Chapter 1 – Quick Start

Ready to start? We are going to dive right in, do a quick pass through the tool, and create a simple SysML model of a doorbell system. Don't worry too much if the SysML concepts presented in this chapter seem strange; we will come back to explain them in more detail in subsequent chapters.

Introducing the Tool

Before we start constructing our first **model**, there are a few routine housekeeping matters to take care of.

Starting the Tool

Assuming that you selected the installation option to have *Rhapsody* install a Windows desktop icon, the easiest way to start the tool is to click on that icon.

Figure 1-1 – Icon

This start page presents a handy list of information for beginners. This information includes tutorials, a step-by-step guide, and model examples. Behind the scenes, the start page is an HTML page that points to different local and remote resources. If you find this start page helpful, feel free to use it. [1]

If you would prefer not to see the start page, you can deselect "Show Welcome Screen at startup" to prevent the start page from appearing when you start the tool.

[1] The entry HTML page is located in the folder: *〈RhpInstallFolder〉/WelcomeWizard/en/index.html*. Larger organizations often modify and adapt this page to fit the organization's needs.

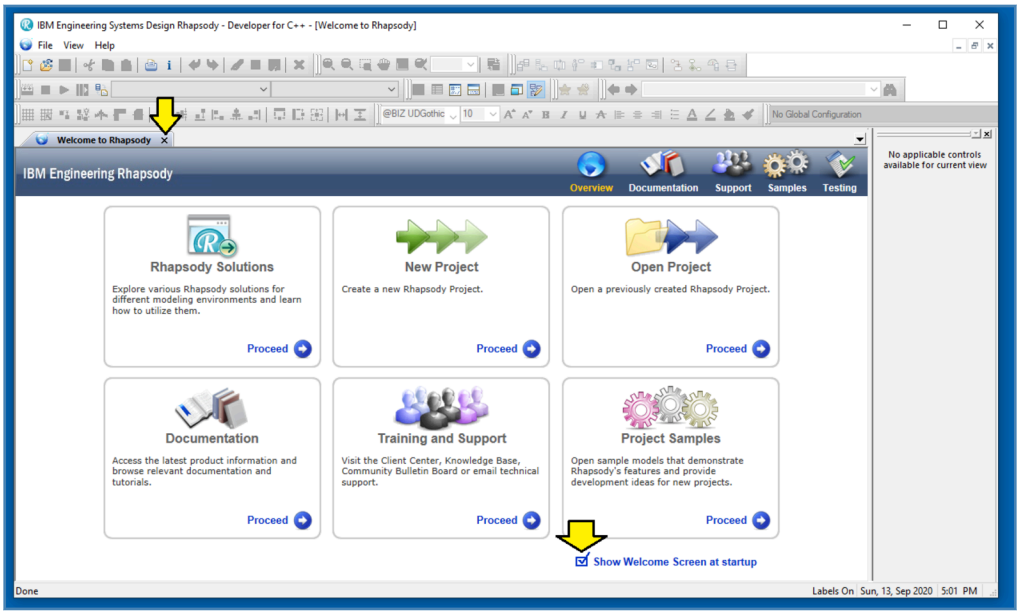


Figure 1-2 – Start Page

Backing Up a Model

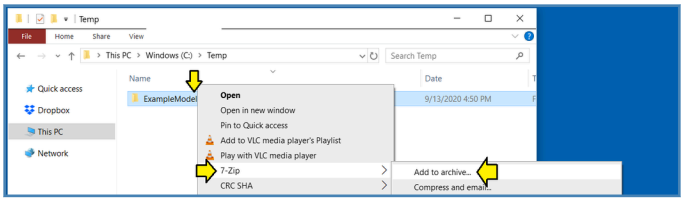


Figure 1-3 – Use a Zip tool to back up the model directory

For each new model, *Rhapsody* creates a directory with several XML files as well as one or more sub-directories for model resources. The easiest way to make a backup copy of the model is to use a Zip tool to create an archive for the entire directory and its contents.

Using A Source Code Control System

Rhapsody is actually very friendly to source code control systems such as Git or Subversion. Not only are the model files text files – unlike most other similar tools – *Rhapsody* contains features to allow fine-grained control of the partitioning of the model into multiple XML files. Although the details of this mechanism are beyond the scope of this book, this functionality is very helpful for partitioning models so that different team members can be working on different sections of a model.

IBM also offers a very sophisticated suite of cloud-hosted tools for engineering lifecycle management.

See: https://www.ibm.com/products/engineering-lifecycle-management-ext

Creating a Model

Now we are ready to create our first **model**.

Note that it is important when you start modeling to create a project directory on your computer somewhere and use it consistently for modeling projects. However, the management of a disk folder structure is beyond the scope of this book. For the moment, we will use the directory *"C:\Temp"* as the project directory.

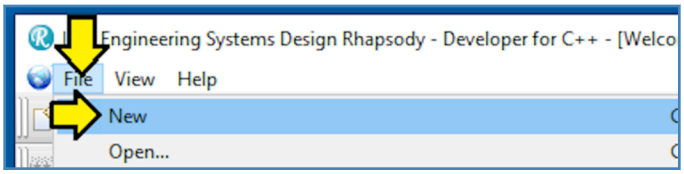


Figure 1-4 – Starting a new project

Pull down the "File" menu in the top left corner of the screen and select "New" to start a new project.

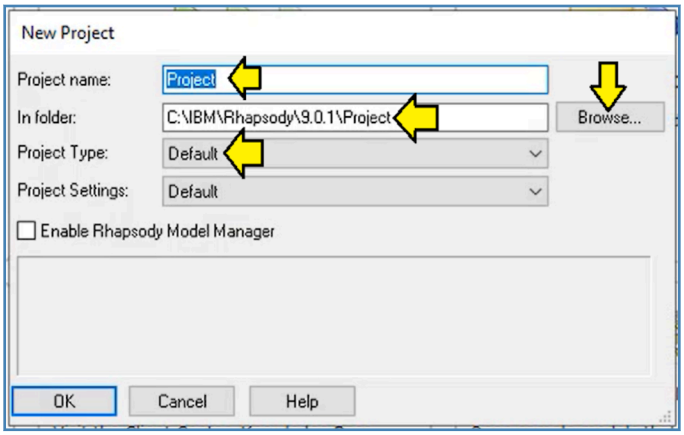


Figure 1-5 – New project panel

The "New Project" panel will appear:

- The "Project name:" field contains a default value of "Project".
- The folder is pointing to the installation directory for the tool. As mentioned above, you should have a well-defined directory for your projects. Creating your projects in the tool installation directory is probably not a good idea from a data management point of view. You will want to override the default with a better choice of project directory.
- The "Project Type:" has not yet been set and contains a default value of "Default".

Click on "Browse..."

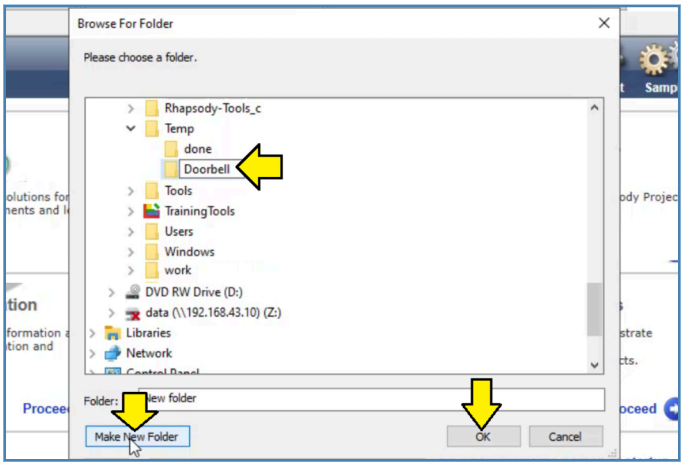


Figure 1-6 – Create a project folder

Navigate to *"C:\Temp"* or whichever directory you have chosen as your project storage directory. From the "Browse for Folder" panel, click on "Make New Folder" and create the folder: "Doorbell" as a subfolder of "*C:\Temp*".

This folder will be the project folder which will store the model as well as any other reference information you might want to store together with the model.

Click "OK" to return to the new project panel.

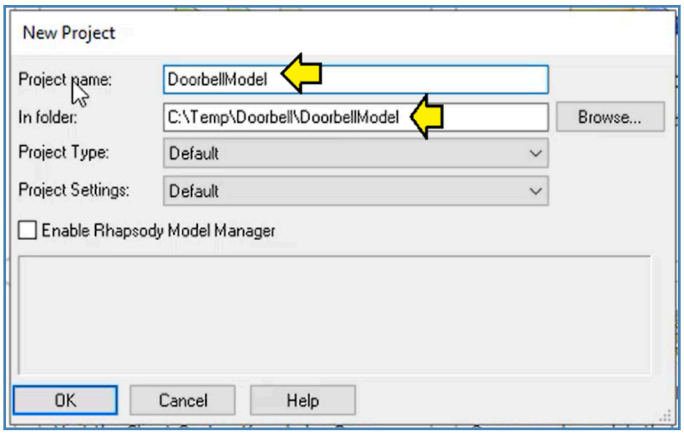


Figure 1-7 – Create a model folder

In the new project panel, use the "Project name:" field to name the model "DoorbellModel". Do ***not*** click "OK" yet.

- You will notice that as you are typing in the name of the model, the tool is automatically filling in the name of a subfolder for the model that it will create shortly.
- We recommend that the name of the model end with the word: "Model".
- The name of the model cannot contain spaces or punctuation marks.

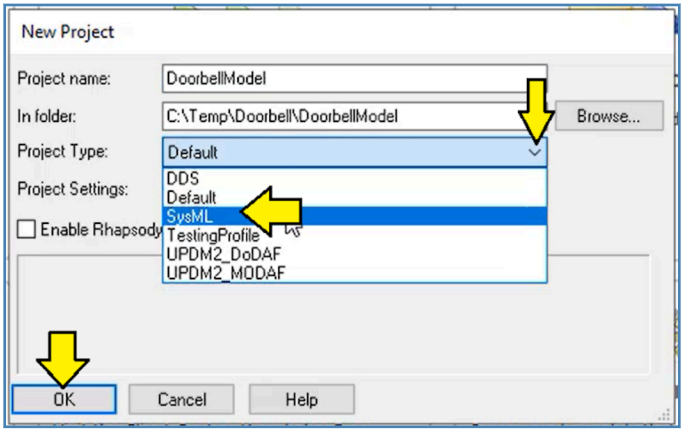


Figure 1-8 – Select SysML

- Pull down the "Project Type:" menu and select "SysML".
- Click "OK".

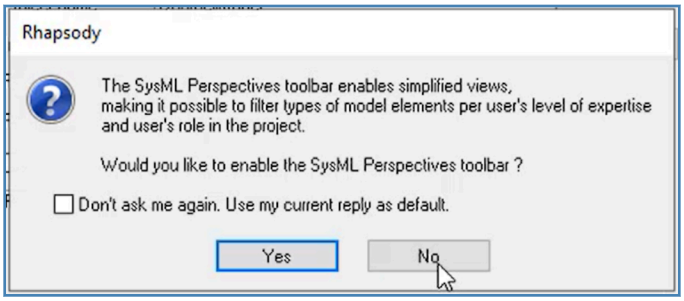


Figure 1-9 – Decide whether you would like a simplified view

The next panel will prompt you to decide whether you would like simplified views or not. In large organizations that have invested in configuring and customizing views for particular jobs, these simplified views can be helpful. However, simplified views can have the disadvantage that they sometimes hide the very thing you were looking for. We will not be using the simplified views in this book. For the purposes of using this book, we recommend that you select "No".

