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 model. Don't worry too much if the *Arcadia* concepts presented in this chapter seem strange; we will be 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 Capella

Since *Capella* is a Java program that does not come with a Windows installation program, you will need to create a desktop shortcut for it.

If you have not yet created such a desktop shortcut yet, you can do so by dragging the tool's executable file the the desktop. In the *Capella* installation directory, you will find a subdirectory “capella”. In this subdirectory, find the file “capella.exe”. Right-click on this file and drag it to the desktop. When you release the right mouse button, a context menu will appear that will allow you to create a shortcut.

As soon as you start the tool, it will prompt you to enter a workspace for your project files. There is some complexity involved in selecting a workspace location, especially if you are going to use a source code control tool like Git. More information is available in the installation whitepaper available here:

https://url4ap.net/Capella-DL-Install

Double-click on the desktop icon to start the tool. A “Capella Launcher” window will appear which will prompt you to select a directory for the *Eclipse* workspace. *Capella* manages only a single workspace at once, but it is possible to switch between several workspaces.

Figure 1-1 – Capella icon

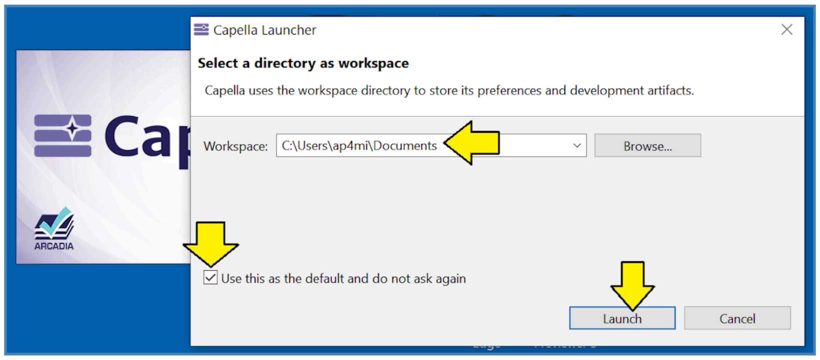

Capella Launcher
Select a directory as workspace
Capella uses the workspace directory to store its preferences and development artifacts.
Workspace: C:\Users\ap4mi\Documents
Browse...
Use this as the default and do not ask again
Launch
Cancel

Figure 1-2 – Select workspace

Enter the directory where you would like new projects stored as mentioned above.

If you will be using the same directory every time, you will want to select the checkbox to make that directory the default and not ask again.

