Chapter 1

OPCAT – An Object-Process CASE Tool for OPM-Based Conceptual Modelling

D. Dori, C. Linchevski, and R. Manor

Abstract

OPCAT is an advanced, commericially-available Object-Process Methodology (OPM)-based software platform for conceptual modelling of complex systems in virtually any domain. OPCAT is used to teach OPM in leading institutions of higher education and is applied in a variety of industrial and scientific domains, from banking through defense to molecular biology.

OPCAT 3 is a product that supports stand-alone OPM-based system development, evolution, lifecycle engineering, and lifecycle management. OPCAT implements the bimodal graphics-and-text expression of the OPM model along with many unique model-based capabilities, including requirements engineering, information layers management, model repository, simulation and validation of the model, and automatic document generation and management. This paper briefly introduces OPM and the basic features of OPCAT 3 by following a running example that is constructed step-by-step, so the interested reader and user can practice it while reading the paper. Along the way, principles of OPM and how they are implemented in OPCAT are explained.

1.1. Introduction to Object-Process Methodology

Object Process Methodology (OPM) is a holistic approach for conceptual modelling 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.

In complex system, the two main ***system aspects***—***structure*** and ***behavior***—are highly intertwined and hard to separate. ***Object-Process Methodology***, OPM (Dori 2002) is ***a holistic approach to the study and development of both natural and man-made complex systems***. Recognized as one of ***INCOSE's six model-based systems engineering methodologies*** (Estephan 2008), OPM is currently undergoing a process of ***adoption*** by ***ISO*** as an ***International Standard***. The PI is co-convenor of the ***OPM Study Group*** (ISO 2009). The recent milestone in this process was the unanumous adoption of the WG interim report at the general assembly of TC 184/SC5 in Tokyo in March 25, 2010, which instrcuted the WG to prepare a Darft International Standard where OPM would be the basis for model-based enterprise standards.

Based on formal mathematical foundations of graph grammars, ***OPM caters to human intuition*** via ***graphics*** and ***natural language text***. Requiring that a ***single model represents structure*** and ***behavior***, OPM is founded upon two elementary building blocks—***objects*** and ***processes***. ***Objects*** are the (physical or informatical) components that ***comprise the system***.

Processes transform objects by ***creating*** them, ***consuming*** them, or ***changing their states***. The concurrent representation of structure and behavior in the same diagram type is balanced, creating synergy whereby each aspect helps understanding the other.

The elements of OPM are three entities and two types of links. Entities are stateful objects, processes, and states. An ***object*** is a thing that ***exists***, possibly in some state, while a ***process*** is a thing that ***happens*** to an object and transforms it. OPM entities are connected via links, which can be structural or procedural.

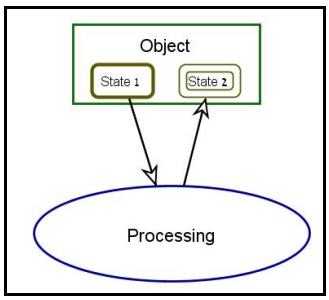


Figure 1: An Object-Process Diagram (OPD) showing the three OPM entities: Object, Process, and State, and the input/output procedural link pair, which expresses that Processing changes Object from State 1 to State 2.

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. These entities are shown in Figure 1. Objects are the (physical or informatical) things in the system that exist, and if they are stateful (i.e., have states), then at any point in time 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 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 three ways: (1) a processes 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; and (3) an object can trigger an event that invokes a process if some conditions are met. Accordingly, a procedural link can be a transformation link, an enabling link, or an event link. A transformation link expresses object transformation, i.e.,

object consumption, generation, or state change. Figure 1 shows a pair of transformation links, the input/output link. It expresses in OPL that Processing changes Object from State 1 to State 2. An enabling (agent or instrument) link expresses the need for a (possibly state-specified) object to be present in order for the enabled process to occur. The enabled process does not transform the enabling object. An event link connects a triggering entity (object, process, or state) with a process that it invokes.

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 refineable (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 modelling a complex system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase that is generated automatically by OPCAT 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 modelling 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 OPCAT 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.

Structural links express a ***relation between pairs of entities*** of the same persistence, i.e., between two objects or between two processes. The four ***fundamental structural relations*** are ***aggregation-participation*** (whole-part), ***generalization-specialization*** (inheritance), ***exhibition-characterization*** (attributes and operations), and ***classification-instantiation*** (class and its members). ***Procedural links***, which connect an object or its state to a process, can be

transforming links (generation, consumption, or effect) or enabling links (agent or instrument), as explained below.

Two ***semantically equivalent modalities***, ***graphic*** and ***textual***, describe each OPM model. The top-right pane of **Error! Reference source not found.** shows an ***Object-Process Diagram*** (OPD), OPM's graphic modality. The bottom pane shows the textual modality—the corresponding ***Object-Process Language*** (OPL) paragraph, the auto-generated OPD counterpart. OPL is a ***subset of English*** which domain experts (e.g., biologists) readily understand. Each element in an OPD has a graphical symbol. An object is a box and a process—an ellipse.

OPM Basics: A Quick Summary

- OPM is a modelling 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.

1.2. The Evolution of OPCAT

Started in 2000 as a students' project at the Technion, OPCAT (Dori et al. 2003) has evolved to be an advanced, commericially available software platform for conceptual modelling of complex systems that is based on Object-Process Methodology (OPM). . OPCAT is taught in leading institutions of higher education and applied in a variety of industrial and scientific domains, from banking through defense to molecular biology (Dori & Choder 2007).

The company that supports the product is also coalled OPCAT and its Web site is www.opcat.com. The site provides links to the OPM book (Dori 2002) and to many papers on OPM, as well as download of academic and trial edition.

1.3. Starting to Model with OPCAT

The OnStar System, specified in Figure 2 in free text, is used as a running example throughout the paper.

OnStar System Specification

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

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.

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.

Figure 2: The OnStar System specification

1.3.1. Opening a new system

Assuming you have Java JRE 6.2 and OPCAT installed[1], a detailed, step-by-step OPM-based model construction of the OnStar System is used to explain how to do the modelling with OPCAT by describing almost each mouse-click, so you, as a new user, can closely follow each step and reproduce the model as you proceed with following the model while it is being constructed in the following pages.

Double clicking the green OPCAT icon application, a blank screen is opened, as shown in Figure 3. Clicking **System -> New** we get the **New System Properties** dialog box shown in Figure 4. Fill in the details and click **OK**. At this point you may also want to save the system in a designated folder. Next we start modelling the system in the *System Diagram*.

[1] OPCAT 3.1 can be downloaded from http://dl.getdropbox.com/u/2083290/opcat-3.1-limited-installer.exe

1.3.2. The System Diagram (SD)

The ***System Diagram*** (SD) is the first, top-level Object-process Diagram (OPD) of our system. SD expresses clearly the main system's function – what it is designed to do, the beneficiary, the main objects that are consumed, created, and/or affected by that function, the main human and non-human enablers of the system, and the objects which interact with our system (called environmental, as opposed to systemic).

1.3.3. Defining the system main function

We now start to actually model the OnStar system. We do this by drawing model elements – processes, objects, and links – on the graphic screen, the top right pane marked as *SD*, short for System Diagram. As we model, we inspect the translation in real-time 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 modelling a new system is to **define the system's major function**. This is going to be the central process which we will draw in the center of the SD. The major function of the OnStar system is **Driver Rescuing**.

Sliding the mouse over the OPM symbols in the bottom toolbox shows their name. In Figure 5 the mouse is over the ellipse and we see that the ellipse is the symbol for a process. Click on the process icon (the ellipse, third from the bottom left toolbar, see Figure 3). You will get the dialog window shown in Figure 6. Type the process name. Since this process is physical, click on the **physical essence** radio button and click OK.

You should get the process as shown in Figure 7. Note that the process is shaded to denote that its **essence** attribute is **physical** (the default essence of OPM things – processes as well as objects – is **informatical**).

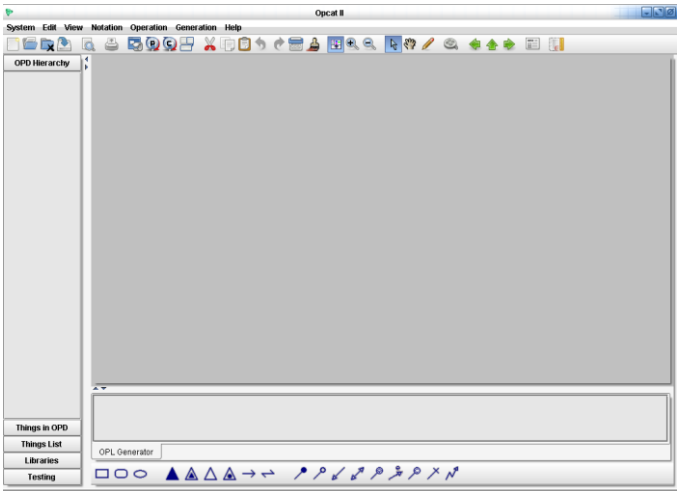


Figure 3. A blank OPCAT opening screen

Note also that the following OPL sentence appeared in the bottom right pane:

Driver Rescuing is physical**.**

To help distinguishing between objects and processes, the process name in the text is in blue, as it is in the OPD. Objects will be green.