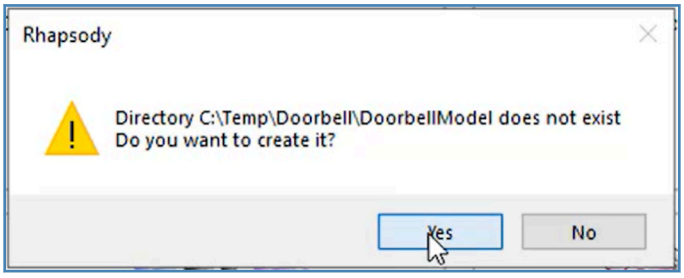


Figure 1-10 – Confirm creation of the directory for the project

Confirm creation of the directory (folder) for the model. This step will create the subdirectory for the model (only) within the project folder which contains the model but can also contain other information.

There will be a brief delay while the tool creates the model.

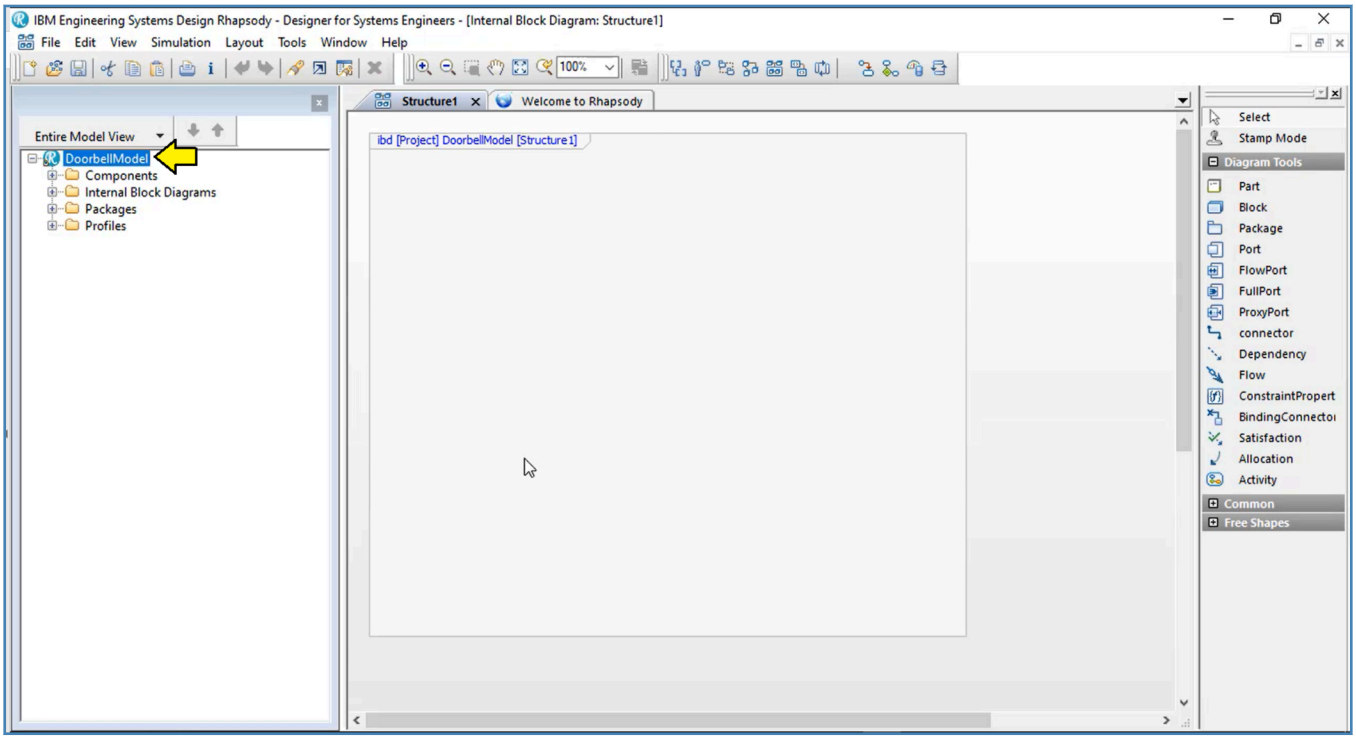


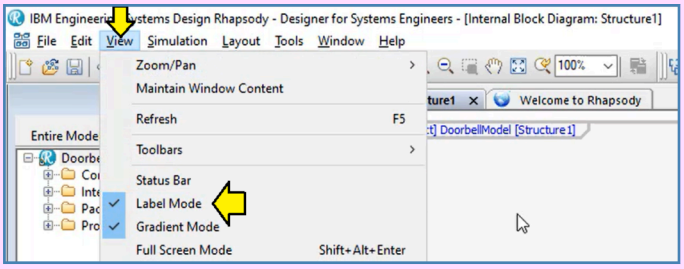
Figure 1-11 – Empty model

We now have an empty SysML model. Notice that the model has been named "DoorbellModel". (2)

(2) Many other SysML tools do not name the model automatically, instead presenting you with something generically named: "Model".

Label Mode

Rhapsody had its origins long before SysML was created as a UML modeling tool with a strong emphasis on automatic generation of source code. As such, the tool is designed to enforce the usual requirements for software source code identifiers on element names. Notably, you will not be allowed to have names like: "Big Red Car" as the spaces would violate identifier rules for most compilers. In systems engineering, we are mostly modeling for humans, not compilers. Most systems engineers prefer to have more flexibility in choosing names for model elements. Fortunately, *Rhapsody* provides an option called "Label Mode" to allow normal human-friendly names. This option can be set in the "View" menu.



Since many of the examples in this book contain spaces in their names, go ahead and set label mode now. This setting is persistent and will stay set in the tool until you choose to deselect it.

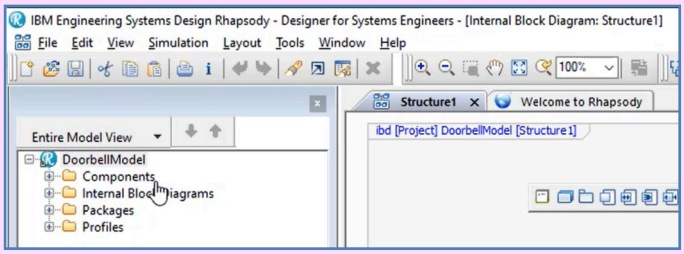
Adding a Package

Before we can do anything useful with our new model, we will need to add a **package**. Packages are a central feature of SysML models. The model itself is a special kind of package. You can think of packages as being very similar to directories in the Windows file system. You can name packages freely. You can put packages within packages. [3]

[3] Depending on which edition you have installed, *Rhapsody* will provide you with either a block definition diagram or an internal block definition diagram in the root of the model. However, it is better practice to define a package structure within the model and put diagrams into the package structure.

Categories vs Packages

Before we start digging into packages, we need to discuss packages versus categories. *Rhapsody* has a unique feature that is not common in other tools. At each level of the model package hierarchy, the tool shows "category folders" and automatically sorts elements into their categories.



At the top of the browser at the left, you will see a number of items with names like "Components" or "Profiles" accompanied by the normal "Manila Folder" icon that most other tools use to identify a directory or a folder. These are not SysML packages, but rather a sorting mechanism. For example, as soon as you create a **block**, one of these category folders will appear called "Blocks" and your new block will be inside the category folder.

If you are coming to *Rhapsody* from another modeling tool, this behavior will initially seem unusual. However, as we continue to model, you will get used to the category folders and find them helpful in keeping your model sorted. There is an option to disable this behavior, but we will not be using that option in this book as most users of *Rhapsody* use the tool with this default presentation behavior enabled.

By default, packages are in the "Packages" category folder. In fact, *Rhapsody* comes with one ready-to-go package called "Default". However, one package isn't going to get us very far. Let's learn how to create a package.

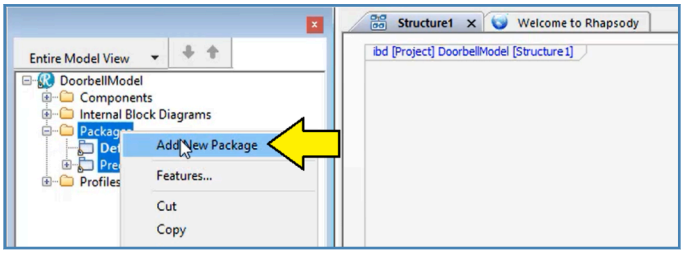


Figure 1-12 – Create a package

Right-click on "Packages" category folder and select "Add New Package".

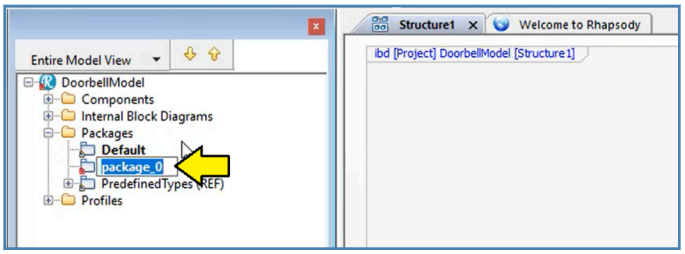


Figure 1-13 – Package created with default name

Rhapsody creates a package with a default name.

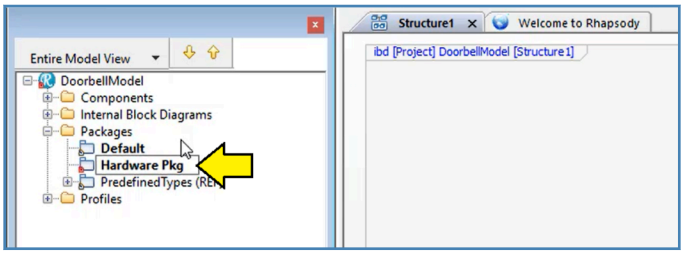


Figure 1-14 – Name the package

Click on the default name and give the package a new name. We recommend that you name the package "Hardware Pkg".

Package Naming Tip

In principle, *Rhapsody* really doesn't care how elements are named. As such, it is very easy in the normal course of modeling to end up with a package, an element, and a diagram all named "Car". As your model grows in size, these identical names for different things can get confusing. In order to make the model easier to understand, Frank recommends getting in the habit of adding "Pkg" to the end of all package names. We will be following this convention for the rest of this book.

Adding a Block Definition Diagram

Now we are ready to add our first diagram. SysML defines nine types of diagrams. The first diagram we will add is a **block definition diagram** – the most basic diagram for defining the elements of a system.

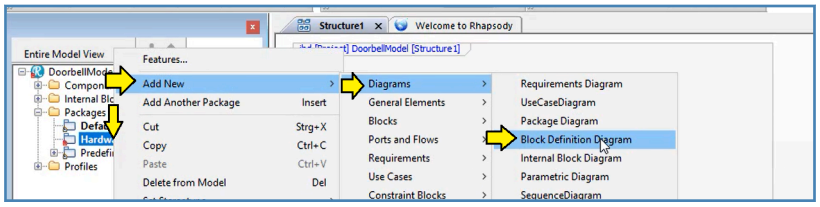


Figure 1-15 – Right-click and add diagram

Right-click on the new package "Hardware" and select "Add New", then "Diagrams", and then "Block Definition Diagram".

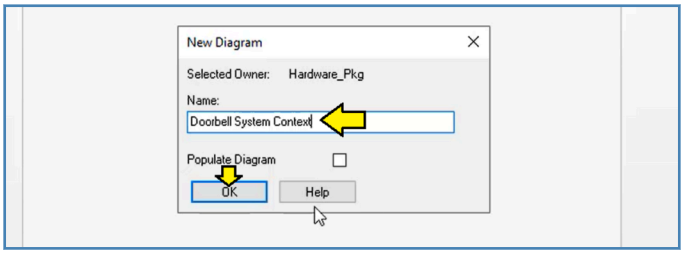


Figure 1-16 – Name the block definition diagram

Name the diagram "Doorbell System Context" and click "OK".