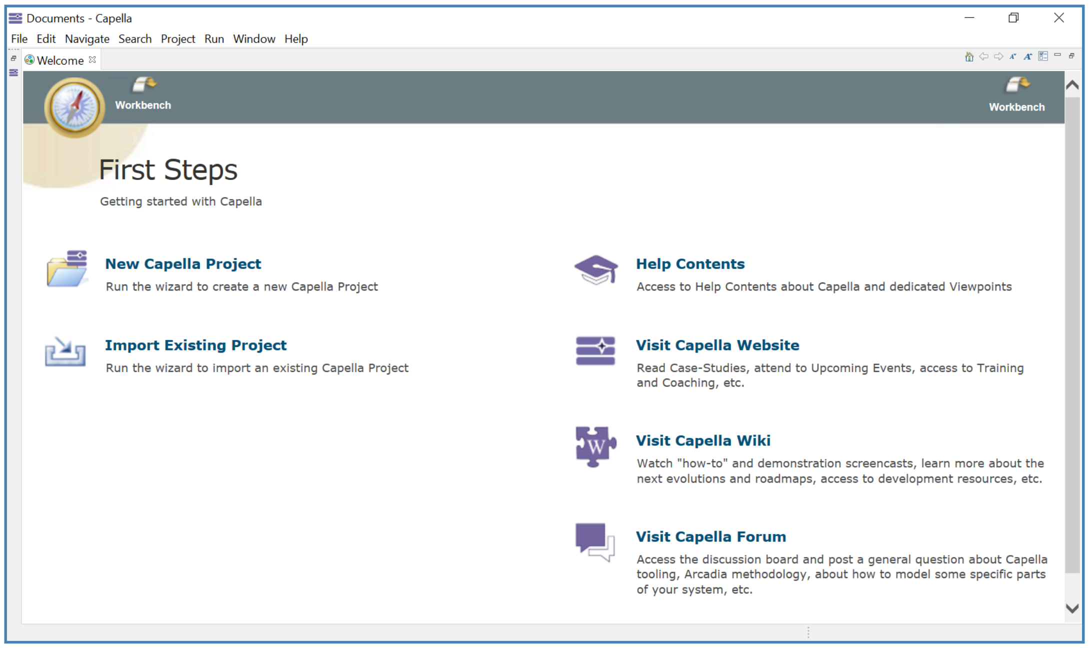

Documents - Capella
File Edit Navigate Search Project Run Window Help
Welcome
Workbench
Workbench
First Steps
Getting started with Capella
New Capella Project
Run the wizard to create a new Capella Project
Import Existing Project
Run the wizard to import an existing Capella Project
Help Contents
Access to Help Contents about Capella and dedicated Viewpoints
Visit Capella Website
Read Case-Studies, attend to Upcoming Events, access to Training and Coaching, etc.
Visit Capella Wiki
Watch "how-to" and demonstration screencasts, learn more about the next evolutions and roadmaps, access to development resources, etc.
Visit Capella Forum
Access the discussion board and post a general question about Capella tooling, Arcadia methodology, about how to model some specific parts of your system, etc.

Figure 1-3 – Welcome screen

The welcome screen presents a handy list of information for beginners. On the left, there are commands to either create a new project or to import an existing project. On the right, there are links to help files and other useful online content.

Creating a Model

Note: if you would prefer to load the example file rather than creating a new model, you will need to import the example model from its "*.zip" archive file. *Capella* does not a simple "file open" operation. For more information see the section: *Archiving and Importing Capella Projects* in the installation whitepaper available here:

https://url4ap.net/Capella-DL-Install

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

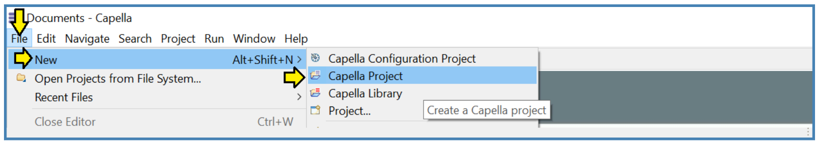
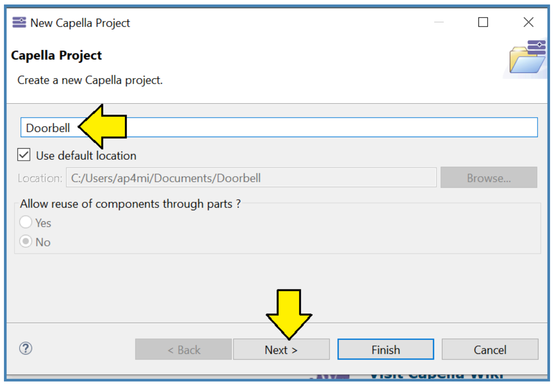


Figure 1-4 – Create a project

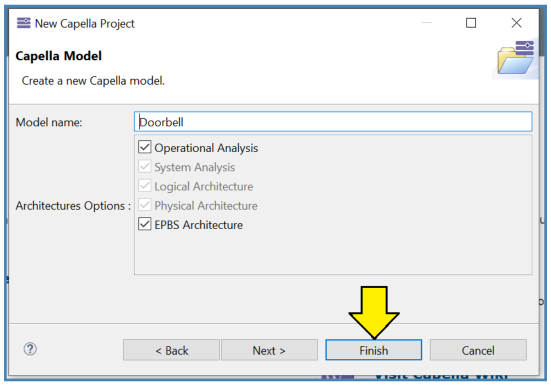
From the main menu, select "File", then "New", and then "Capella Project".



Doorbell
☑ Use default location
Location: C:/Users/ap4mi/Documents/Doorbell Browse...
Allow reuse of components through parts ?
Yes [✓ No]

Figure 1-5 – Name the project

Our first project will be a doorbell system. Name the project: "Doorbell" and click "Next".