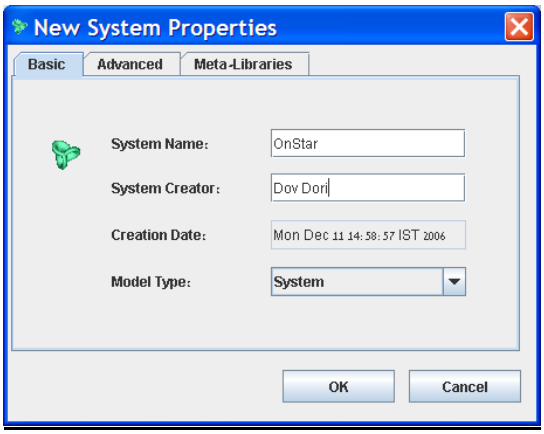


Figure 4. The New System Properties dialog box

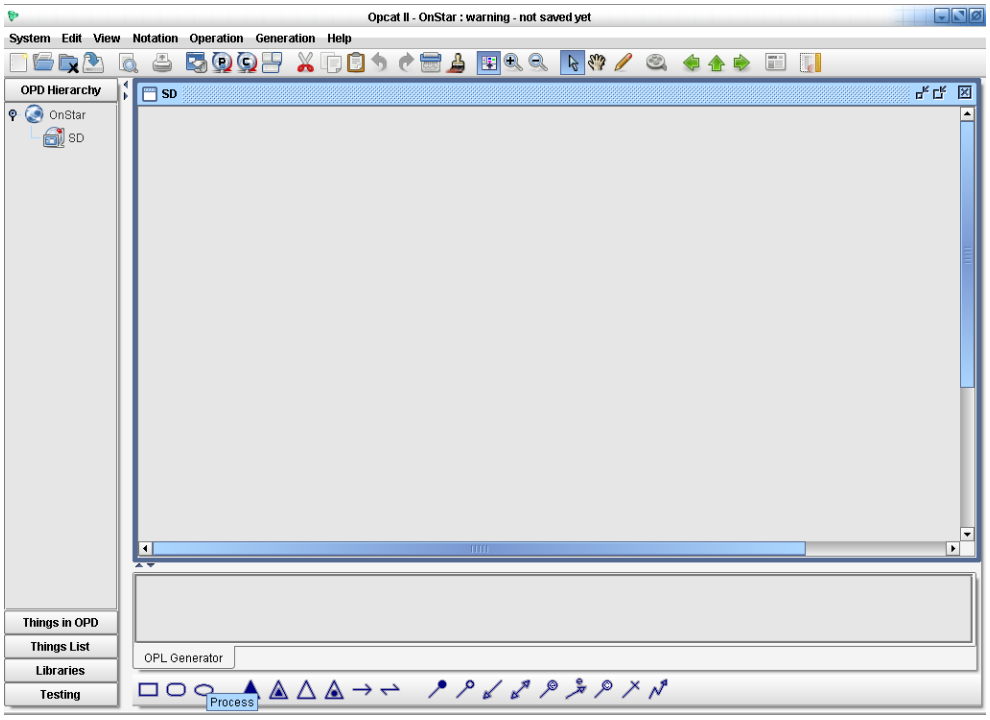


Figure 5. OPCAT Screen after opening the new OnStar System model

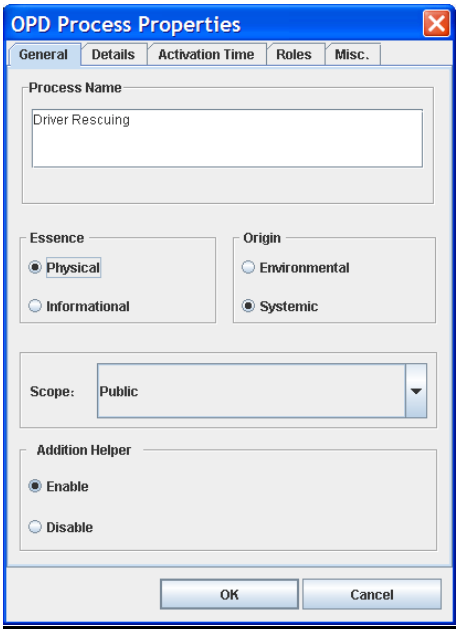


Figure 6. The Process Properties dialog box

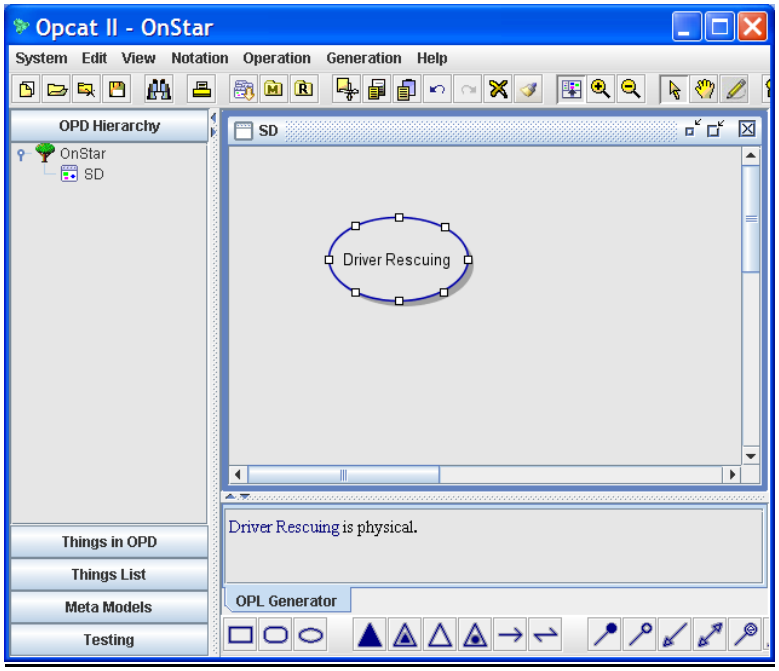


Figure 7. Driver Rescuing, the main function of the system, has been inserted as a physical process.

1.3.4. Defining and linking the system's main objects

Having defined the major system function, we now turn to depicting the main objects in the system. We need to think first who is the beneficiary of the system, i.e., who gets value from the systems. Obviously, it is the driver, so we insert **Driver** as a physical object. Since the driver is not part of the system, but interacts with it, its Affiliation is *environment* rather than *systemic*, so we mark both radio buttons and get the OPL sentence:

Driver is environmental and physical**.**

Another vital object in the model is the **OnStar System**, which we also add as a physical object. Figure 8 shows SD, the System Diagram, after the objects **Driver** and **OnStar System** were added.

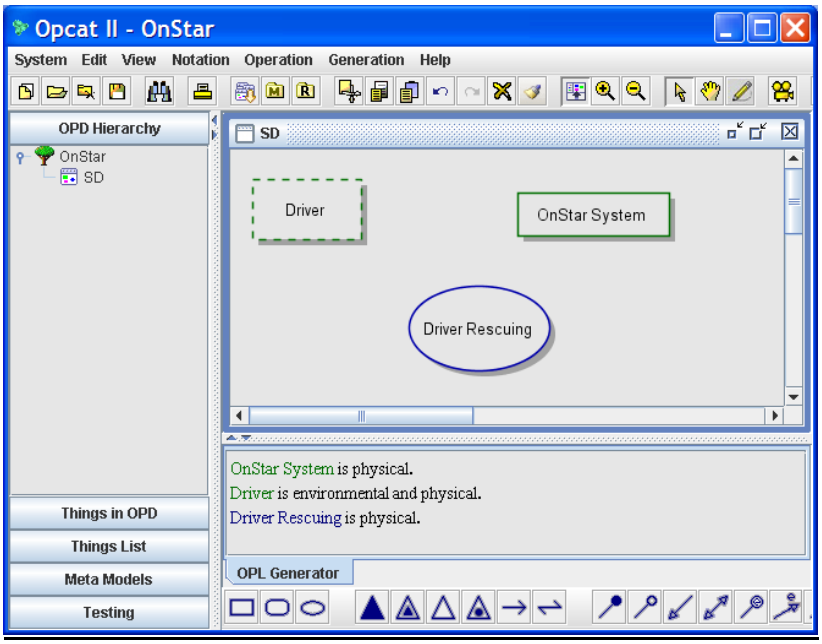


Figure 8. SD after adding the two objects

As the beneficiary, **Driver** is affected by the **Driver Rescuing** process. To denote this, click on the effect link symbol – the bidirectional arrow at the bottom right, then click on the process and drag it from the process to the object. The result, shown in Figure 9, is expressed in a new OPL sentence:

Driver Rescuing affects **Driver.**

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** process in more details. In this sense Effect link is a generalization of the detailed description to come.

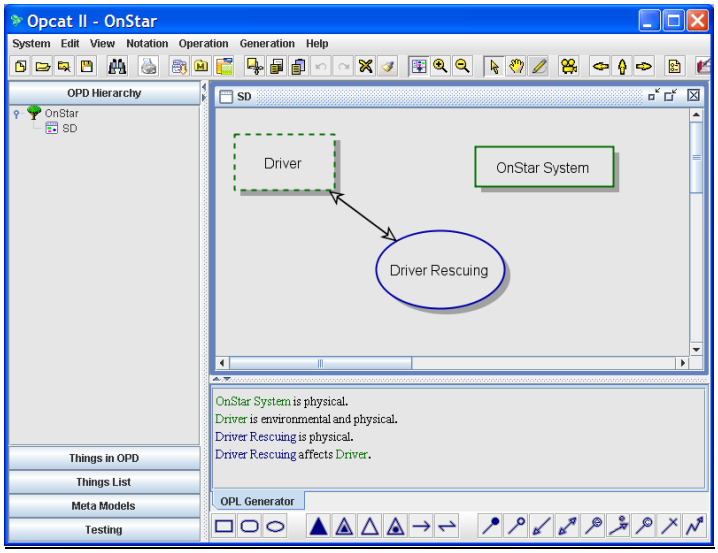


Figure 9. SD after linking **Driver** and **Driver Rescuing** with an effect link

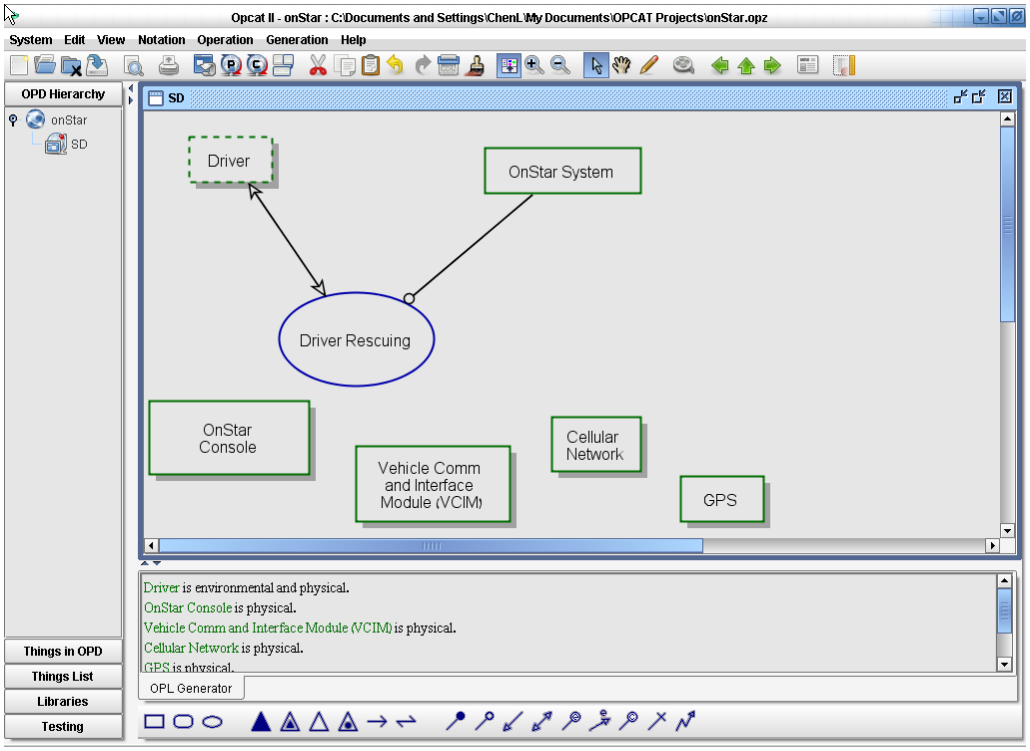


Figure 10. Adding the four parts of OnStar System

Next, in Figure 10, the **OnStar System** is added as an instrument to the **Driver Rescuing** process. This is done by clicking on the instrument link button – the line with the white circle (white lollipop) – and dragging it from the **OnStar System** – the instrument – to the **Driver Rescuing** process. The sentence added as a result is:

Driver Rescuing requires **OnStar System.**

Reading the OnStar description we find that " At its most basic, OnStar consists of four different systems: cellular phone, voice recognition, GPS and vehicle telemetry." We now wish to model this sentence, which describes the parts of the **OnStar System**. Adding the four parts of **OnStar System** is shown in Figure 10. The parts are added one by one. The location of each object is determined by where we click on the screen after selecting the Object button.