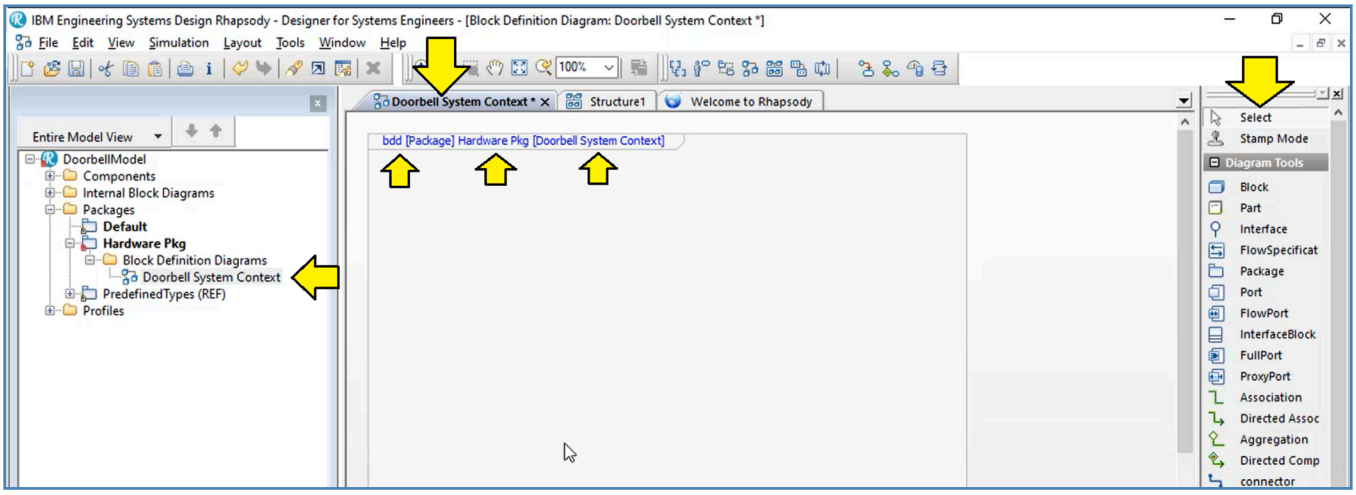


Figure 1-17 – Quick look at the tool

Let's take a quick look at the tool now that we have created the block diagram as shown in Figure 1-17:

1) On the left side of the screen we can see a pane called the **browser**. This pane presents a hierarchical view of the model than can expanded and explored similar to the file explorer user interfaces in Windows and other operating systems.

2) In technical articles, this view is often referred to as the "Containment Tree".

3) In the browser, we see the new block definition diagram called "Doorbell System Context" in the package "Hardware".

4) At the right of the screen we can see a pane called the **drawing toolbar**. This pane contains icons for different SysML elements that are appropriate for block definition diagrams.

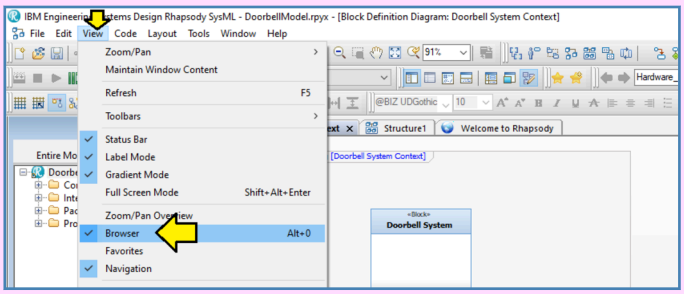
5) To the middle of the screen, we see that the tool has created our block definition diagram.

6) The diagram pane is a "tabbed view" layout which can show multiple diagrams as tabs. We can see the tab for our diagram at the top of the screen.

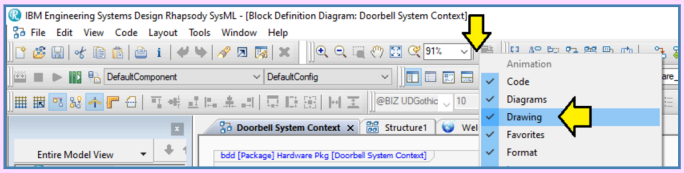
7) Our diagram has a standard SysML frame with a header. In the header, "bdd" stands for "block definition diagram". This diagram lives in the package "Hardware Pkg". The name of the diagram is: "Doorbell System Context".

What if the Browser or the Drawing Toolbar isn't there?

What if the browser or the drawing toolbar isn't visible on your screen? This kind of thing happens all the time with this sort of highly configurable tool.



The first problem can be solved by selecting "View" and then selecting "Browser".



The second can be solved by right-clicking on any toolbar and selecting "Drawing".

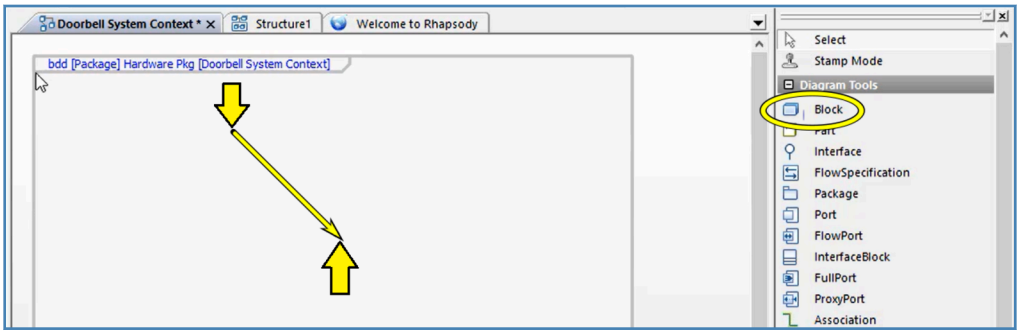


Figure 1-18 – Click on Block icon and drag to diagram

Now we are ready to add our first **block** to our model.

1) In the **drawing toolbar** pane at the right of the screen, select the "Block" icon.
2) Move the mouse to the diagram and pick a position for the upper-left corner of the block icon.
3) Click and drag to create the outline of the shape as shown in Figure 1-18.
4) Release the mouse button.

This motion will add a block to the model and format its shape in the diagram.

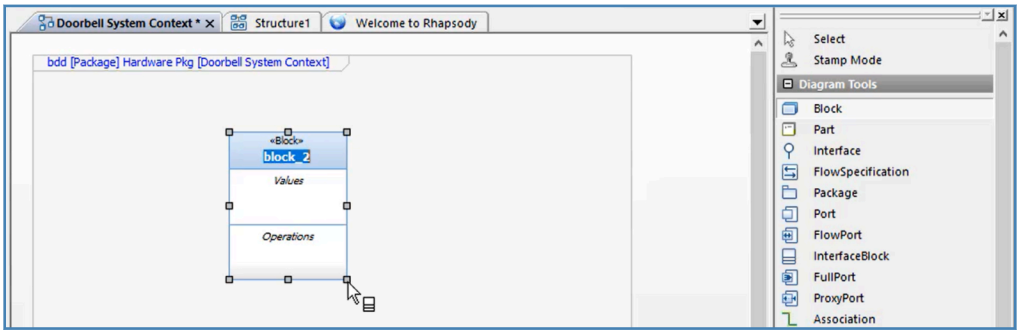


Figure 1-19 – A block will appear in the diagram

A block will appear in the diagram.

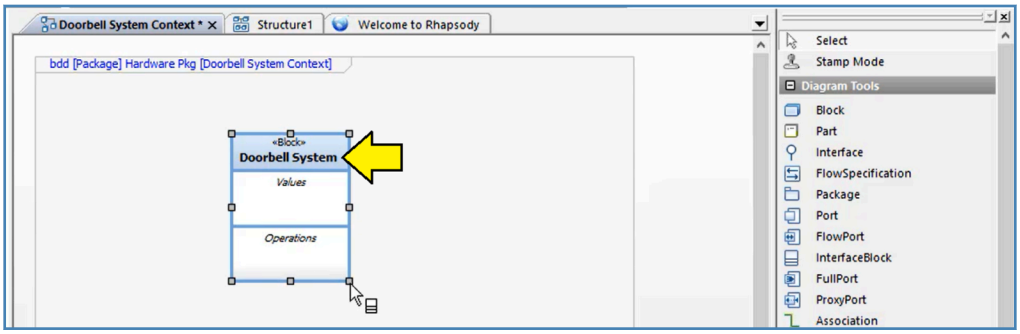


Figure 1-20 – Click on the block and name it

If the block is not selected, click to select the block. After the block is selected, click the block and the name of the block will highlight. Now you can edit the name. Name the block "Doorbell System".

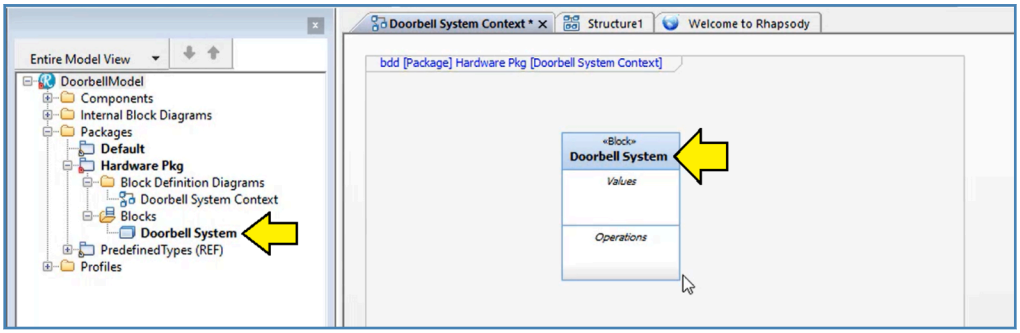


Figure 1-21 – The new block appears in model at left and in the diagram

Examining Figure 1-21 we see the block in two places. We see the block in the browser in the left pane. We also see the block in the diagram in the right pane.

Examining 1-21 you will notice that the tool is displaying two empty compartments: "Values" and "Operations". Since we don't yet have any values or operations to display, let's go ahead and hide these compartments.

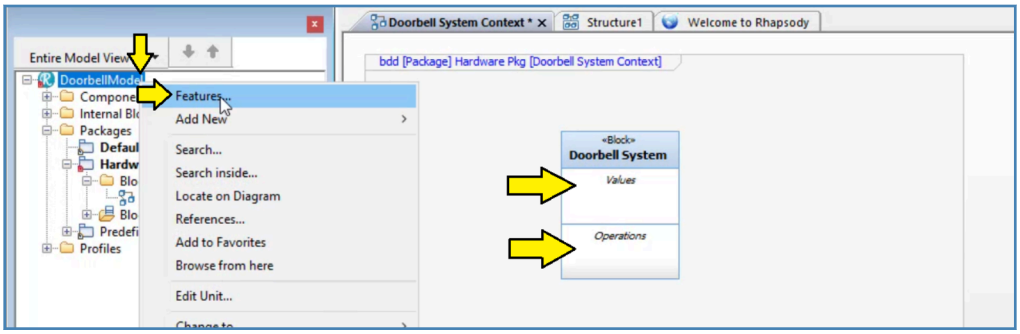


Figure 1-22 – Open model features

Right-click on the model and select "Project Properties..."