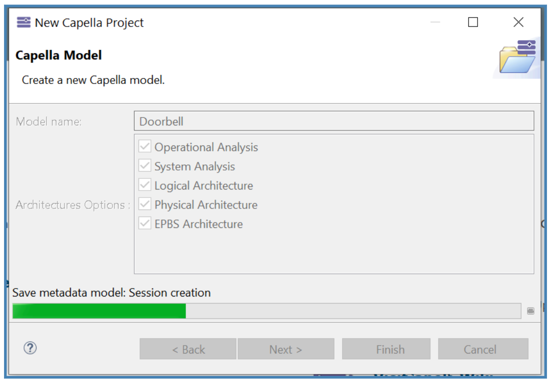


Model name: Doorbell
Architectures Options :
☑ Operational Analysis
☑ System Analysis
☑ Logical Architecture
☑ Physical Architecture
☑ EPBS Architecture

Figure 1-6 – Select options

The next panel allows you to select which layers of the *Arcadia* method you want to include in your model. For the moment, accept the defaults and click "Finish".

This step will create the subdirectory for the model (only) within the workspace folder which contains the model but can also contain other information.



Capella Model

Create a new Capella model.

Model name: Doorbell

Architectures Options :
- ✓ Operational Analysis
- ✓ System Analysis
- ✓ Logical Architecture
- ✓ Physical Architecture
- ✓ EPBS Architecture

Save metadata model: Session creation

Figure 1-7 – Model progress indicator

A model progress indicator will appear briefly.

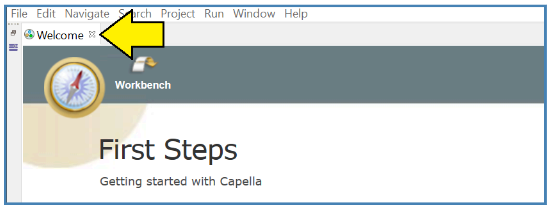
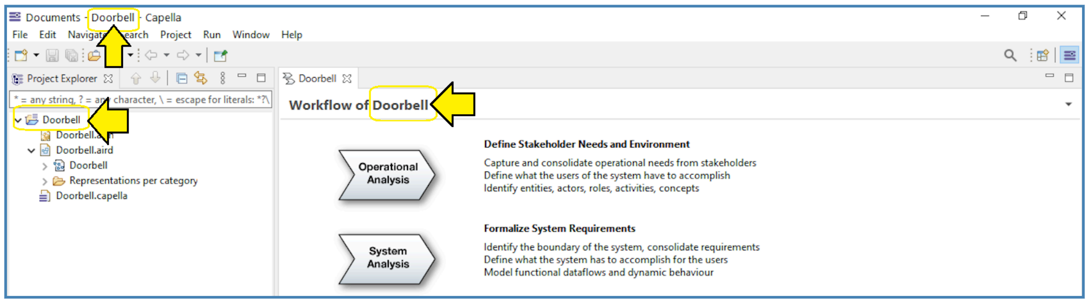

File Edit Navigate Search Project Run Window Help
Welcome
Workbench
First Steps
Getting started with Capella

Figure 1-8 – You may need to dismiss the welcome screen

Where is my model?! You may need to dismiss the welcome screen. Click the small "x" at the right of the "Welcome" tab.


Documents - Doorbell - Capella
File Edit Navigate Search Project Run Window Help
Project Explorer
* = any string, ? = any character, \ = escape for literals: *?\
Doorbell
Doorbell.afm
Doorbell.aird
Doorbell
Representations per category
Doorbell.capella
Doorbell
Workflow of Doorbell
Operational Analysis
System Analysis

Define Stakeholder Needs and Environment

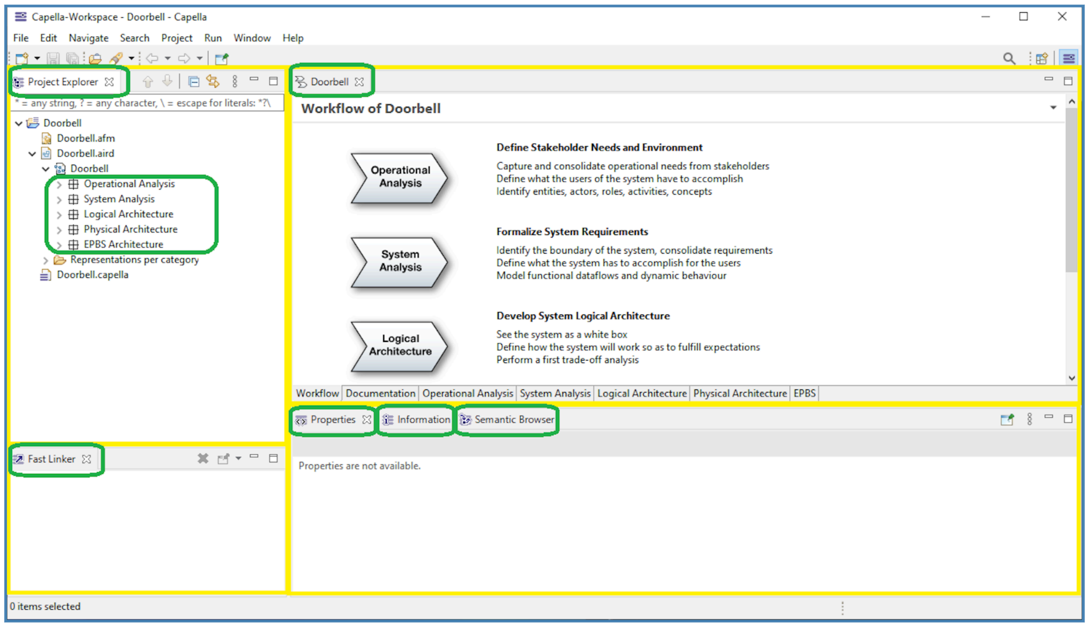
Capture and consolidate operational needs from stakeholders
Define what the users of the system have to accomplish
Identify entities, actors, roles, activities, concepts