Naturally, the objects are therefore not quite aligned, and we may wish to align them. Marking the four parts of **OnStar System** for horizontal alignment is shown in Figure 11. We mark all at once by clicking and dragging the mouse to include all the things we wish to mark within the dotted area. The marking is shown by small white squares on each object.

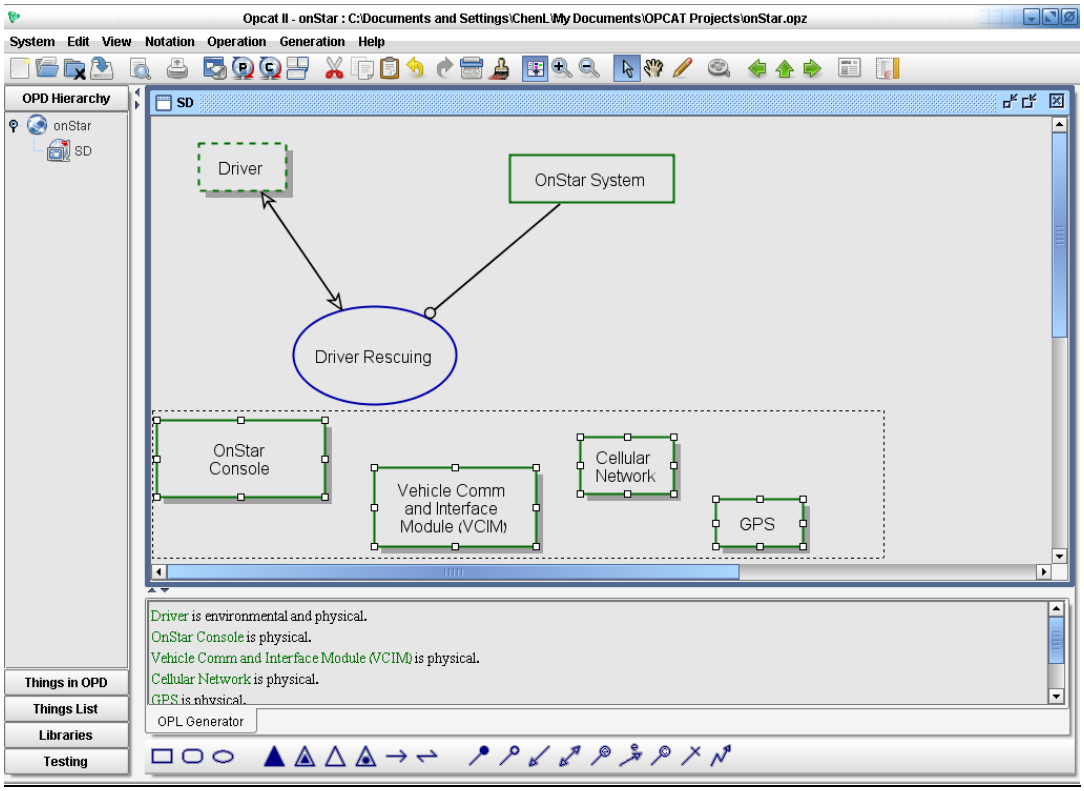


Figure 11. Marking the four parts of OnStar System for horizontal alignment

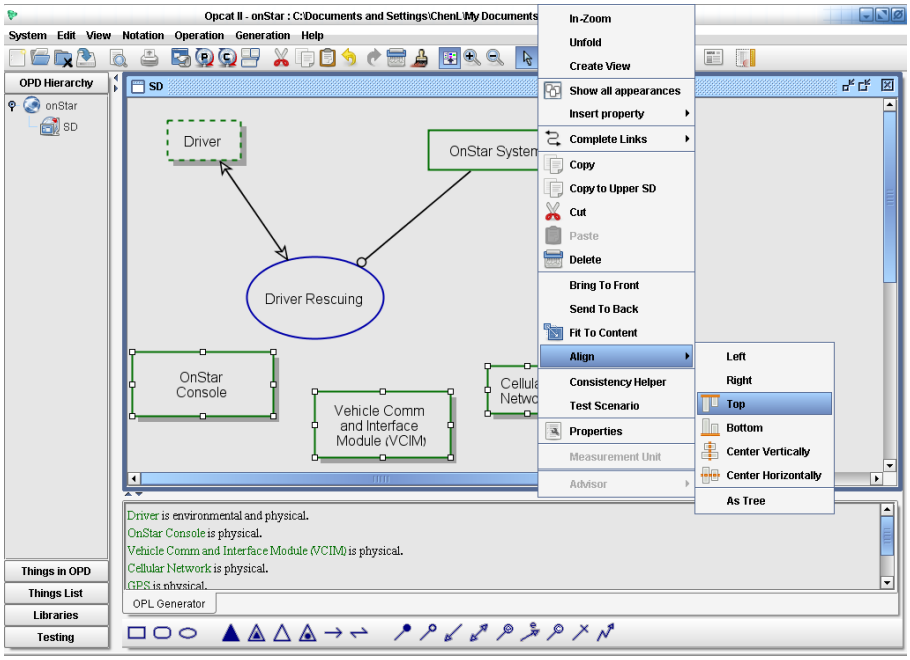


Figure 12. Top alignment of the four parts of OnStar System

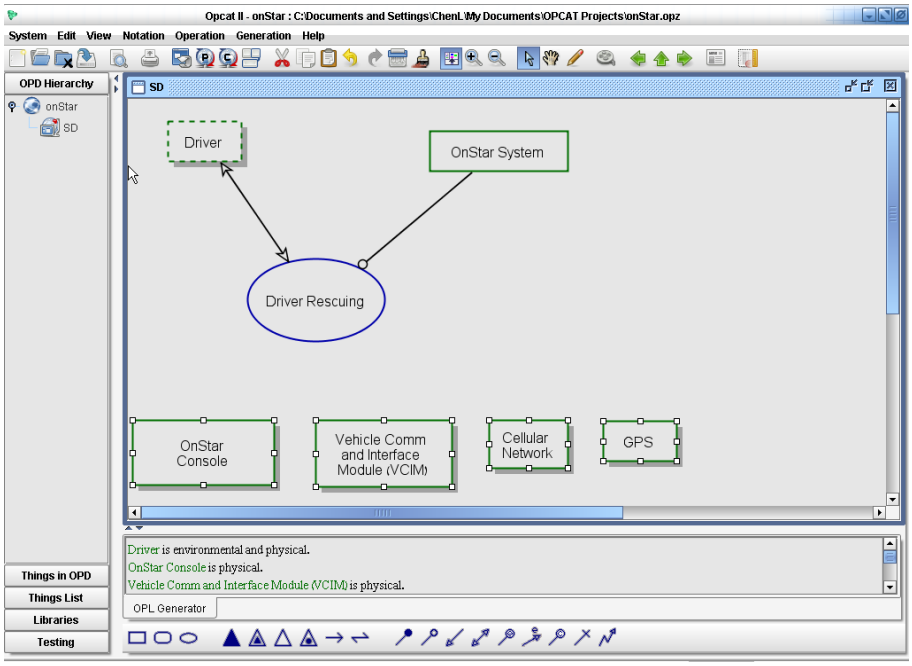


Figure 13. The four parts of OnStar System after top alignment

Right Clicking on one of the four marked objects brings the pop-up menu shown in Figure 12, and clicking "Align" brings a sub-menu, from which we select "Top". The result of the aligned objects is shown in Figure 13. At this point, we wish to denote the fact that all these four objects are part of the OnStar System. This is an opportunity to introduce the set of OPM symbols which appear at the bottom of the screen.

OPCAT Basics: A Quick Summary

- The first OPD (the **SD**) is used to establish the **main function** of the system and the objects involved in this process.
- We start off by modelling the **main function** of the system as the **central process** in SD.
- Things (Objects or Processes) have an **Essence** attribute and an **Affiliation** attribute.
- The default Essence is **informatical** and the default Affiliation is **systemic**.
- Things which are not informatical (such as file, command, message, algorithm) will be marked as physical.
- 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** on their design.
- The system can affect environmental objects but it is not responsible for creating them.
- 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, or features) of the Things in the SD.

1.4. The OPM symbols

Symbol		Name: Definition	OPL	Semantics/ Effect on the system flow/ Comments
Things		Object **A**: A thing that exists	**A** is physical [and environmental].	**A** is informatical and systemic by default.
		Process **B**: A thing that transforms (generates, consumes, or changes the state of an) object.	**B** is physical [and environmental].	**B** is informatical and systemic by default.
		State: A situation of an object.	**A** is s1. **A** can be s1 or s2. **A** can be s1, s2, or s3.	Always within an object.

Figure 14. The OPM entities

The OPM symbol buttons at the bottom of the screen are divided into three groups.

1.4.1. Entities:  Object, Process, and State

The group on the left at the bottom of the screen is the group of the entities: Object, state, and process. Figure 14 shows the OPM entities with their symbols, names and defintions, OPL snetences and semantics.

1.4.2. Structural Links: 

At the bottom of the screen in the middle is the group of the structural links: four triangles and two arrows with open heads. The structural links denote static, long-term relations that are not linked to the time dimension and hold true between an object and other objects or between a process and other processes for some time during the system's existence. These are described in Figure 15.