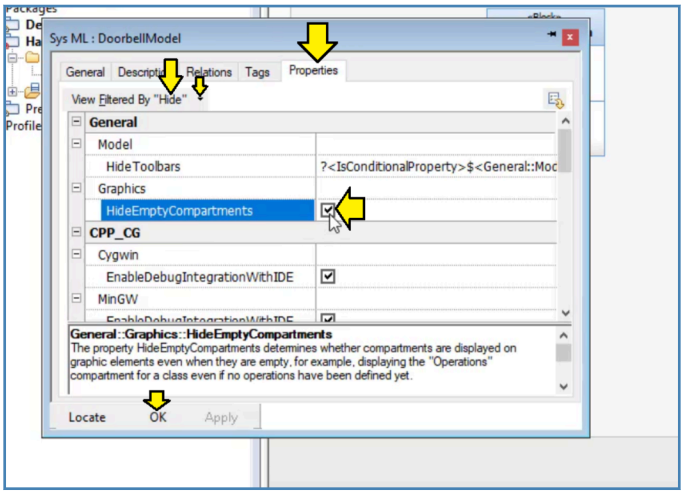


Figure 1-23 – Filter by hide and select

The model features panel will appear.

1) Select the "Properties" tab.
2) At the top left of the "Properties" tab, you will find a data entry field for entering a text string to filter the properties shown. Pull down the small arrow at the right of this field and enter the filter string "Hide".
3) The list of properties will resort itself, showing only properties containing the filter string. One of these will be the property: "HideEmptyCompartments".
4) Select "HideEmptyCompartments".
5) Click "OK".

That will hide empty compartments for all new blocks that you create. Next let's fix the one you already created.

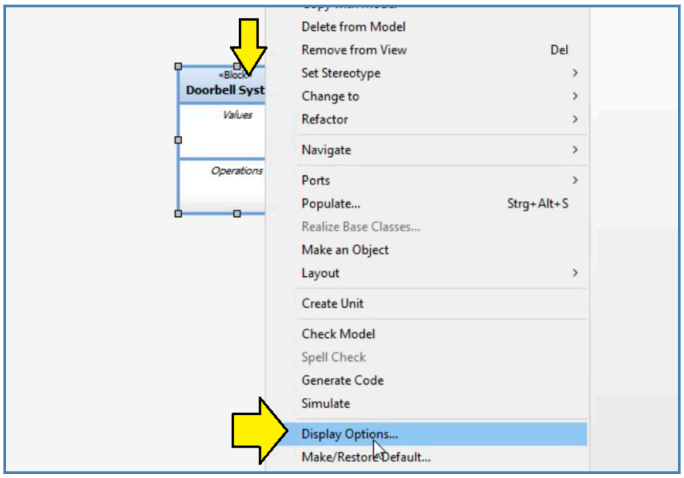


Figure 1-24 – Right-click and select display options

Right-click on the block in the diagram and select "Display Options..."

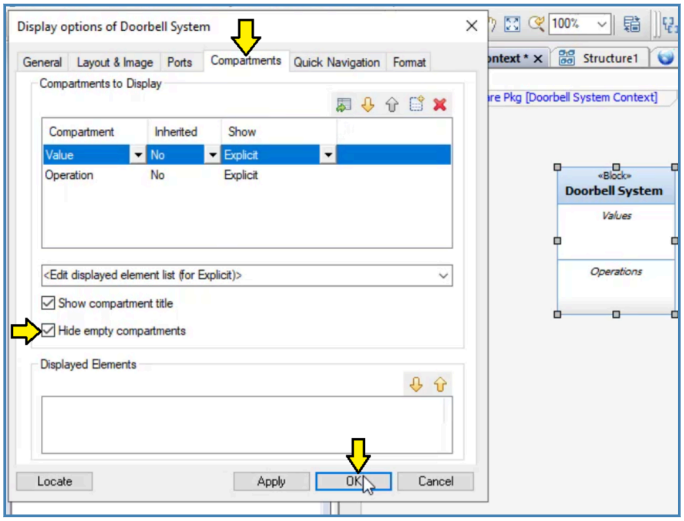


Figure 1-25 – Select compartments tab and hide empty compartments

Select the "Compartments" tab. Select the "Hide empty compartments" option. Click "OK".

Our First Glimpse of Model-Based Systems Engineering (MBSE)

We are at now at the starting line for "Model-Based Systems Engineering (MBSE)". That is, the browser is showing "the model" and the diagram is just a diagram. We could make two or three more diagrams with the same "Doorbell System" block. In the model there would still be one and only one block. That is, the model represents the *actual system* and the diagrams are just *views* of the model. That is, if we think of the model as a building, the diagrams are like snapshots taken of the building from different angles. The snapshots (diagrams) are not the building. The model is the building.

A model element can be shown in as many or as few diagrams as desired. In fact, you can create a model element directly in the browser and not include it in any diagrams at all. This flexibility to make many different diagrams including the same element becomes a central feature of MBSE because this technique allows the systems engineer to make many simple diagrams for different stakeholders, each diagram showing only the model elements of interest to that stakeholder.

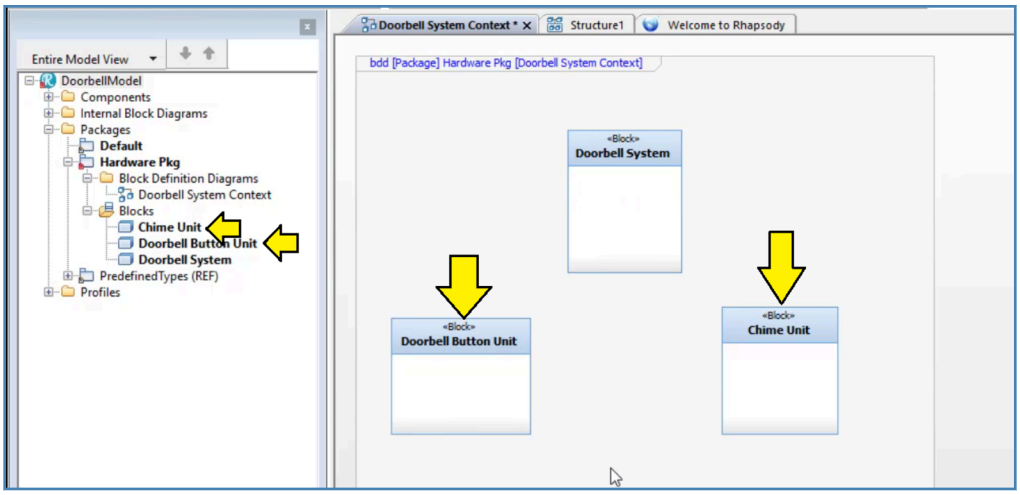


Figure 1-26 – Add two more blocks to the diagram

Using the same procedure, add two more blocks and name them:

- Doorbell Button Unit
- Chime Unit

Now that we have the two main units for our system, we are going to introduce a relationship called a **composite association**. This relationship is often referred to more simply as "composition".

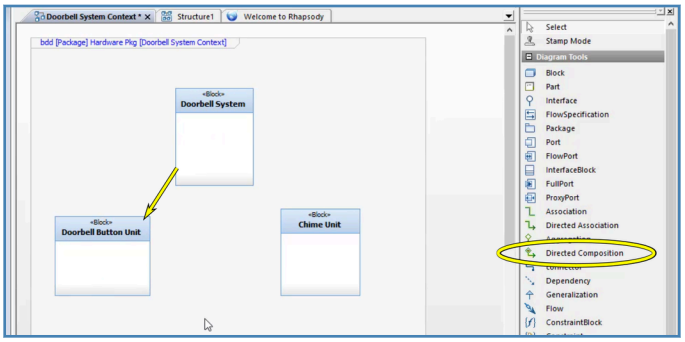


Figure 1-27 – Select the composition relationship and drag as shown

In the drawing toolbar, you will find an icon called "Directed Composition". Select this icon. Then drag from the "Doorbell System" block to the "Doorbell Button Unit" block as shown in Figure 1-27.

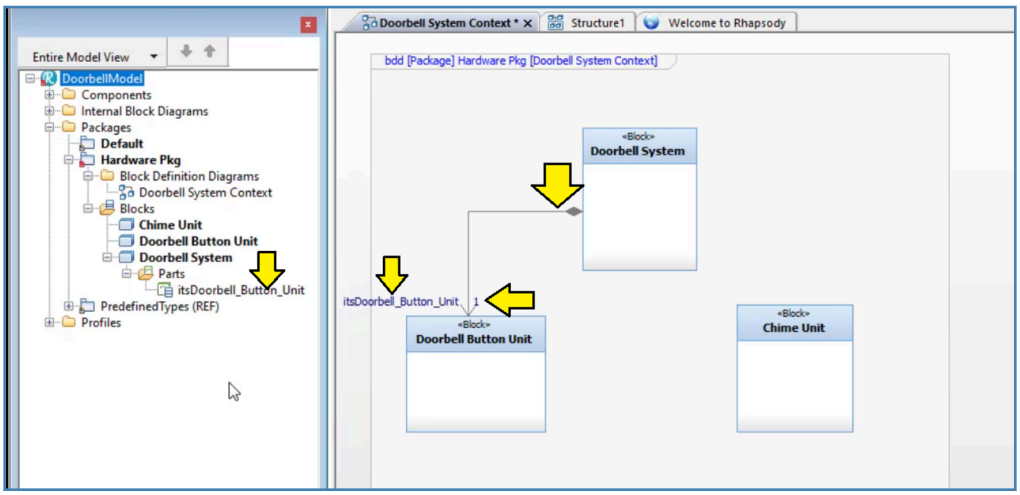


Figure 1-28 – Composition relationship

After you have completed the drag operation to connect the two blocks, four changes will be visible in the tool as shown in Figure 1-28.

1) There is now an arrow from the "Doorbell System" block to the "Doorbell Button Unit" block.
    - At the base of the arrow (the "Doorbell System" block) the connector is a black diamond. The black diamond indicates composition as in: "The Doorbell System is composed of a Doorbell Button Unit."
    - The other end of the arrow is an open arrowhead. This arrowhead indicates the direction of navigation, which does not matter here, but which we will cover a little more in the chapter on block definition diagrams.

2) There is text label next to the open arrowhead "itsDoorbell_Button_Unit" next to the open arrowhead which we will look at a bit more closely in a minute.

3) On the other side of the arrowhead, we will see the number "1". This number is the multiplicity which we will also look at a little more below.

4) In the browser, we can see that "Doorbell System" now has a part nested underneath it called "itsDoorbell_Button_Unit".

What we have done is modeled the idea that the "Doorbell System" contains a **part**, which *Rhapsody* has helpfully named: "itsDoorbell_Button_Unit" for us. [4] This part is an instance of the block of type "Doorbell Button Unit". For the moment, we are not going to worry too much about the differences between "types" and "instances". We will cover that topic in more detail in later chapters.

Before we continue, let's give the part a name that is more useful for our project.

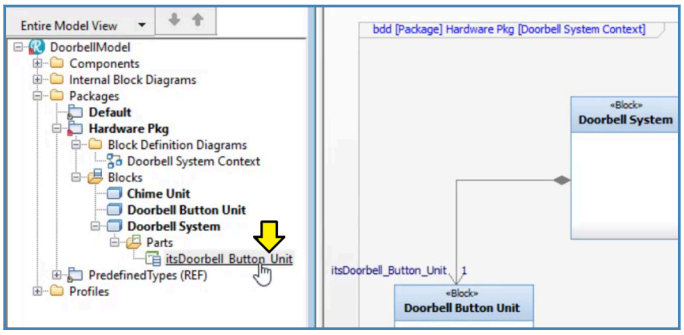


Figure 1-29 – Double-click on the part

In the browser, double-click on the part name.

[4] Note that part names are actually "roles" from a UML point of view. We will discuss this in more detail in the chapter about internal block definition diagrams.