Formalize System Requirements

Identify the boundary of the system, consolidate requirements
Define what the system has to accomplish for the users
Model functional dataflows and dynamic behaviour

Figure 1-9 – The doorbell model appears

The doorbell model appears. Notice that the model has been named to match the model file name.



* = any string, ? = any character, \ = escape for literals: *?\

Workflow of Doorbell

Define Stakeholder Needs and Environment
Capture and consolidate operational needs from stakeholders
Define what the users of the system have to accomplish
Identify entities, actors, roles, activities, concepts

Formalize System Requirements
Identify the boundary of the system, consolidate requirements
Define what the system has to accomplish for the users
Model functional dataflows and dynamic behaviour

Develop System Logical Architecture
See the system as a white box
Define how the system will work so as to fulfill expectations
Perform a first trade-off analysis

Properties are not available.

Figure 1-10 – Default tool layout

Capella is based on the *Eclipse* platform. As such, windows and views can generally be rearranged to suit your personal taste. However, let's take a closer look at the default view of the tool.

1) In the top-left corner of the screen, we can see a pane called the **Project Explorer**. This pane presents a hierarchical view of the model than can be expanded and explored similar to the file explorer user interfaces in Windows and other operating systems. In the project explorer, we see that the tool has already created 5 packages mapping to the Arcadia levels.

2) In the center of the screen, we can see a big **Editor** pane containing the **Activity Explorer** (in our case, named: "Workflow of Doorbell") and the future diagrams. The activity explorer contains icons for the different *Arcadia* levels and will be the entry point for the creation of diagrams. The editor pane is a tabbed view layout that will show diagrams as tabs as we create them.

3) There are three predefined tabs in the bottom pane: **Properties**, **Information**, **Semantic Browser**.

   - The properties view displays the features of the model element currently selected in the project explorer or in a diagram. You can also use the properties view to display the features of a diagram by selecting the diagram in the project explorer or by clicking in the background of the diagram itself.
   - The information view displays model validation results and miscellaneous additional messages, validation traces, and similar console output.
   - The semantic browser displays all relationships entering and leaving the model element currently selected in the project explorer or in a diagram.

4) In the bottom-left corner of the screen, we can see a pane called the **Fast Linker**. In this book, we will not be making much use of this pane. Generally, we close it to make more room for the project explorer.

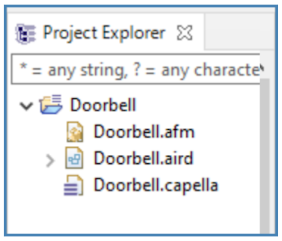
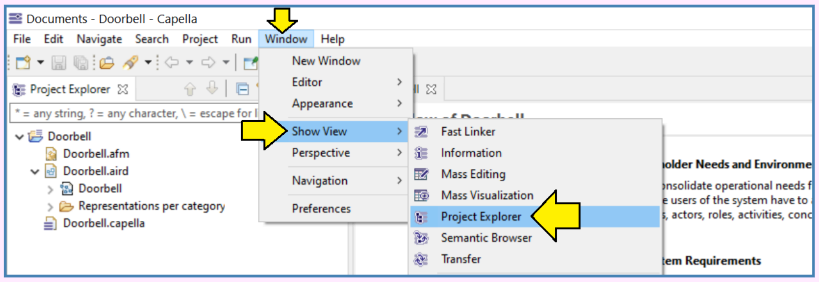


Figure 1-11 – Closer Look at the Project Explorer

Taking a closer look at the project explorer, we can see that it presents information as the contents of three files:

- **"*.afm" –** The *.afm file is an eclipse generated file and cannot be browsed from the tool.
- **"*.capella" –** The *.capella file contains the actual semantic model elements. However, this file cannot be browsed from the tool.
- **"*.aird" –** The *.aird file contains the representations (views of the model, through diagrams or tables) and is the only file we can browse and explore.