Symbol		Name: Definition	OPL	Allowed Source-to-Destination connections	Semantics/ Effect on the system flow/ Comments
fundamental	▲	Aggregation-Participation	**A** consist of **B**.	Object-Object Process- Process	Whole -Part
	△	Exhibition-Characterization	**A** exhibits **B**.	Object-Object Object-Process Process-Object Process- Process	Relation between a thing and its feature—an attribute (object) or operation (process)
	△	Generalization-Specialization	**B** is an **A**. (objects) **B** is **A**. (processes)	Object-Object Process- Process	Gives rise to inheritance
	△	Classification-Instantiation	**B** is an instance of **A**.	Object-Object Process- Process	Relation between a class and its instance
	→	Tagged structural links: Unidirectional	According to text added by user	Object-Object Process- Process	Describes structural information with user-defined semantics as the tag
	⇄	Bidirectional			

Figure 15. The OPM structural links

1.4.3. Procedural Links: 

The OPM symbols grouped on the left are the procedural links: four triangles and two arrows with open heads. The procedural links denote dynamic, transient relations that have everything to do with the time dimension. Unlike the structural links, they connect an object and a process but not an object to another object or (with the exception of the invocation link) a process to another process.

Procedural links are divided in two groups: transforming links and control links. Transforming links, presented in Figure 16, have one or two colsed arrows, casue transformation to an object: object generation, consumption, or change of state.

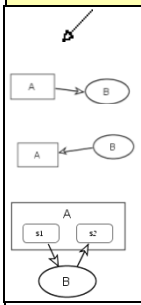
Symbol	Name: Definition	OPL	Allowed Source-to-Destination connections	Semantics/ Effect on the system flow/ Comments
	Consumption Link Result Link Input-Output Link Pair	**B** consumes **A.** **B** yields **A**. Or for state-specified objects: **B** consumes s1 **A.** **B** yields s1 **A**. **B** changes **A** from s1 to s2.	Object to Process Process to Object State to Process Process to State Process – State s1 to Process and Process to s2.	Process consumes the object. Process creates (yields) the object. Process changes the state of object.
	Effect Link	**B** affects **A**.	Object (A) to Process (B)	Used when details of the effect are not necessary or will be add at a lower level (also created when states lined to a process with an input-output pair are hidden-suppressed) Affect at an high level signifies at lower levels – • State changes • Consumption and later generation
	Invocation Link	**B** invokes **C**.	Process-Process	Execution will proceed if the triggering failed (due to failure to fulfil one or more of the conditions in the precondition set).

Figure 16. The transforming procedural links

Control links, shown in Figure 17, have a circular end. They denote enablers—objects required for the process tooccur, events that trigger the process, or conditions under which the process, if triggerred, will occur. There are two enbales: agent—a human enabler in charge or at least involved in the process, and instrument—a non-human enabler which is required for the process to occur. For example, Pilot is the agent of Flying, while Aircraft is instrument for Flying.

The consumption event link is both a control and a transforming link, because if the porcess occurs as a result of the event's process triggerring, the triggerring object is consumed by the triggered process.

Symbol	Name: Definition	OPL	Allowed Source-to-Destination connections	Semantics/ Effect on the system flow/ Comments
	Agent Link	**A** handles **B**.	Object (A) to Process (B)	Denotes a human operator. Activating the link triggers the process B.
	Instrument Link	**B** requires **A**.	Object (A) to Process (B)	Wait until A is generated and exists.
		B requires s1 **A**.	State (s1) to Process (B)	Wait until A is at state s1.
	Condition Link	**B** occurs if **A** exists.	Object (A) to Process (B)	Execute if object A exists, and if not then skip process B and continue the regular system flow.
		B occurs if **A** is s1.	State (s1) to Process (B)	Execute if object A is at state s1, and if not then skip process B and continue the regular system flow.
	Instrument Event Link	**A** triggers **B**.	Object (A) to Process (B)	Execution will proceed if the triggering failed (due to failure to fulfil one or more of the conditions in the precondition set). For normal, non-triggered execution, the object or state linked is not required for the process to take place.
		s1 **A** triggers **B**.	State (s1) to Process (B)	
	Consumption Event Link	**A** triggers **B**, which, if occurs, consumes **A**.	Object (A) to Process (B)	Execution will proceed if the triggering failed (due to failure to fulfil one or more of the conditions in the precondition set). For normal, non-triggered execution, the object or state linked is not required for the process to take place.
		s1 **A** triggers **B**, which, if occurs, consumes **A**.	State (s1) to Process (B)	

Figure 17. The control procedural links

1.5. Using structural relations

Structural relations are used to model the structure of the system. To denote the fact that the four marked objects in Figure 18 are parts of OnStar System we click on the solid (black or blue) triangle, click the mouse on the whole – the OnStar System – and drag it to one of the parts, e.g., Cellular Network. In response, OPCAT asks the question presneted in Figure 19, "Continue with all marked?" Clicking OK will generate the four links in Figure 20. The resulting OPL sentence, shown in the second line of the OPL pane, is:

OnStar System consists of **OnStar Console, Vehicle Comm and Interface Module (VCIM), Cellular Network,** and **GPS.**

In order to specify that Driver communicates via OnStar Console, we click on the unidirectional tagged structural link, the fifth button in the middle, structural links group, and

drag it from Driver to OnStar Console. The result (without the label along the link) is shown in Figure 21.

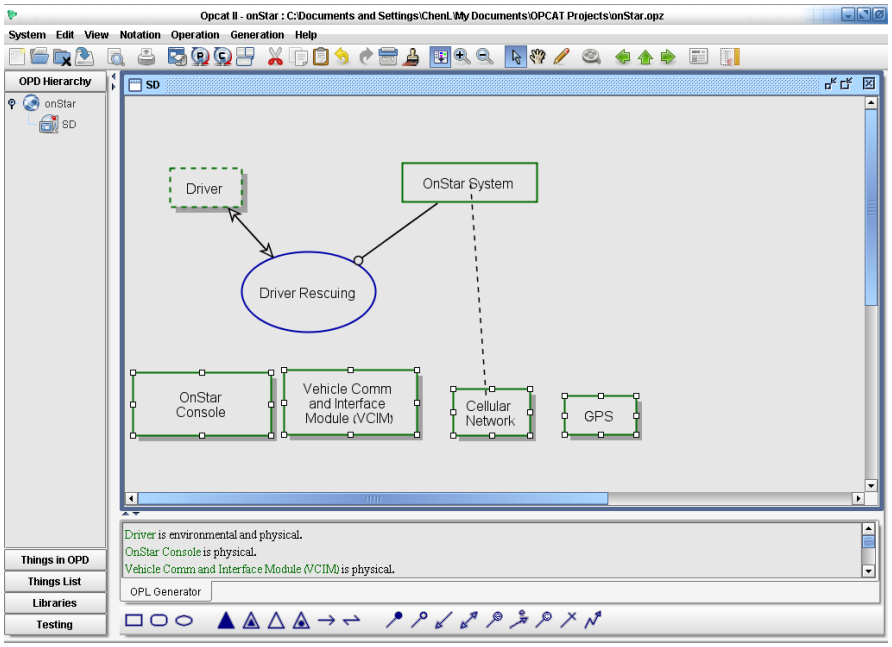


Figure 18. Linking the Cellular Network part of OnStar System to the whole OnStar System

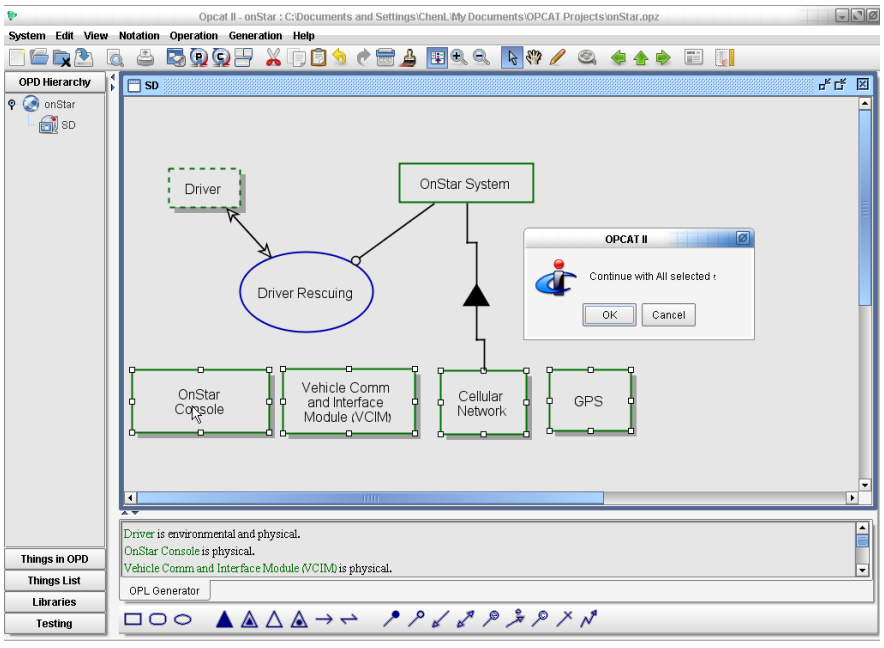


Figure 19. The question asked after linking Cellular Network to the whole OnStar System

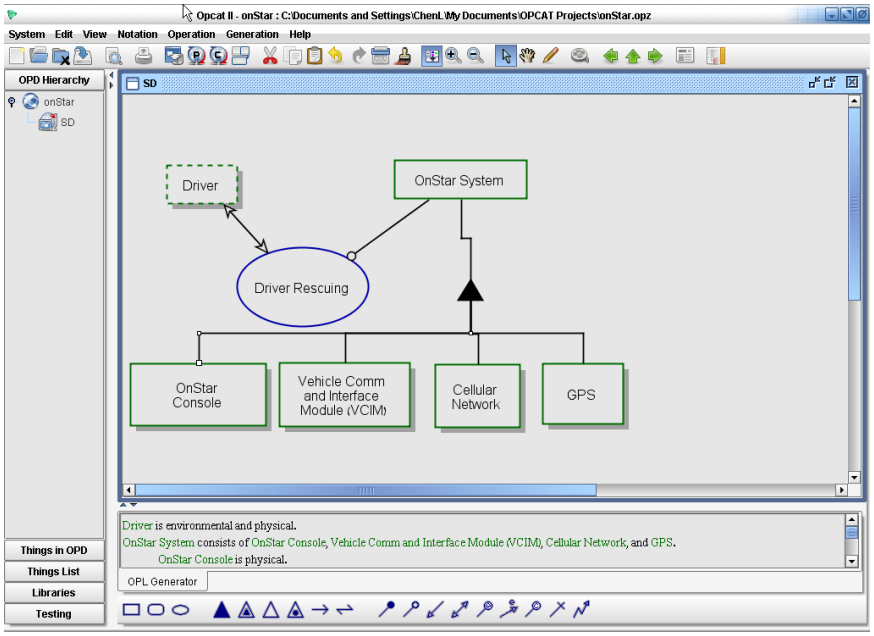


Figure 20. The four parts of Cellular Network are now linked to the whole OnStar System