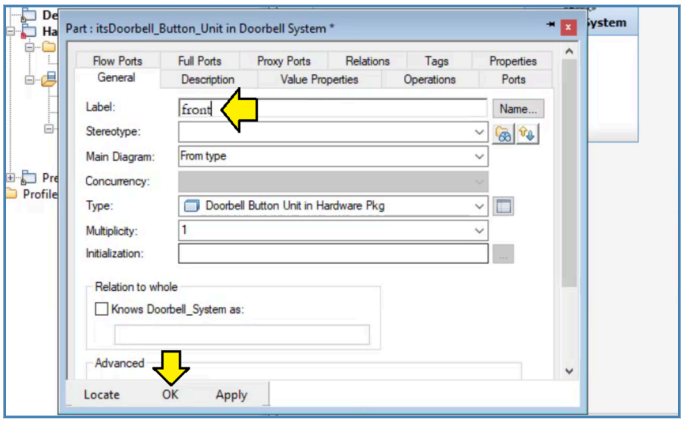


Figure 1-30 – Name the part

In order to give the part a more useful name, replace "itsDoorbell_Button_Unit" with "front" and click "OK".

What we have done here is identify this part as the "front" doorbell button. That is, some houses may have doorbell buttons at other doors such as the kitchen door. The particular doorbell button we are modeling here is the one at the front door.

Now, about that number "1". This number is the multiplicity. The number "1" indicates that there is exactly one doorbell button at the front door – as opposed to say, seven front doorbell buttons. We are going to cover multiplicities in more detail in the chapter on internal block diagrams. Since we don't really need this information in the diagram at the moment, let's take a look at how we can hide it.

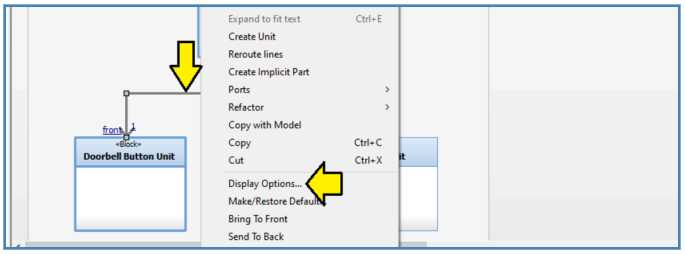


Figure 1-31 – Right-click on the arrow and select Display Options...

Right-click on the arrow and select "Display Options..."

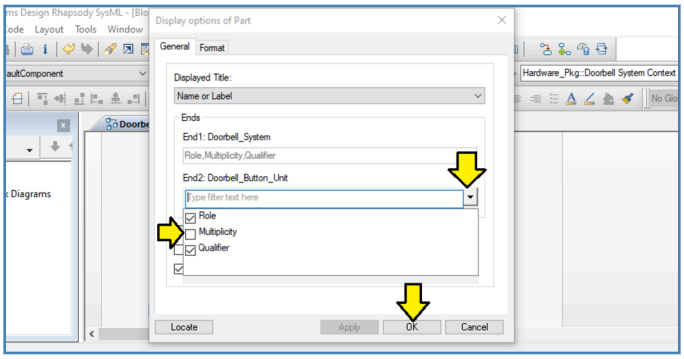


Figure 1-32 – Pull down End2 and unselect Multiplicity

Pull down "End2", unselect "Multiplicity", and click "OK".

Draw a second composition arrow from the "Doorbell System" block to the "Chime Unit" block and name the new part "kitchen". Drag the arrows and text boxes around to make the diagram look a little neater.

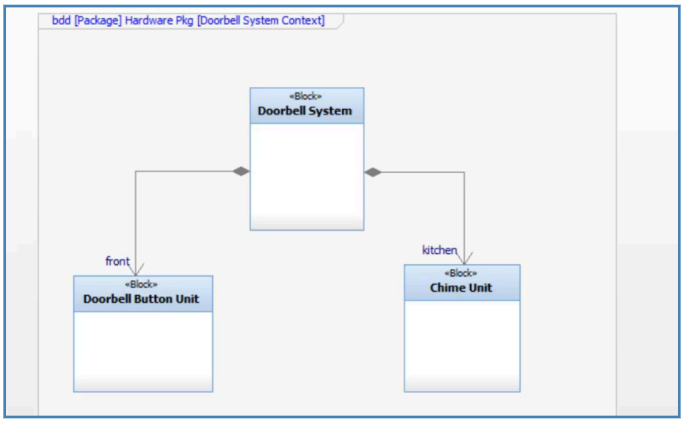


Figure 1-33 – Both part relationships

Your diagram should now look like Figure 1-33.

Next, we will add an **actor** called "Visitor" to the diagram.

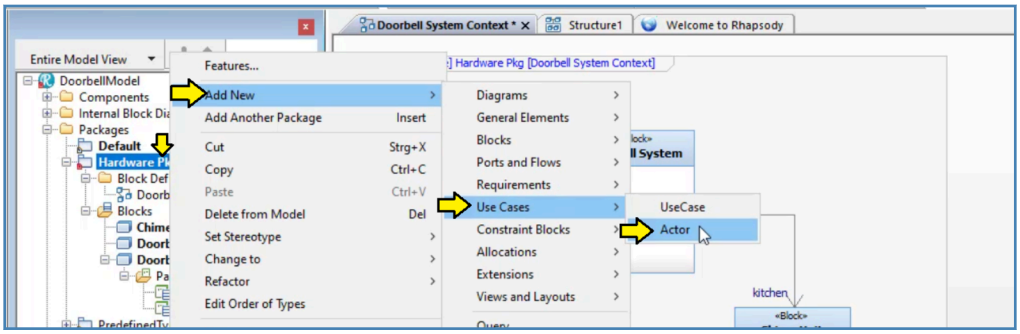


Figure 1-34 – Right-click on the hardware package

Right-click on the "Hardware Pkg" package in the browser and select "Add New" and then "Use Cases" and then "Actor".

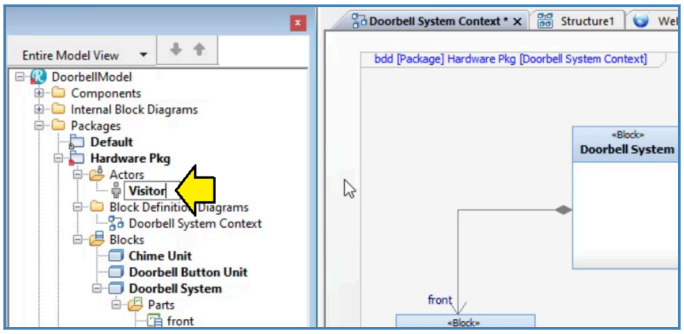


Figure 1-35 – Name the actor

Single-click on the new actor in the browser to enter name-editing mode. Name the actor "Visitor".

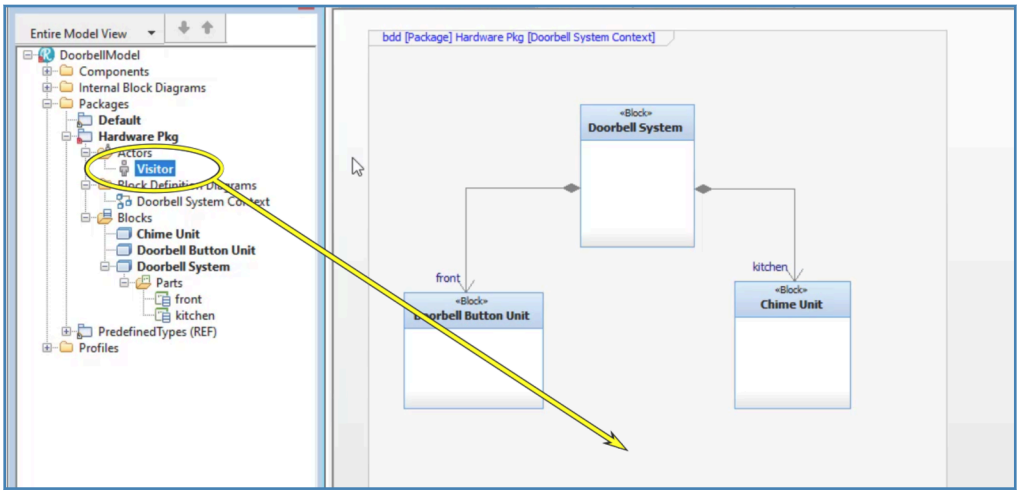


Figure 1-36 – Drag the actor icon to the diagram

Drag the "Actor" icon to the diagram.

We can now see that the visitor is visible in the diagram and has a human shape to it. However, since our actor is a human [5], we will not want to model the actor as belonging to the package "Hardware Pkg".

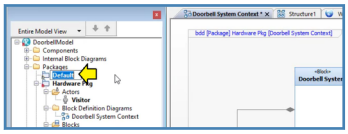


Figure 1-37 – Single-click on Default package and rename it

In the browser you will find a package called "Default" that was automatically created by *Rhapsody* when we created the model. Single-click on this package and rename it to "Actors Pkg".

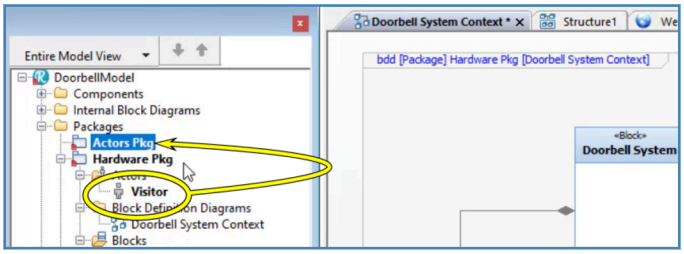


Figure 1-38 – Drag visitor to actors package

Drag the visitor from the "Hardware" package to the "Actors" package.

[5] Actors do not need to be human. In fact, SysML provides an alternate box notation for actors such as cloud server processes that are not humans.

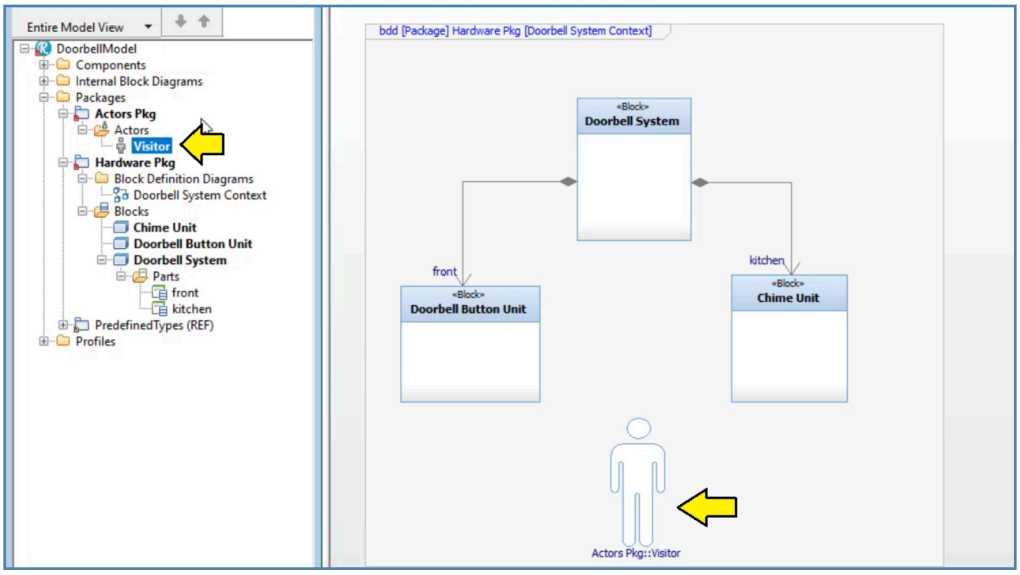


Figure 1-39 – Visitor now in proper package in the browser

Your diagram should now look like Figure 1-39. There are two important points about the diagram in its current form:

- Notice the notation in the diagram "Actors Pkg::Visitor". This notation indicates that this element belongs to a different package. That is, the diagram belongs to the package "Hardware Pkg" and the visitor is a guest from the package "Actors Pkg" so to speak.
- Notice also that the visitor is part of the context for the system but is not part of the system itself. That is, the visitor is shown in the system context diagram, but there is no composition arrow from the doorbell system block to the visitor. This is an important point: deciding which elements are part of the system and which others are merely things in the neighborhood is one of the first steps in a disciplined systems engineering process.

