springer.

Dori, D. (2016). Overview of ISO 19450. In *Model-Based Systems Engineering with OPM and SysML* (pp. 375-386). Springer, New York, NY.

ISO. (2015). ISO/PAS 19450: 2015—Automation systems and integration—Object-Process Methodology.

Shiffrin, R. M., & Nosofsky, R. M. (1994). Seven plus or minus two: a commentary on capacity limitations.

11. *Annex A*

OnStar System Specification

Being in a car accident, you push a button on a console and are instantly connected with an OnStar advisor. The advisor can pinpoint your exact location and relay your problem to emergency services. If you're in an accident, your car can "tell" OnStar without you having to do a thing.

Source: http://auto.howstuffworks.com/onstar.htm/printable

Cars equipped with OnStar have a small panel located in the rearview mirror, the dashboard or an overhead console, depending on the model. The blue OnStar button allows you to contact a live or virtual advisor. The red button with the cross on it is for emergencies, and the phone or "white dot" button allows you to make phone calls just as if you were using a cell phone.

At its most basic, OnStar consists of four different systems: cellular phone, voice recognition, GPS and vehicle telemetry. All of the services that OnStar provides are a result of one or more of these systems working together, making it a ""System of Systems".

OnStar's cellular service is voice-activated and hands-free. The console contains a built-in microphone and uses the car speakers. To make a call, you speak a phone number, or a previously stored name associated with a phone number. The console is connected to a Vehicle Comm and Interface Module (VCIM), which uses a cellular antenna on top of the car to transmit signals to OnStar's cellular network.

For calls to the advisor, OnStar uses voice recognition software. OnStar can "surf the Web" using the Virtual Advisor automated system. For this service, OnStar uses text-to-voice technology called VoiceXML. When you ask for information, such as "weather," the software translates your request into XML (Extensible Markup Language) and matches it to settings in your OnStar profile. Then it translates the information into VoiceXML and reads it to you.

The GPS receiver is called OnCore, and it is part of the VCIM. A GPS receiver in the vehicle picks up signals form GPS satellites and calculates the location of the vehicle. The OnStar Call Center uses four different satellites to pinpoint the car's location when either the driver or the car itself asks to be located. That location is stored in the vehicle's OnStar hardware. The GPS uses the amount of time that it takes for a radio signal to get from satellites to a specific location to calculate distance.

When the driver pushes the blue OnStar button or red emergency button or an airbag deploys, OnStar places an embedded cellular call the OnStar Center with the vehicle location data. An OnStar advisor handles the call.

To give a vehicle the ability to call when it is in an accident, OnStar uses an event data recorder (also known as a crash data recorder). GM calls the entire process the Advanced Automatic Crash Notification System (AACN). The AACN system comprises four components: sensors, the Sensing Diagnostic Module (which includes the event data recorder), the VCIM and the cellular antenna. When the car is in a crash, sensors transmit

information to the Sensing Diagnostic Module (SDM). The SDM also includes an accelerometer, which measures the severity of the crash based on gravitational force. The SDM sends this information to the VCIM, which uses the cellular antenna to send a message to the OnStar Call Center. When an advisor receives the call, he uses the GPS to find the vehicle's location and calls the car to check with the driver. Even if there's not a measurable impact, the VCIM also sends a message when the air bag goes off, prompting the advisor to call the car's driver.