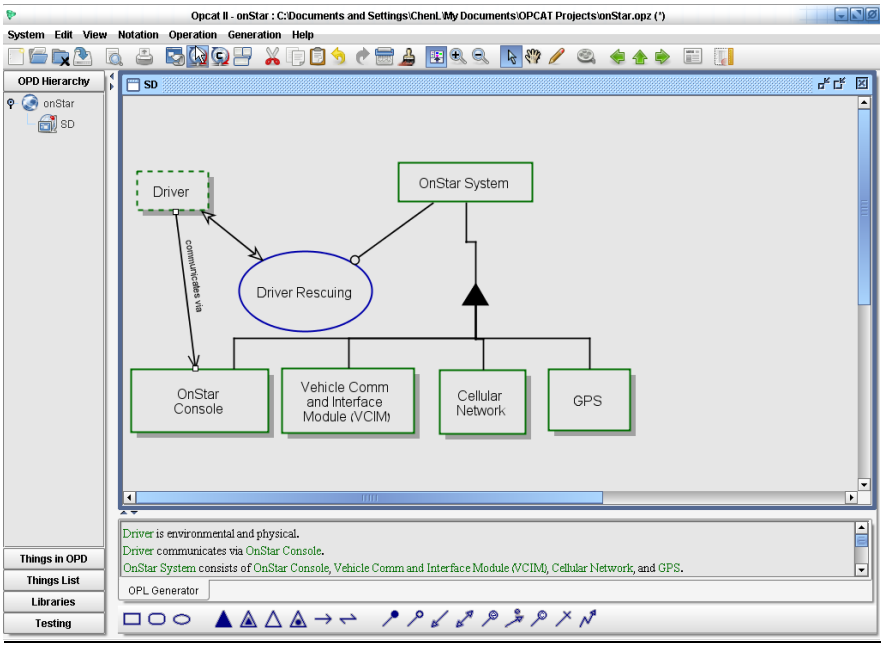


Figure 21. Specifying that Driver communicates via OnStar Console

We type "communicates via" in the Tag box, click OK, and the OPL sentence now changes to:

Driver communicates via OnStar Console.

This OPL sentence appears at the last line in Figure 21. Tagged Structural Links will be used to provide additional information about the system that cannot be expressed by using any of the other structural links.

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

Driver relates to **OnStar Console.**

Since we wish to be more specific, we double click the link we just drew and get the dialog box in Figure 22.

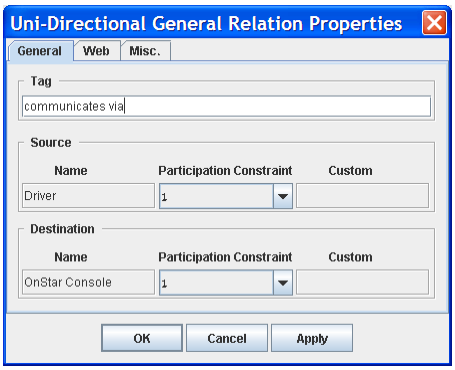


Figure 22. Inserting the tag "communicates via" in the unidirectional general structural relation

1.6. Complexity Management: 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 modelling while maintaining the OPD clear and readable? To the rescue comes the OPM built-in in-zooming mechanism. In-zooming is a refinement mechanism which helps mange the system's complexity. It enables starting a new Object-process Diagram (OPD), in which a thing (process or object) and whatever is linked to it, are copied from the ancestor OPD, and it is blown-up in order to enable specifying its sub processes. Along with the in-zoomed thing, things that were attached to it are brought along so as to maintain the consistency across the entire OPD set.

As we have seen above we first selected the main function of our system, Driver Rescuing, and described the main necessary objects and the main result or effect of this process. Now we will continue to refine the specification of the system using the processes as the skeleton for decomposition.

Following Miller's "magic number" of 5 plus or minus 2, we will continue to refine the system's specification by adding between 3 to 6 nested subprocesses, which, in turn can be further in-zoomed into new 3 to 6 subprocesses. We continue this until we are satisfied with the level of detail of t he system. SD1 should hence describe the 3 to 6 main subprocesses comprising the Driver Rescuing process by the OnStar system.

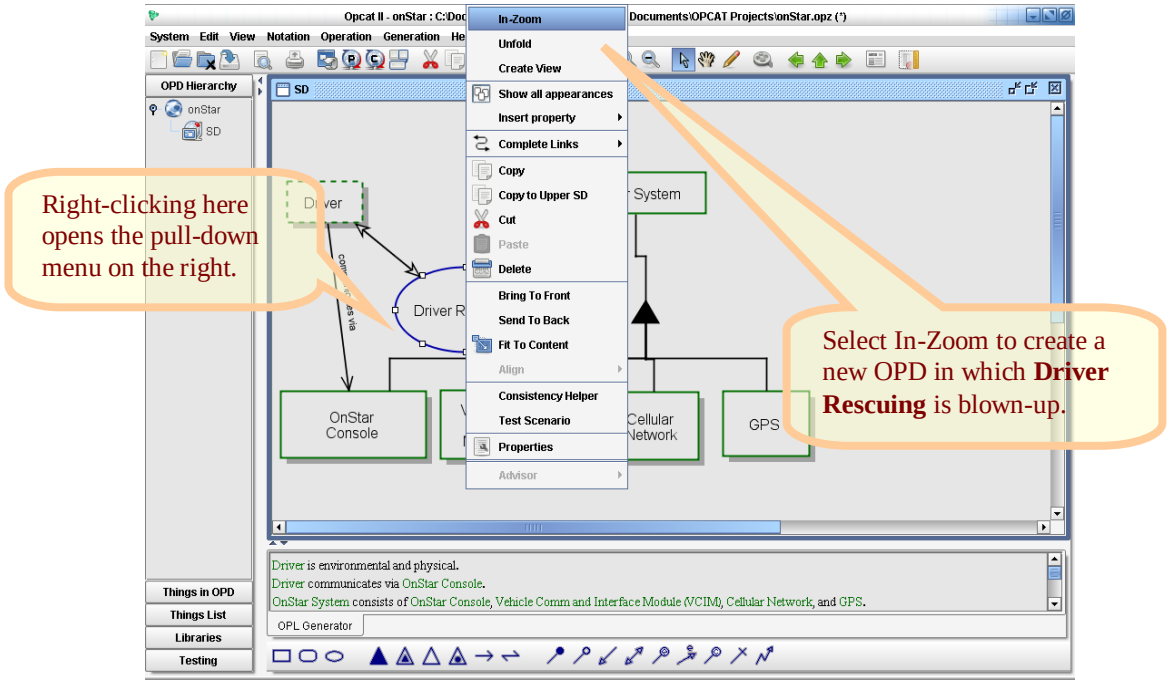


Figure 23. Zooming into the Driver Rescuing process

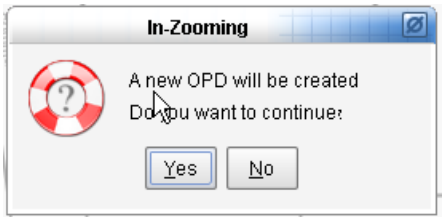


Figure 24. Zooming into the Driver Rescuing process

For each process we will describe the preconditions and the result or effect. We will do this by adding objects or states to objects, to the level of detail required to describe the preconditions and results of the nested subprocesses. More details about these additional objects can possibly be added when we zoom into the sub-subprocesses of each subprocess.

OPM uses detail-based refinement, i.e., lower level OPDs provide detailed descriptions of upper levels ones. Consistency must be maintained between any two OPDs related by an ancestor-descendant relation. To make a call in the OnStar system, you say out-loud a phone number or a previously stored name associated with a phone number. Modelling this statement presents us with an opportunity to recommend an effective OPCAT design methodology. The first subprocess of **Driver Rescuing** is **Call Making**. In Figure 26 we add this subprocess within and at the top of the enclosing **Driver Rescuing** process.