System Context Diagram

SysML does not actually define something called a "System Context Diagram". However, a block definition diagram can be used to model the system context. Nailing down the system context is an important early step in any disciplined systems engineering effort. Specifically, a skilled systems engineering practitioner will help the stakeholders nail down the definition of which things are part of the system being designed versus which things are merely "nearby".

This activity can be surprisingly difficult! Typically, the meeting will start with the stakeholders looking bored and a little irritated. Some will be asking why they have to be at this meeting. Others will be confidently asserting that "Everyone knows what is in the system anyway." However, as soon as the systems engineering practitioner starts drawing the specifics on the whiteboard, the room will explode into fractious disagreement. In most cases, the stakeholders will have overestimated their level of common understanding and consensus.

Adding a Requirements Diagram

Now we are ready to add a few requirements to our system. The first thing we will want to do is add a package to the model called: "Requirements Pkg" (See the procedure previously outlined in *Adding a Package* on page 8).

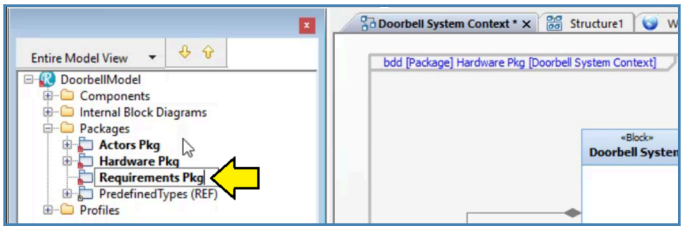
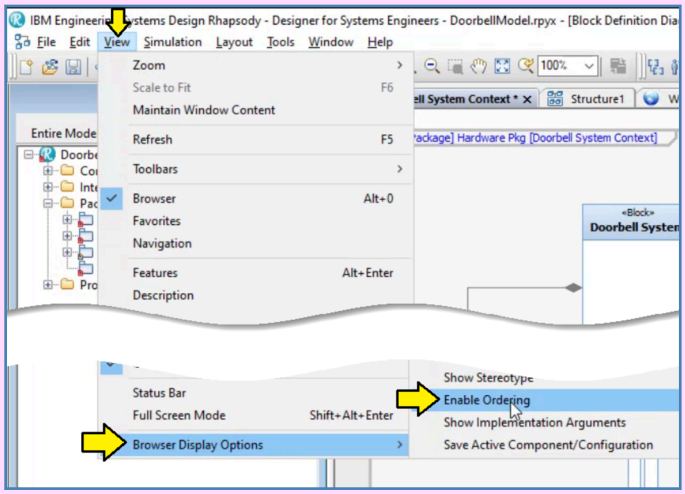


Figure 1-40 – Requirements package

Enable Ordering in the Browser

Taking a close look at Figure 1-40, you will notice that there is another built-in package called "Predefined Types (REF)" in the middle of your list of packages. You may not want that in the middle of your list of packages. More importantly, you may want to control the order of the elements displayed in the browser.



Rhapsody supports manual control of the ordering of elements in the browser. To enable this ordering function, pull down the "View" menu, select "Browser Display Options" and then "Enable Ordering". After that, you will be able to use the up and down arrows at the top of the browser to move the packages and other elements up and down.

Note that it is also possible to simply hide the "Predefined Types (REF)" package. Select the root of the model in the browser. From the "View" menu, select the properties panel, use the filter function to search for the string "predefined", and unselect the option "ShowPredefinedPackage".

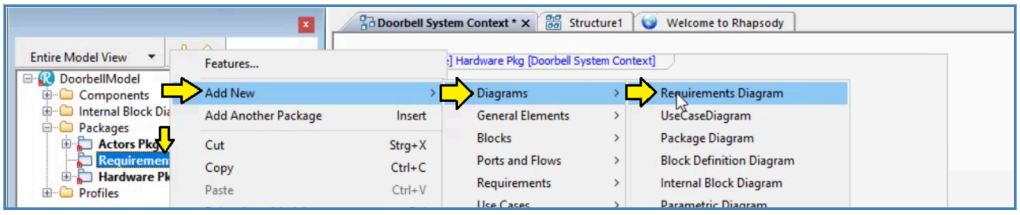


Figure 1-41 – Right-click to add diagram

Right-click on the "Requirements Pkg" package and select "Add New" and then "Diagrams" and then "Requirements Diagram".

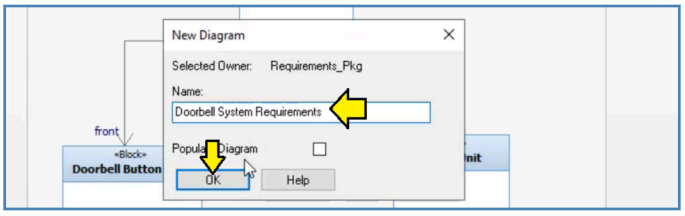


Figure 1-42 – Name the diagram

Name the new diagram "Doorbell System Requirements" and click "OK".

Now that we have created the requirements diagram, we can add a **requirement** to it. This operation is similar to adding a block to a block definition diagram.

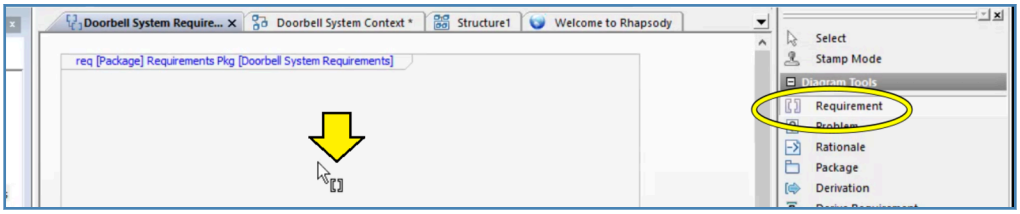


Figure 1-43 – Drag the requirement icon from the drawing toolbar to the diagram

Select the requirement icon from the drawing toolbar. Move the cursor over the diagram. Click to drop a default size requirement icon onto the requirements diagram.

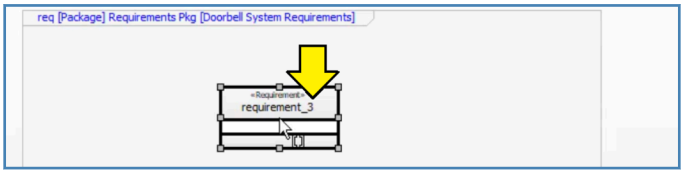


Figure 1-44 – Double-click to open the requirement

We now have a requirement element. Double-click on the element in the diagram to open the features dialog for the requirement.

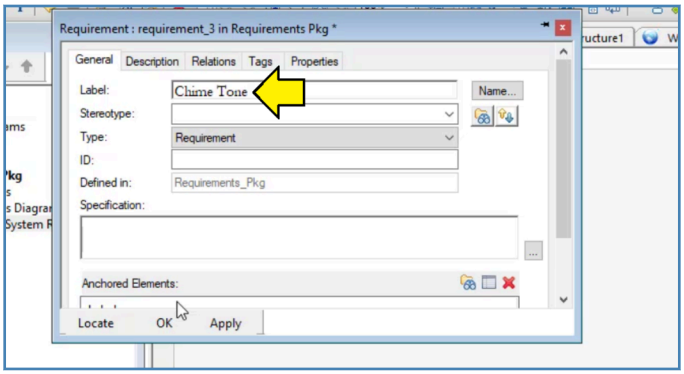


Figure 1-45 – Give the requirement a name

The first order of business is to give the requirement a **name**. Let's call this requirement: "Chime Tone". Enter the name as shown in Figure 1-45.

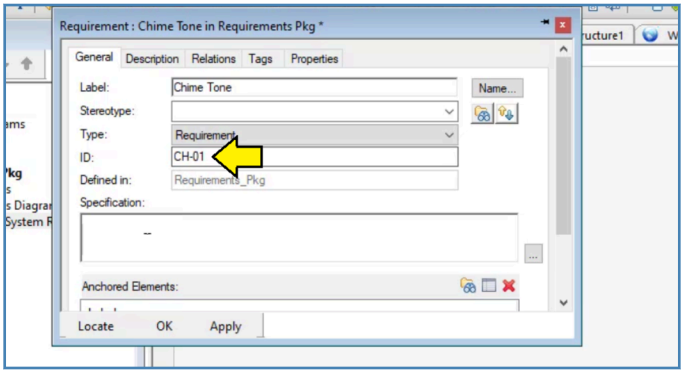


Figure 1-46 – Assign an ID

Enter an id as shown. The id can be any text string. For the moment, let's call this one "CH-01" for "First requirement related to the chime unit".

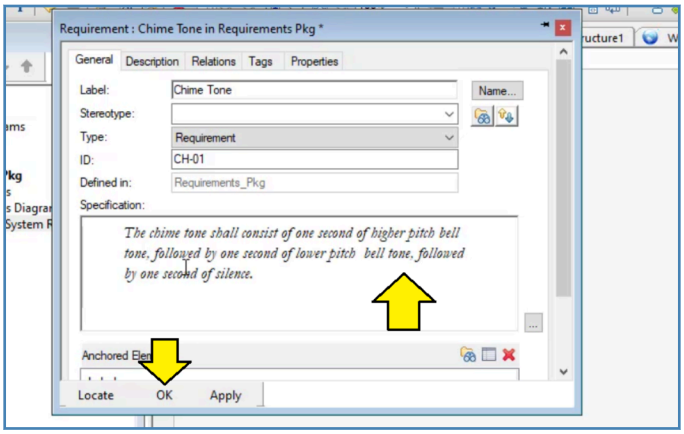


Figure 1-47 – Enter the text of the requirement

Enter the following text for the requirement:

> *The chime tone shall consist of one second of higher pitch bell tone, followed by one second of lower pitch bell tone, followed by one second of silence.*

When you are done, click "OK".

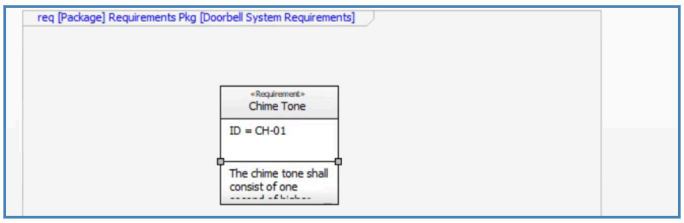


Figure 1-48 – By default the diagram does not show id or text

After you click "OK" your diagram may look like Figure 1-48 with the requirement text chopped off. In fact, if you accepted the default shape for the requirement element, it may look something like 1-44 with the id and the text not showing at all. In order to make the id and text show up properly, right-click on the requirement and select "Expand to fit text".

Tip - Resizing the Diagram
What if you want to zoom the diagram in or out? You can resize any diagram and zoom in or out by holding down the control key and rolling the wheel on your mouse. Actually, there are two motions (which are common to some other graphical software packages):

- control key + mouse wheel = zoom
- (nothing) + mouse wheel = move up or down

F6 is also very useful. F6 will scale the diagram to fit the contents to the space in the pane.

Have you ever visited someone's house and been frustrated because it wasn't clear whether the doorbell button was really doing anything or not? Our marketing department has identified this weakness in traditional doorbell design as an opportunity for product differentiation. The marketing department has developed this brief **user story** to describe the desired behavior for the doorbell system:

1) As the visitor approaches the door, the doorbell button will be glowing dimly.
2) After the visitor pushes the button, the chime will sound inside, and the doorbell button will begin flashing brightly on and off.
3) When the chime is finished, the doorbell button will return to the glowing dimly state. [6]

Starting with marketing's user story, our requirements engineering team has developed the following small set of **requirements** for our doorbell system:

[6] The observant reader will notice that we have just created the beginnings of a "user story" for our system. User stories are an important part of almost any design methodology.