What if the Project Explorer isn't there?
What if the project explorer isn't visible on your screen? This kind of thing happens all the time with this sort of highly configurable tool.



This problem can be solved by selecting "Window", then selecting "Show View" and then selecting "Project Explorer".

As mentioned previously, *Capella* is based on the *Eclipse* platform. At this point we will introduce some more precise terminology related to the underlying *Eclipse* platform:

1) The default layout shown in Figure 1-10 on page 5 is actually an *Eclipse* **perspective**.

2) *Capella* consists of one single *Eclipse* perspective, which is loaded by default when *Capella* is started.

3) The perspective defines the initial set of actions and panes that appears in the **workbench** window.

4) A perspective is a group of **views** and **editors** in the workbench window.

5) A view is a visual component within the workbench. It is typically used to navigate a list or hierarchy of information (such as the resources in the workbench) or display properties for the active editor. It can support editors and provides information and alternative presentation. Modifications made in a view are saved immediately if the view is not attached to an editor. For example, the project explorer and other navigation views display projects and other resources the user is working with, while the information view displays information that provides feedback to several *Capella* "complex" processes.

6) An editor is also a visual component within the workbench. It is typically used to edit or browse a resource. Editors are launched by (double-)clicking on a resource in a view. Modifications made in an editor follow an open-save-close model.

How do I Restore the Default Layout?
In the normal course of using *Capella*, views will be opened, moved, resized, and closed. The default layout can be restored with the "Window, Perspective, Reset Perspective" menu operation.

Understanding the Arcadia Template

In fact, the new model shown in Figure 1-9 is not really empty. Because we accepted the default options in Figure 1-6 on page 3, the model is already laid out with a structure matching the five *Arcadia* levels. Subfolders have been created automatically, and some model elements have been predefined.

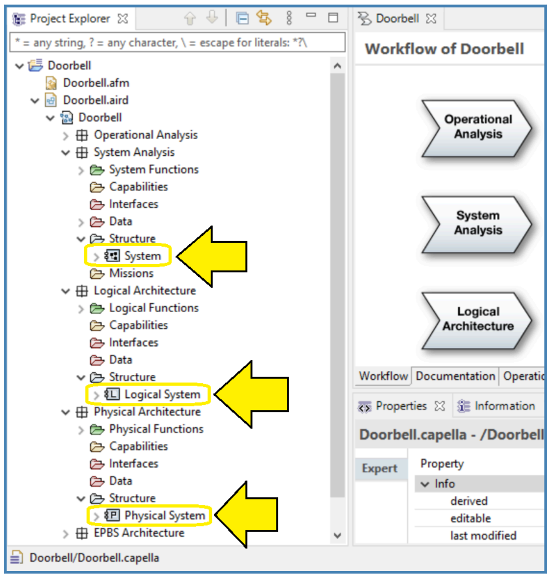


Figure 1-12 – Predefined elements

For example:

- A **system** has been created at the System Analysis level.
- A **logical system** has been created at the Logical Architecture level.
- A **physical system** has been created at the Physical Architecture level.

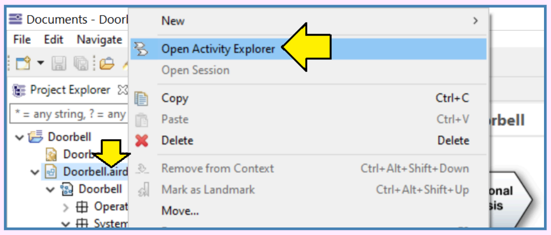
Depending on the options selected, the tool will create other types of elements as well.