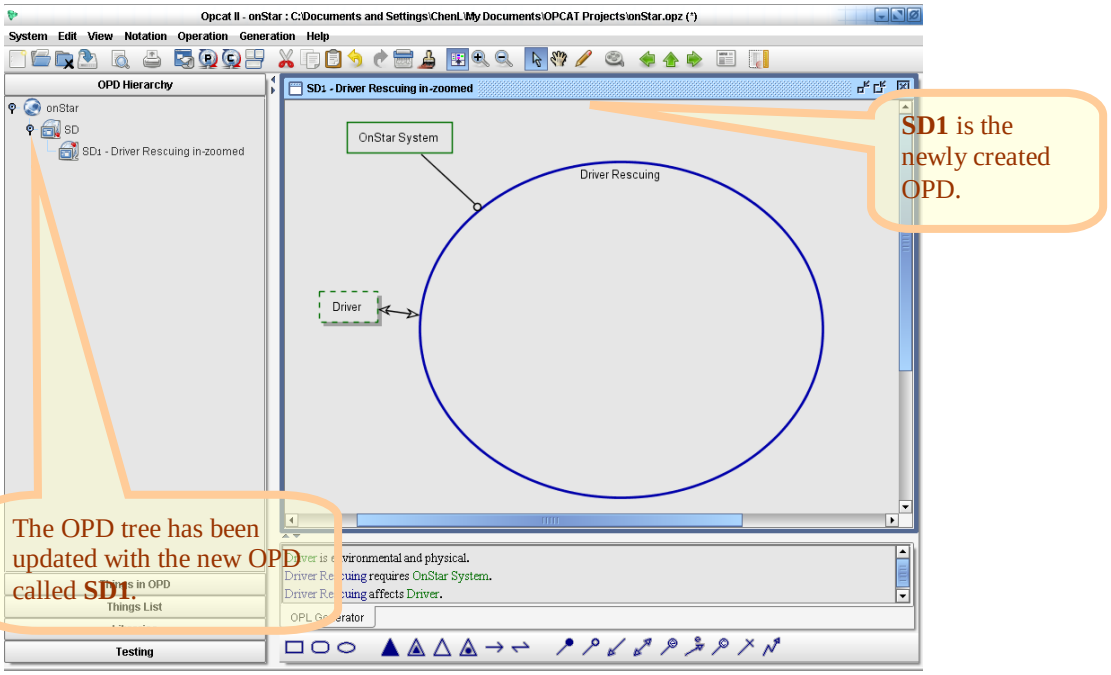


Figure 25. The new OPD titled SD1 – Driver Rescuing in-zoomed

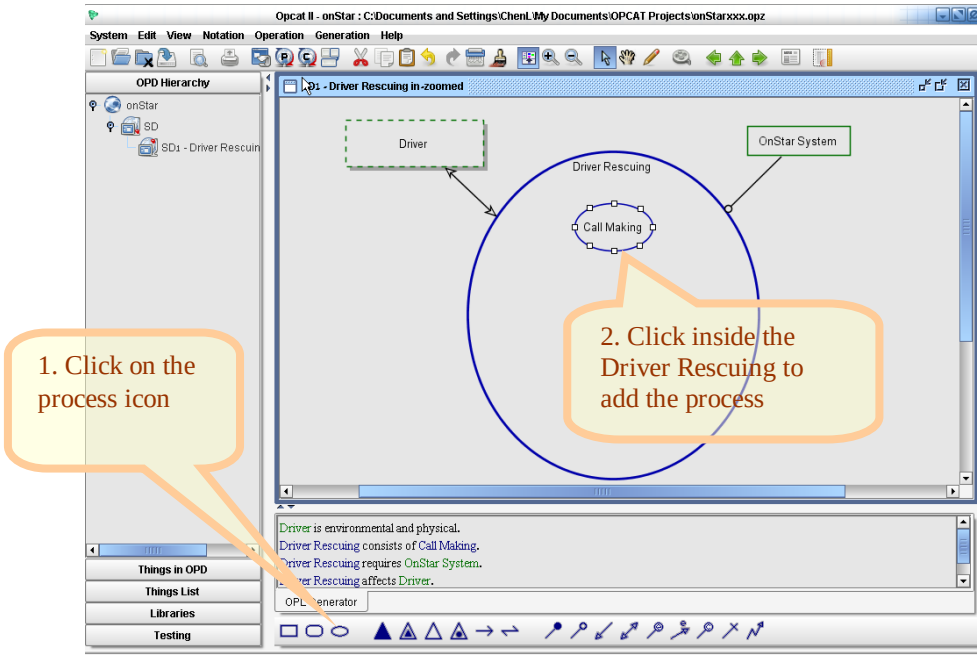


Figure 26. Call Making is inserted as the first subprocess within the in-zoomed Driver Rescuing process.

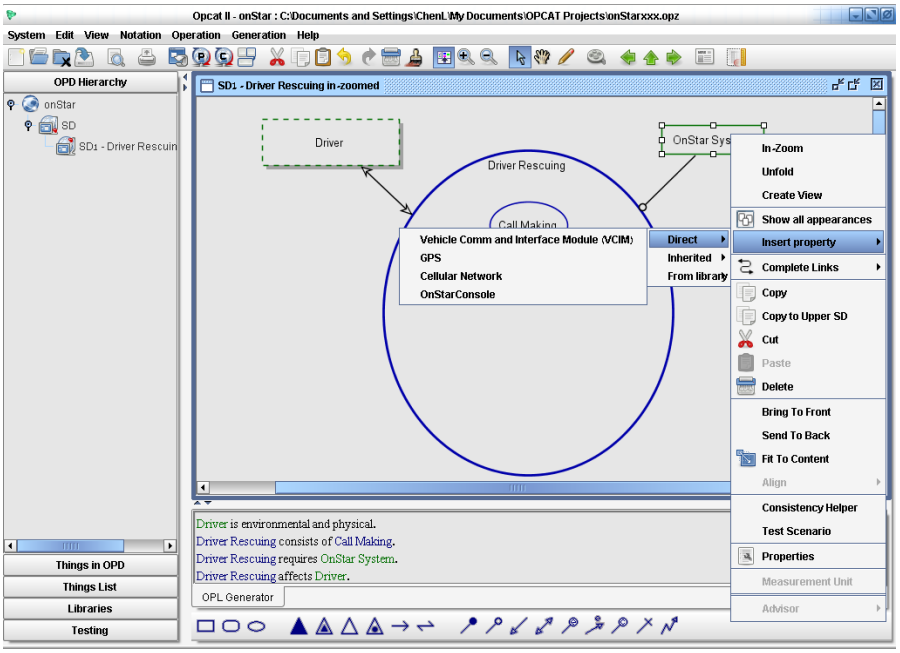


Figure 27. Adding OnStar Console by using the Insert Property feature

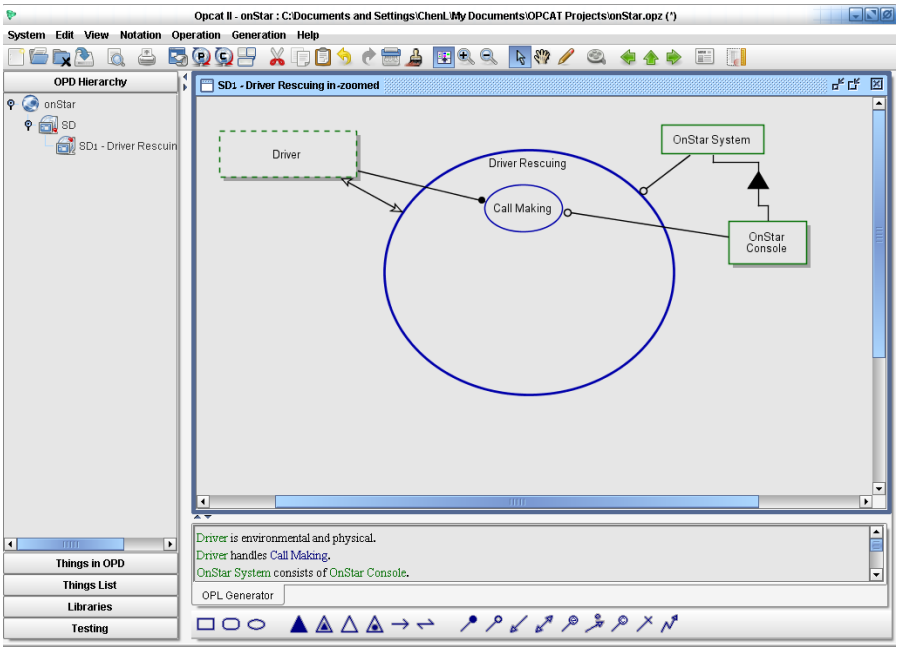


Figure 28. Connecting Call Making

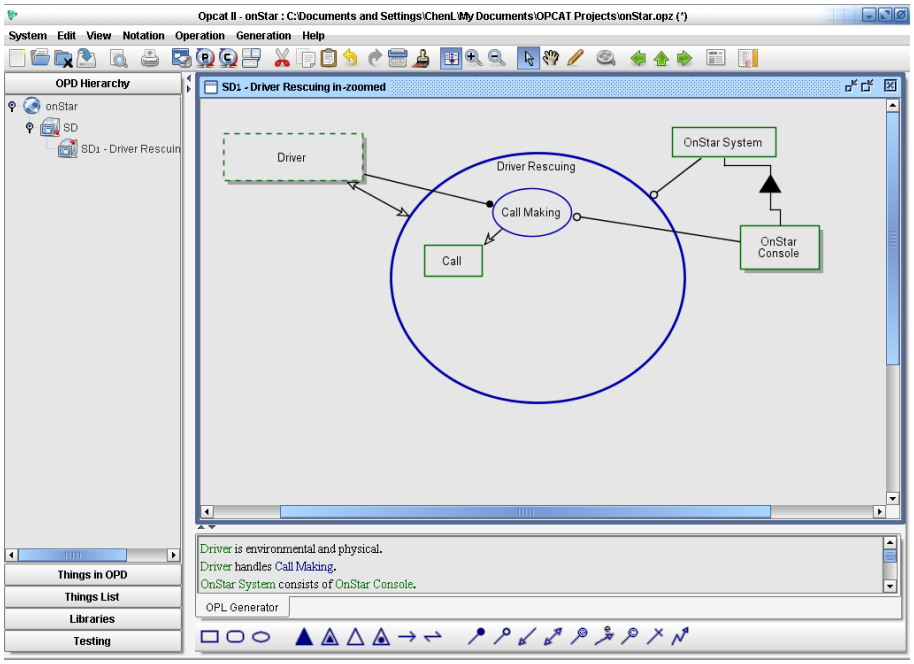


Figure 29. Call is generated as a result of Call Making.

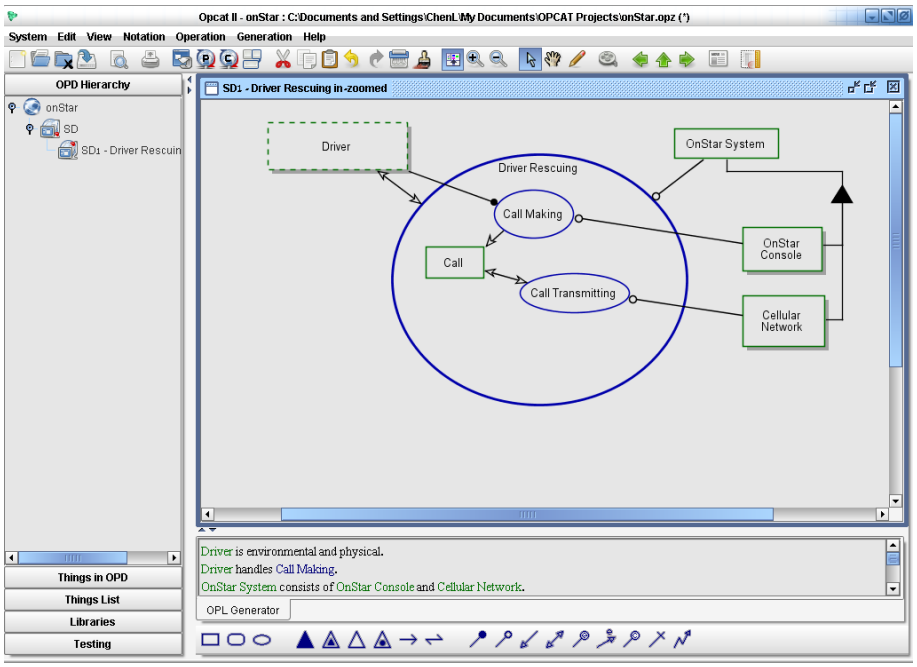


Figure 30. Call Transmitting preconditions

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 already connected to the OnStar System elsewhere, we use the Insert Property feature to facilitate this action. The Insert Property feature is available if the relationship is defined in other OPD or is inherited. We now wish to state that the Driver and OnStar Console are required for Call Making.