ID	Name	Description
CH-01	Chime Tone	The chime tone shall consist of one second of higher pitch bell tone, followed by one second of lower pitch bell tone, followed by one second of silence.
CH-02	Chime Cycle	The chime cycle shall consist of three chime tones.
CH-03	Chime Cycle After Button Press	When the doorbell button is pressed, the chime unit shall complete one chime cycle.
BT-01	Flash During Chime	While the chime cycle is in progress, the doorbell button shall flash.
BT-02	Flash Cycle	The button flash cycle shall be 0.5 second bright illumination followed by 0.5 second of no illumination.
BT-03	Idle Dim Glow	When the doorbell system is idle, the doorbell button shall glow dimly.

Figure 1-49 – Requirements for the doorbell system

Go ahead and repeat the procedure and add the remaining five requirements to the model. In order to save you time and effort in typing, we have provided a spreadsheet of these requirements in the example files in the directory *Examples\Ch02_Quick_Start\S2.5_Req_Diag*.

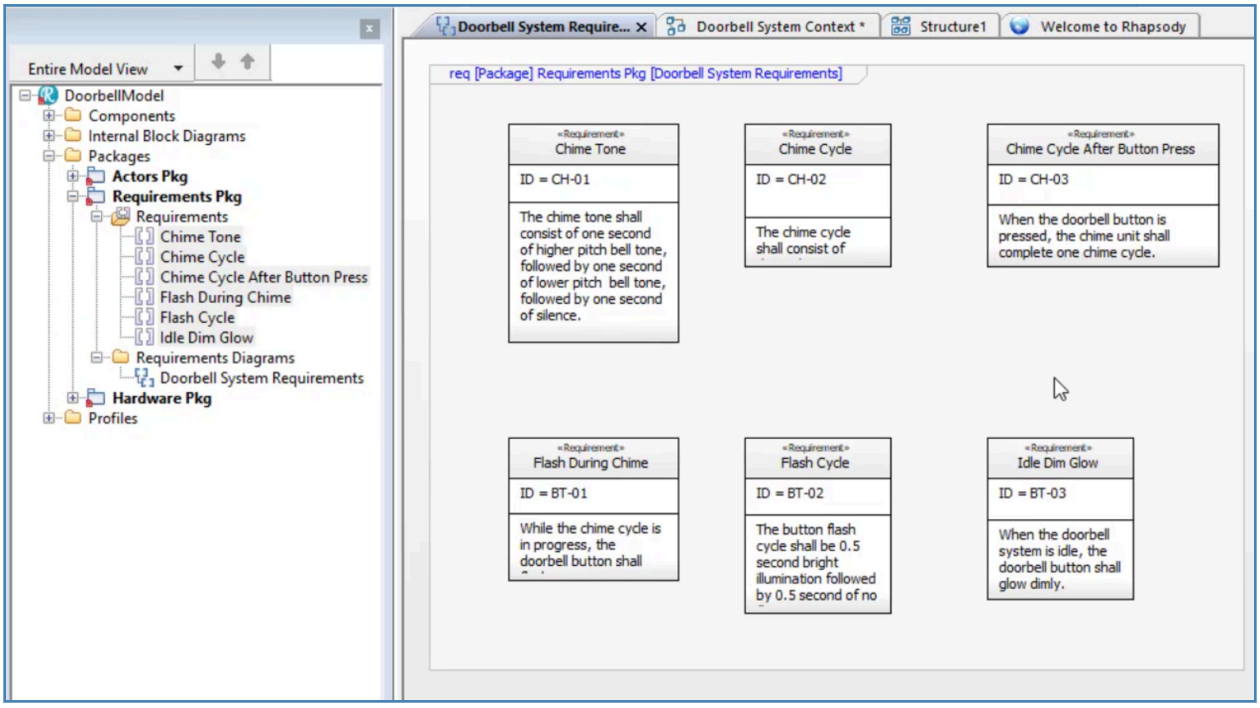


Figure 1-50 – Completed requirements for the doorbell system

When you are done adding the requirements, your requirements diagram should look like Figure 1-50.

So far, we have shown how to get a set of requirements into the system and how to place them on a diagram, but we really haven't yet shown the compelling benefit of integrating requirements into a graphical system model. In order to show this benefit, create a second requirements diagram called "Button Glow Detail".

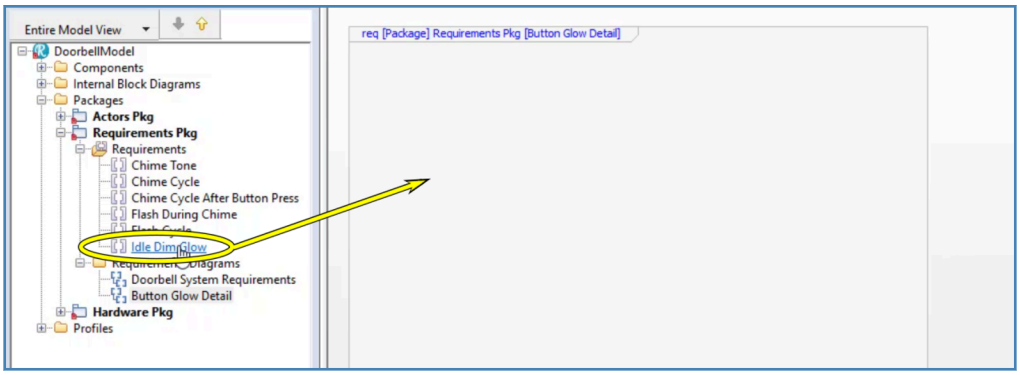


Figure 1-51 – Drag the idle dim glow requirement to the diagram

Drag the "Idle Dim Glow" requirement to the diagram.

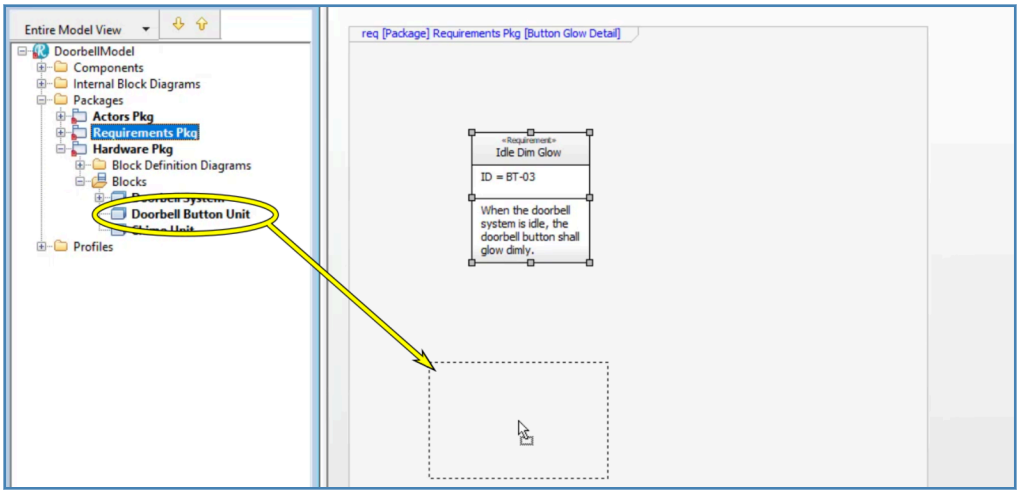


Figure 1-52 – Drag the doorbell button unit block to the diagram

Next drag the "Doorbell Button Unit" block to the diagram.

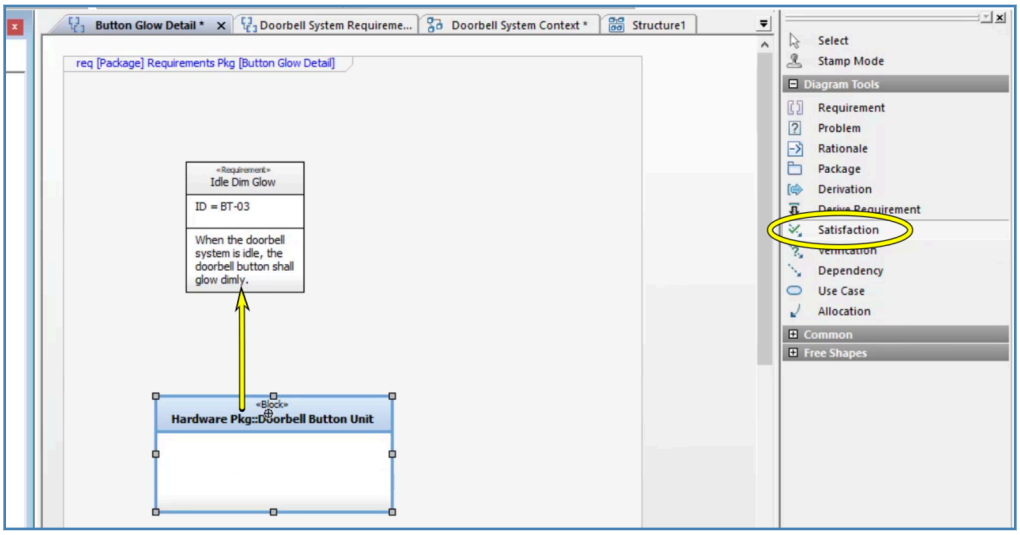


Figure 1-53 – Select satisfy and drag relationship

In the drawing toolbar select the "Satisfaction" icon and drag a **satisfy** relationship from the button unit to the requirement.

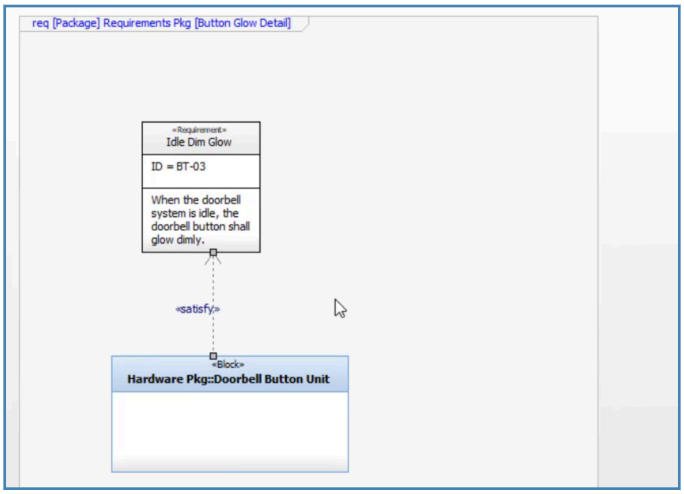


Figure 1-54 – Completed satisfy relationship

When you are done adding the satisfy relationship, your requirements diagram should look like Figure 1-54. This diagram can be understood as meaning: "The doorbell button unit is responsible for satisfying the idle dim glow requirement."

Quickly examining the drawing toolbar, the reader will notice that there are several more descriptive relationships. In the chapter on requirements diagrams, we will go into more detail about how these relationships can be used to make helpful diagrams that help stakeholders understand complex relationships between requirements, system elements, and test cases.

Direction of Arrows

When first shown a diagram like Figure 1-54, many engineers will feel like the arrow is pointed in the wrong direction. Such engineers are used to thinking of requirements as flowing downward from top to bottom. SysML inherits its arrow direction philosophy from UML. One point about UML (and SysML) is that arrows are navigated in the direction that they point. Hence, the requirement relationship arrows are "owned" by the elements that need to satisfy the requirements. This arrangement makes it slightly easier later to answer the question: "Which requirements does this block need to satisfy?"