Our First Glimpse of the Five Arcadia Levels

- **Operational Analysis –** "What the users of the system need to accomplish". Operational analysis is quite similar to business process modeling. At this level, the system is not (yet) recognized as a model element.
- **System Analysis –** "What the system has to accomplish for the users". The system is seen as a black box containing no other structural elements, only allocated functions and interactions with external users or systems.
- **Logical Architecture –** "How the system will work to fulfill expectations". The logical architecture identifies logical components inside the system, the relationships between the logical components, and the content of the logical components. The logical architecture is implementation and technology independent.
- **Physical Architecture –** "How the system will be developed and built". The physical architecture defines the final architecture of the system, and how the system must be implemented, taking into account technological choices.
- **End Product Breakdown Structure (EPBS) –** "What is expected from the provider of each component". The physical components are grouped into **configuration items** for the purposes of formal configuration management and supplier contracting.

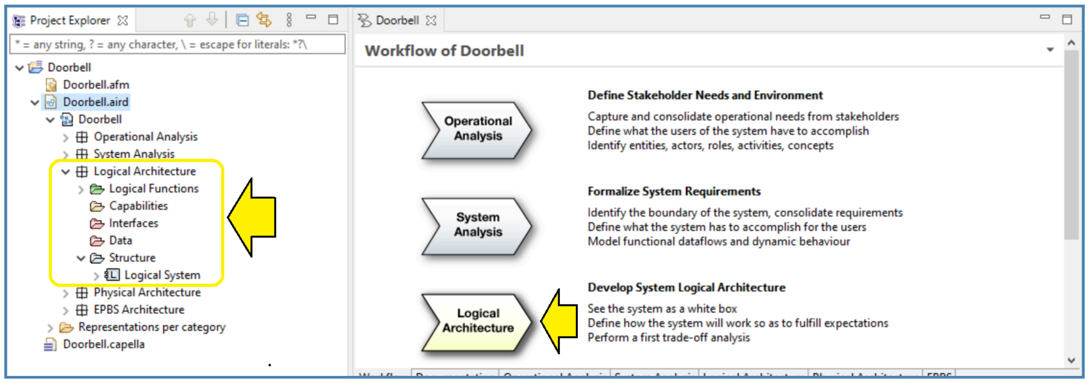
The activity explorer serves as the focal point for organizing *Arcadia* modeling activities. You will want to keep it open at all times.

What if the Activity Explorer isn't there?
What if the activity explorer isn't visible on your screen?



This problem can be solved by right-clicking on the *.aird file in the project explorer and selecting "Open Activity Explorer".

Arcadia contains five levels. Where should we start? *Arcadia* itself does not impose any particular sequence of modeling; it can be used top-down, bottom-up, or middle-out. For this "Quick Start" to get readers as familiar with the tool and method as rapidly as possible, we will start in the middle at the logical layer.



Define Stakeholder Needs and Environment
Capture and consolidate operational needs from stakeholders
Define what the users of the system have to accomplish
Identify entities, actors, roles, activities, concepts

Formalize System Requirements
Identify the boundary of the system, consolidate requirements
Define what the system has to accomplish for the users
Model functional dataflows and dynamic behaviour

Develop System Logical Architecture
See the system as a white box
Define how the system will work so as to fulfill expectations
Perform a first trade-off analysis

Figure 1-13 – Expand Logical Architecture

In the project explorer collapse all layers except the logical architecture layer. Move the mouse over to the activity explorer and hover over the "Logical Architecture" icon. The icon will turn pale yellow/green.