OnStar Console is required for Call Making and therefore connected with an Instrument Link. Driver is also required for Call Making but is connected with Agent link. The Agent link denotes that Driver is a human required for the Call Making process. Now we need to ask ourselves what is the result or effect of Call Making. In this case the result is a call.

The next process is Call Transmitting. In OPM, the timeline of synchronous processes is effective within an in-zoomed process ellipse, and it runs from top down. If processes happen in parallel they will be positioned within the in-zoomed process ellipse such that the top of their ellipses will be at the same height. For design of asynchronous processes, that might happen at any order and are therefore triggered without a time line, we use the aggregation-participation symbol, such that the parent process is the whole and its acynchronous descendants are the parts.

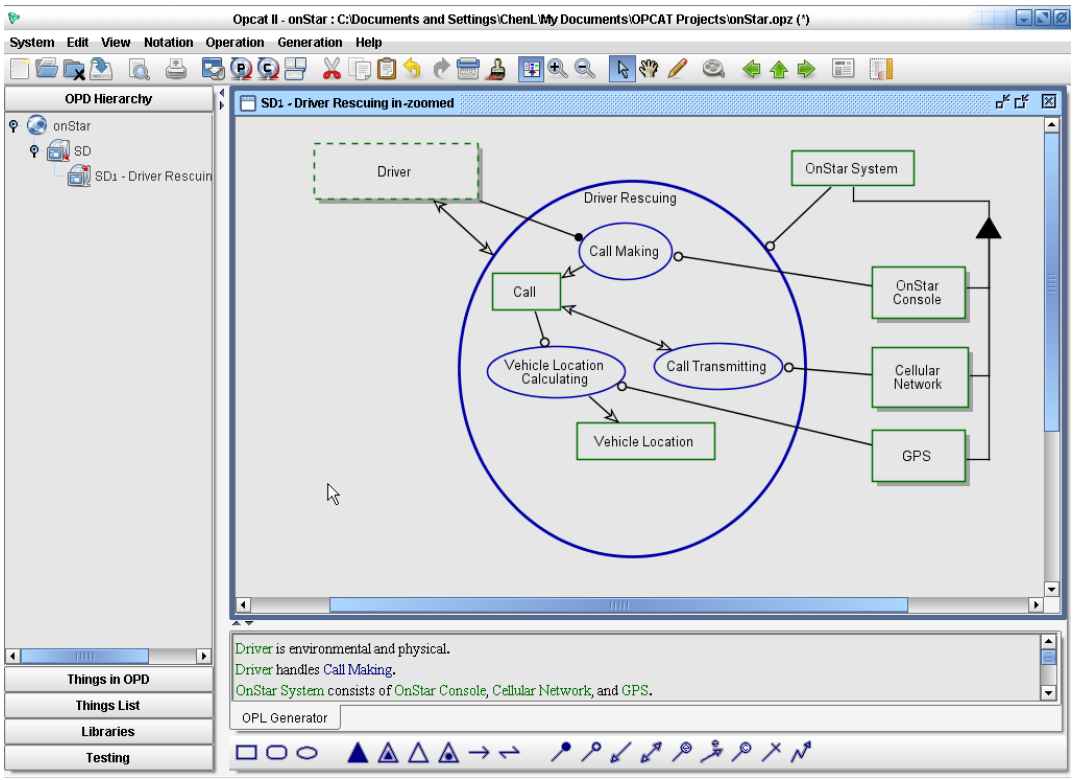


Figure 31. Adding Vehicle Location Calculating

Call Transmitting uses the Cellular Network and it affects the Call. In order to add Cellular Network to SD1 we can use the Insert Property feature or copy it from another OPD and paste it

in SD1. As Figure 31 shows, in parallel to Call Transmitting , the Vehicle Location Calculating process uses the GPS to produce the Vehicle Location.

If an object is involved in all the subprocesses, e.g., as an instrument, it is connected with one link to the in-zoomed process, which acts as parentheses in algebra. For example, OnStar System in Figure 31 is connected with an isntrument link to the in-zoomed Driver Rescuing process, implying that it is instrument to all the subprocesses inside Driver Rescuing.

In-Zooming: A Quick Summary

- To add process details, we zoom into the main process and add the subprocesses inside.
- For synchronous processes, the process occurrence order is from top down. Parallel processes are positioned in parallel.
- For asynchronous processes, we use the aggregation-participation relation, so processes are positioned as descendants of the main process.
- While specifying subprocesses, we add the objects involved in each subprocess and connect them with the appropriate link.
- If an object is involved in all the subprocesses, e.g., as an instrument, it is connected with one link to the in-zoomed process, which acts as parentheses in algebra.

1.7. Consistency Checking

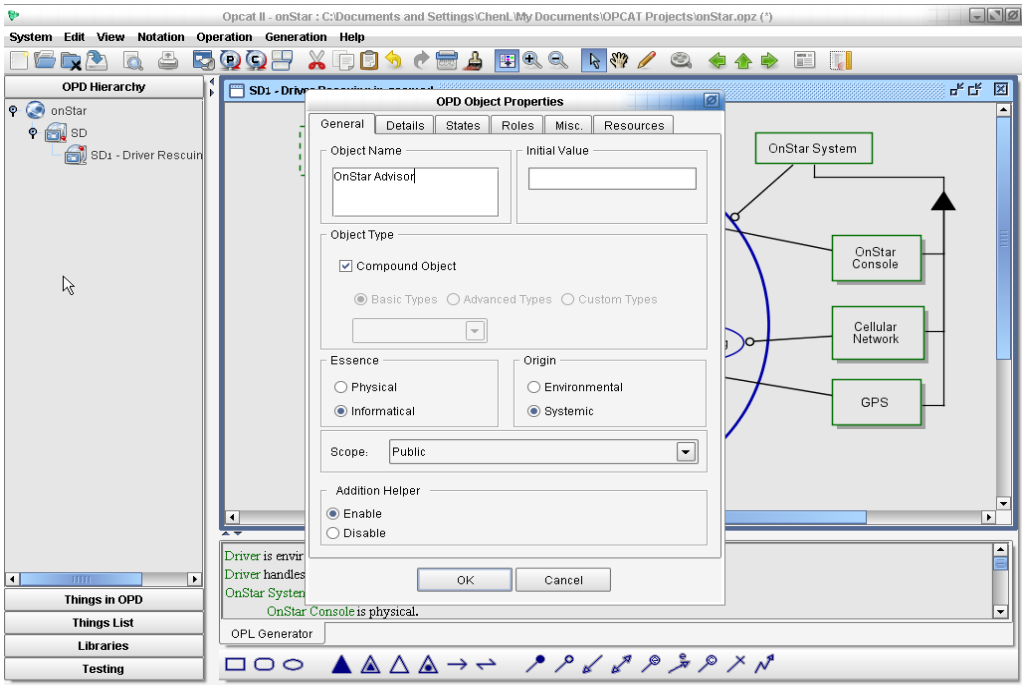


Figure 32. Adding the OnStar Advisor

According to the OnStar System description, an OnStar advisor handles the Call. Let's now insert Call Handling, the final subprocess of Driver Rescuing. First, we need to add OnStar Advisor to our design. This is depicted in Figure 32.

While Call had been previously added inside the in-zoomed Driver Rescuing process, OnStar Advisor is added outside this process. The reason for the difference is that Call exists only within, and in the context of Driver Rescuing, while OnStar Advisor exists even when Driver Rescuing is not active. The general rule for object placement is that an object that is required only within the scope of a process such that is is linked to that peocess subprocesses is located inside the process. Otherwise the object must be positioned outside the process.

As noted, any thing at a lower OPD can only be a refineable (part, specialization, feature, or instance) of a thing that already exists at an upper level. In our case, the OnStar Advisor is part of the OnStar System, but we had not added it in SD snce we did not need it up until now. To help us with maintaining the consistency, as soon OnStar Advisor is added outside Driver Rescuing, OPCAT's Addition Helper switches the OPD in focus to be SD, the top-level OPD, and asks the question shown in Figure 33.

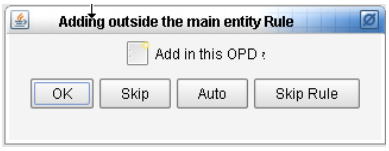


Figure 33. OPCAT's Addition Helper in action

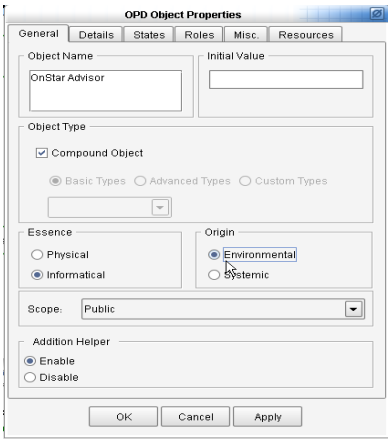


Figure 34. Setting OnStar Advisor's Essence and Origin attribute values

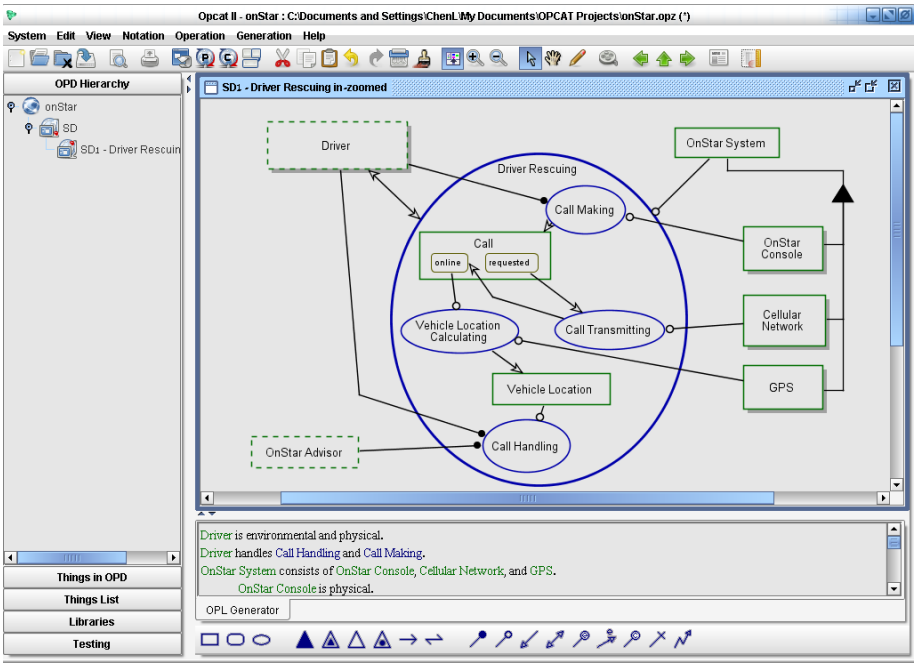


Figure 35. The enablers of Call Handling

Selecting OK will add OnStar Advisor to SD. Another question is what the OnStar Advisor Essence and Origin attribute values are. We can mark these attributes of any OPM thing in the window that opens by double-clicking the thing, as shown in Figure 34.