Adding a Sequence Diagram

In the preceding sections, we have taken a quick look at diagrams for structure and for requirements. The last thing we will do in this chapter is add a diagram that describes system behavior. SysML defines three behavioral diagrams:

- Activity diagram
- Sequence diagram
- State machine diagram

In this section, we will create a simple **sequence diagram** for our doorbell system. Although it is possible to add behavior diagrams to a package, the intent of the creators of SysML was that behavior diagrams would show the behavior of a specific model element, usually a block or a use case. We will add our sequence diagram to the "Doorbell System" block.

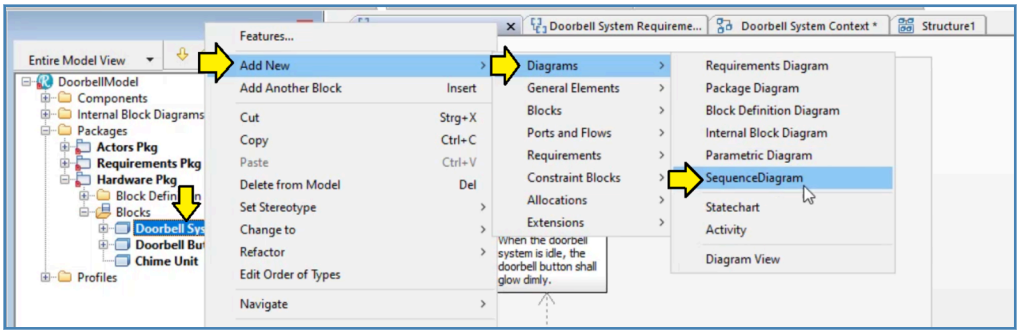


Figure 1-55 – Add sequence diagram to doorbell system

Right-click on the block "Doorbell System" and then select "Add New" followed by "Diagrams" followed by "SequenceDiagram".

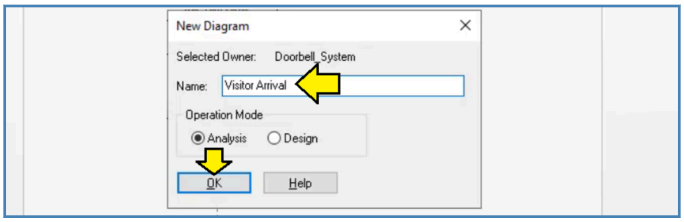


Figure 1-56 – Name the diagram

Give the diagram a name. The sequence diagram is actually the internal representation of a model element known as an **interaction**. As such, it makes sense to name a sequence diagram with a noun phrase that represents an interaction or activity. Name this diagram: "Visitor Arrival". Click "OK" when you are done. [7]

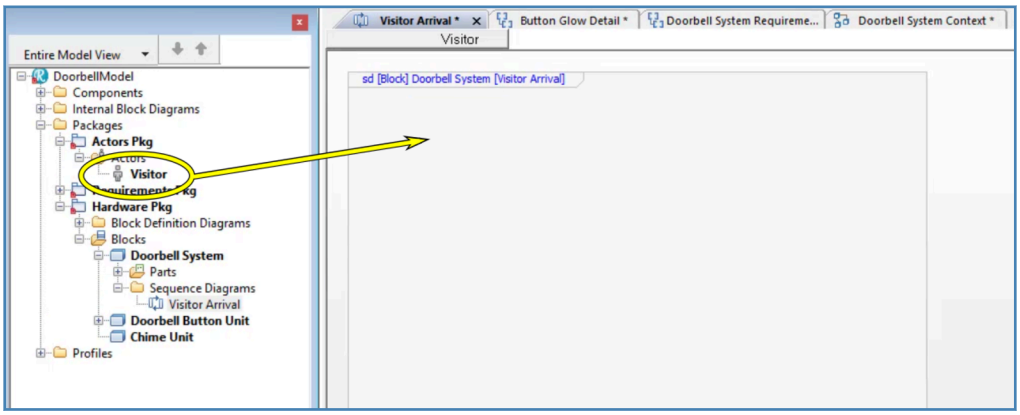


Figure 1-57 – Drag visitor to diagram

Drag the visitor from the browser to the diagram.

[7] *Rhapsody* does not actually support the creation of an explicit interaction element. Nevertheless, the naming approach is still recommended.

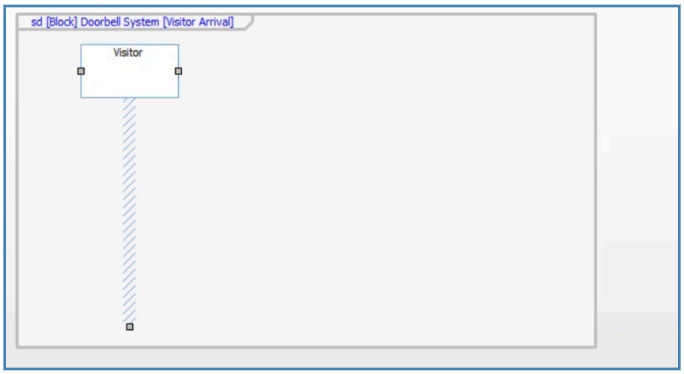


Figure 1-58 – Lifeline

After you click "OK" *Rhapsody* will create something called a **lifeline** for the visitor element.

- At the top is the element itself.
- Underneath the element is a cross-hatched bar. The cross-hatched bar indicates the passage of time in the "life" of the element, starting at the top and progressing downward in time.

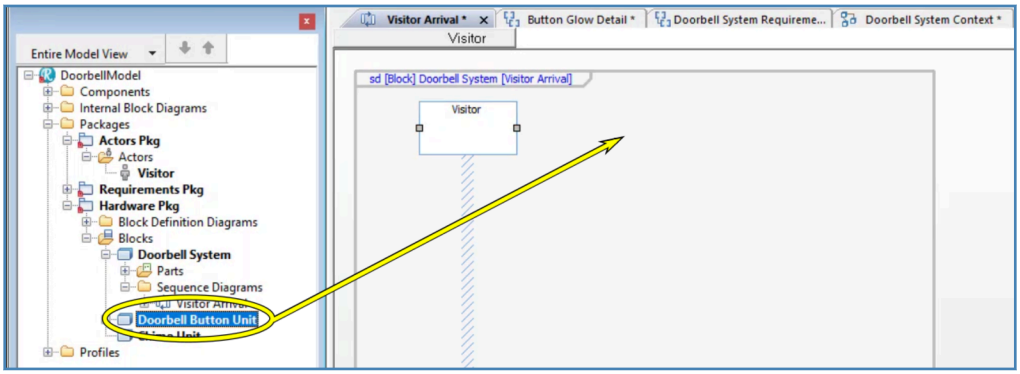


Figure 1-59 – Drag doorbell button to diagram

Next drag the block named "Doorbell Button Unit" from the browser to the diagram.

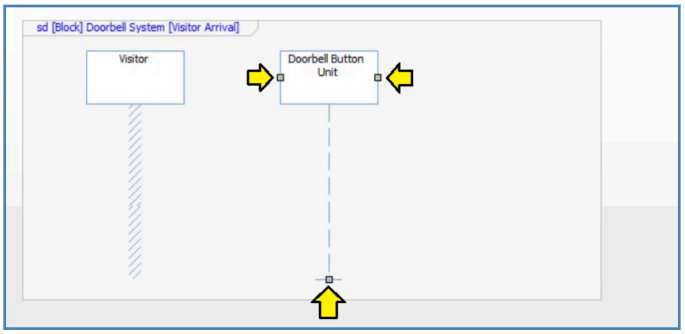


Figure 1-60 – Drag boxes to resize lifeline

The lifeline for a block looks a little different than the lifeline for the actor. We will look at that more carefully in the chapter on sequence diagrams. Notice that you can resize the lifeline by dragging the boxes around as shown in Figure 1-60.

Drag the block named "Chime Unit" to the diagram as well.

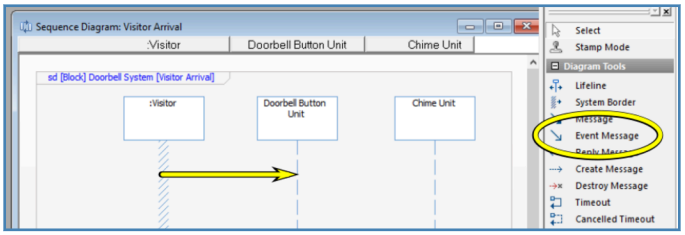


Figure 1-61 – Drag message from visitor to button

We now have three lifelines for three elements. Now we are ready to start creating the actual interaction between these three elements. In the drawing toolbar you will find two main **message** icons:

- **solid arrowhead –** "Message" is a synchronous message
- **open arrowhead –** "Event Message" is an asynchronous message

Unless we expect the visitor to stand with finger attached to the button until the doorbell chime is complete, we probably don't want to model this message as a synchronous message. An asynchronous message will be more appropriate. Select "Event Message", click on the lifeline for the visitor, and drag rightward to draw a message arrow from the visitor to the doorbell button unit.

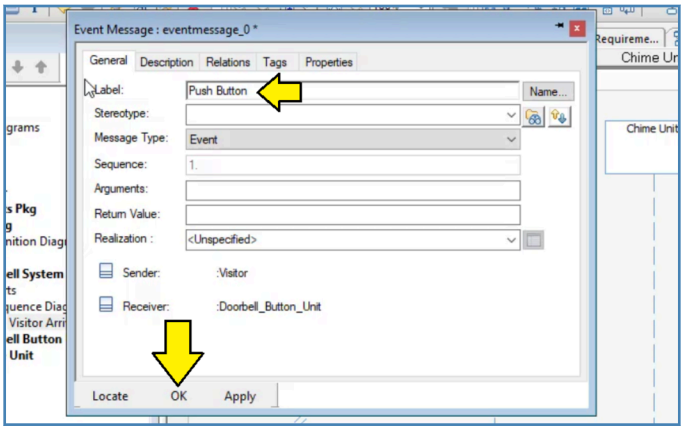


Figure 1-62 – Name message

Double-click on the message arrow to open the properties window. Name the message: "Push Button". Click "OK".

Now let's go ahead and add some additional asynchronous messages to complete the modeling of our story:

1) Add a message called "Button Pushed" from the button to the chime.
2) Add a message called "Chime Starting" from the chime to the button. Later, as we add more detail to the model, this will become the trigger to make the button begin flashing brightly.
3) Add a message called "Button Flashing" from the button to the visitor. The flashing is just a visual indication, but it is a message nonetheless and we can model it as an asynchronous message.
4) Add a message called "Chime Ending" from the chime to the button. This will become the trigger to return the button to "dull glow" mode.
5) Add a message called "Button Dull Glow" from the button to the visitor. The sudden absence of flashing is itself a message and can be modeled as such.

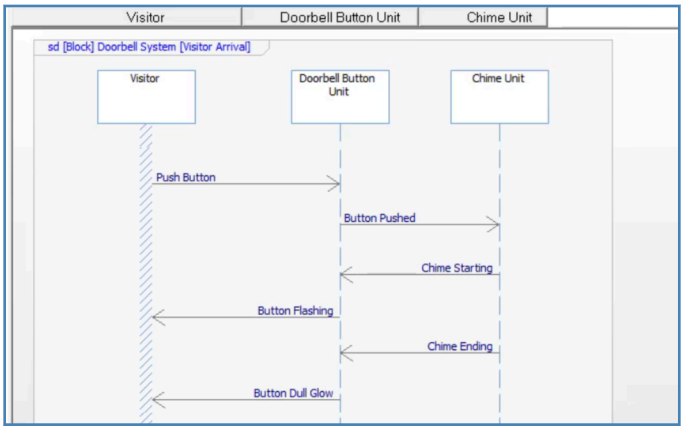


Figure 1-63 – Finished sequence diagram

Your finished sequence diagram should look like Figure 1-63.

This concludes our quick look at SysML and at the *Rhapsody* tool. In the next chapter, we will cover some general topics and questions before we start looking at each of the nine SysML diagram types in more detail.