We now connect the OnStar Advisor with an Agent Link. At SD the OnStar Advisor will be connected via an agent link to the Driver Rescuing process. At SD1, we specify that within Driver Rescuing, the subprocess which the OnStar Advisor handles is Call Handling. In order for Call Handling to occur, the Driver and Vehicle Location are alos needed. These three objects are therefore linked to Call Handling, as shown in Figure 35.

1.8. States

Pertinent to the system, Driver exhibits (has) a Danger Status attribute, with states endangered and safe, as shown in Figure 36. The Call Handling Process changes the Driver's Danger Status from endangered to safe.

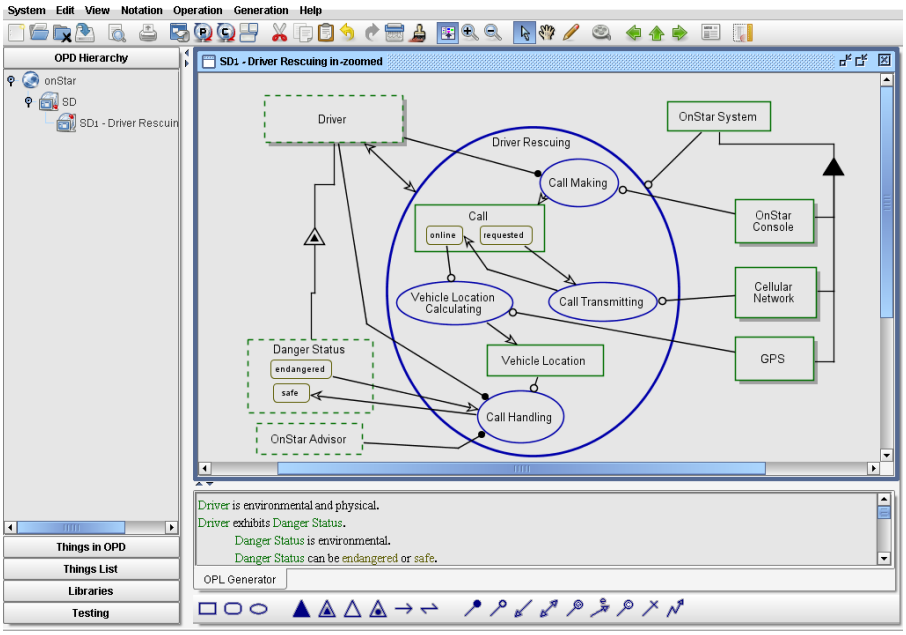


Figure 36. Call Handling changes the Danger Status of Driver from endangered to safe.

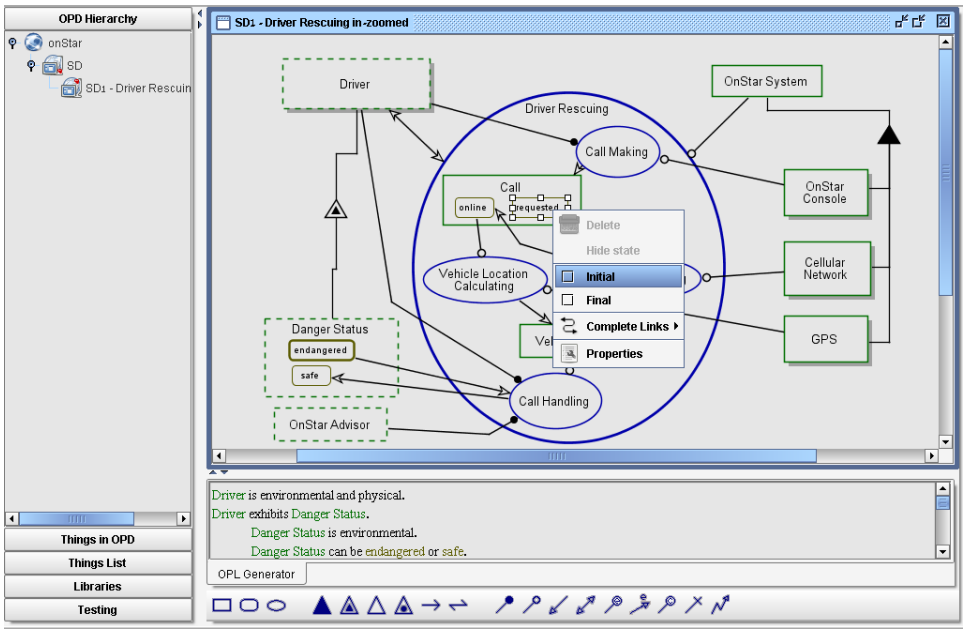


Figure 37. Marking initial states

To specify that the initial Danger Status state is endangered, and the the intial state of Call is requested, we mark them as “initial.” This is shown in

Figure *37.*

1.9. Possible Next Modelling Steps

At this stage when our Driver is finally safe we can continue specifying the details of each of the subprocesses and other details. When zooming into subprocesses we must remember that each in-zoomed process defines the border of the lower-level OPD that is created as we zoom into that process. For example, Call Making uses OnStar Console and the Driver and yields a Call. At the next level we might describe the different parts of the OnStar Console, user interface, authentication processes and so on. Nevertheless, these will only be details of the Driver, Call Making or OnStar Console. If we find that something is missing, then we add it also to the upper OPD, as demonstrated above with regard to the OnStar Advisor.

Going back to the requirements described at the beginning of this guide we read 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. This may be also a good time to revisit the upper OPD and decide whether we need this level of details at the top level. We may decide for example that the parts of the OnStar System will be described only at lower levels.

1.10. Debugging via Animated Simulation

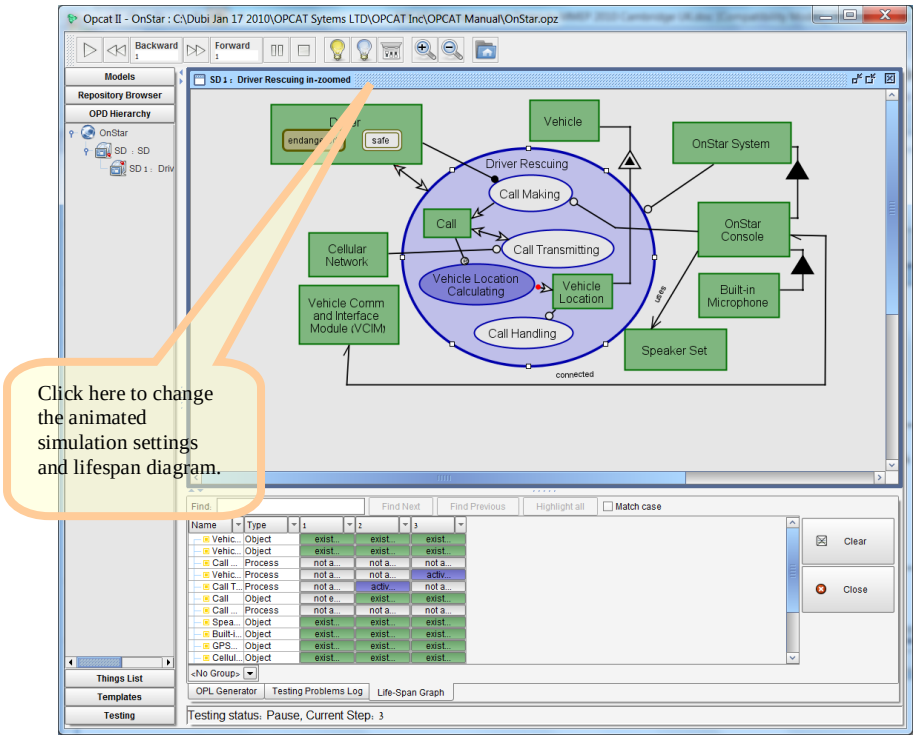


Figure 38. Animated simulation and the lifespan diagram

A very useful feature of OPCAT is its possibility to debug the system as it is being modelled. This can be done not just by reading the OPL text that is being automatically generated in response to every user's graphic edit action, but also via the animated simulation and lifespan diagram that is created gradually with each step of the simulation, as shown in Figure 38 on a somewhat different version of the system.

1.11. Summary

By way of closely following a running case study, this paper has presented OPCAT 3, an advanced software product that supports OPM-based conceptual systems modelling. Only the basic features of OPCAT have been described. Features like requirements management, information layers management, model repository, and automatic document generation and management were not covered but they do exist in OPCAT 3. Advanced features such as simulation, tagging, views creation, model repository and additional OPCAT 3 feaatures are elaborated on in the OPCAT Advanced Course Package, which can be obtained by contacting consulting@opcat.com. OPCAT 4, which is beyond the scope of ths paper, is an enterprise version of OPCAT, which supports also collaborative modelling, version management, central model repository, policies, and much more.

1.12. References

Dori D. Object-Process Methodology – A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York, 2002.

Dori D. Words from Pictures for Dual Channel Processing. Comm. of the ACM, 51(5): 47-52, 2008.

Dori D and Choder M. Conceptual Modelling in Systems Biology Fosters Empirical Findings: The mRNA Lifecycle. *Proceedings of the Library of Science ONE* (PLoS ONE), 2007.

Dori D, Reinhartz-Berger I, Sturm A. Developing Complex Systems with Object-Process Methodology using OPCAT. LNCS 2813:570-572, 2003.

Estephan J. Survey of Model-Based Systems Engineering (MBSE) Methodologies. INCOSE-TD-2007-003-02, accessed Feb. 22 2010. http://www.incose.org/ProductsPubs/pdf/techdata/MTTC/MBSE_Methodology_Survey_2008-0610_RevB-JAE2.pdf

ISO N1049 - OPM Study Group, Terms of Reference, 2009. Accessed Feb. 20, 2010.http://isotc.iso.org/livelink/livelink?func=ll&objId=547513&objAction=RunReport&InputLabel1=004315.