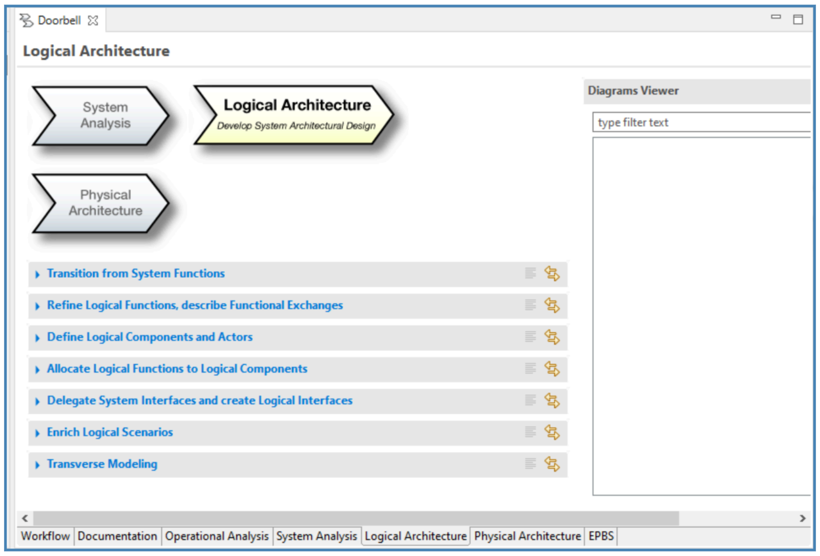

Doorbell
Logical Architecture
System Analysis
Logical Architecture
Develop System Architectural Design
Physical Architecture
Diagrams Viewer
type filter text
Transition from System Functions
Refine Logical Functions, describe Functional Exchanges
Define Logical Components and Actors
Allocate Logical Functions to Logical Components
Delegate System Interfaces and create Logical Interfaces
Enrich Logical Scenarios
Transverse Modeling
Workflow
Documentation
Operational Analysis
System Analysis
Logical Architecture
Physical Architecture
EPBS

Figure 1-14 – Logical architecture activities

If you click on the "Logical Architecture" icon the activity explorer will expand to display a list of methodology activities related to logical architecture.

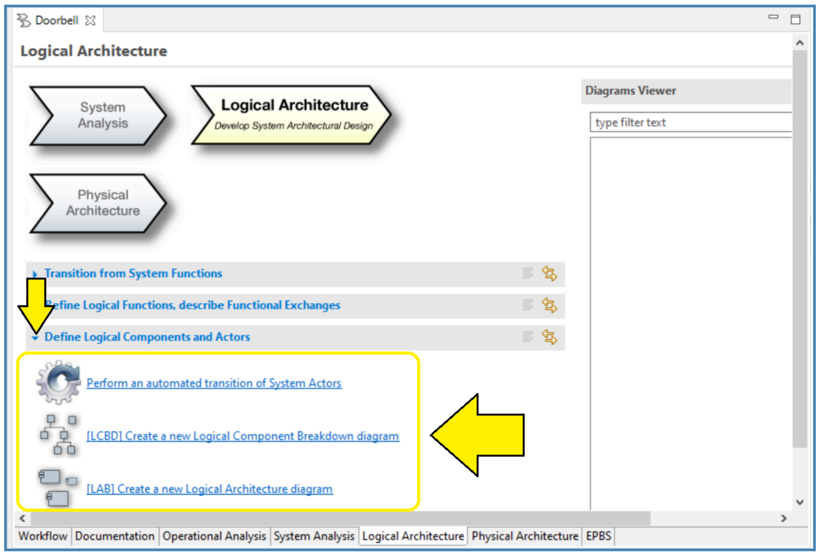

Doorbell
Logical Architecture
System Analysis
Logical Architecture
Develop System Architectural Design
Physical Architecture
Diagrams Viewer
type filter text
Transition from System Functions
Define Logical Functions, describe Functional Exchanges
Define Logical Components and Actors
Perform an automated transition of System Actors
[LCBD] Create a new Logical Component Breakdown diagram
[LAB] Create a new Logical Architecture diagram
Workflow
Documentation
Operational Analysis
System Analysis
Logical Architecture
Physical Architecture
EPBS

Figure 1-15 – Specific steps for an activity

If you select one of the activities, the activity explorer will expand to display a list of possible steps and diagrams to complete that activity. Note that the steps can be completed in any order.

Adding an Architecture Blank Diagram

Comparing Arcadia and SysML
If you are interested in comparing *Arcadia* and *Capella* with SysML, the doorbell system example we are working through in this chapter is very similar to the doorbell system example presented in the three *SysML for Beginners* books also published by Asatte Press. The quick start tutorials for those three books containing the SysML doorbell system examples are available from the Asatte Press website:

https://asattepress.com/Downloads/Tutorials.html

Now we are ready to add our first diagram. *Arcadia* defines many types of diagrams. The first diagram we will add is an **architecture blank diagram** – a very representative and important diagram for defining the elements of a system, as well as their functions and functional exchanges.

The main purpose of an architecture blank diagram is to show the allocation of **functions** to **components**. Architecture blank diagrams are used at all levels. We will be creating a “Logical Architecture Blank [LAB]” diagram.

As mentioned above, the purpose of logical architecture is to show “How the system will work to fulfill expectations”. At the logical architecture level, functions and components are independent of technology and implementation. As such, the diagram shows the allocation of **logical functions** to

logical components. At other architectural levels, the allocation might be to operational entities, the system, or other elements such as external actors.

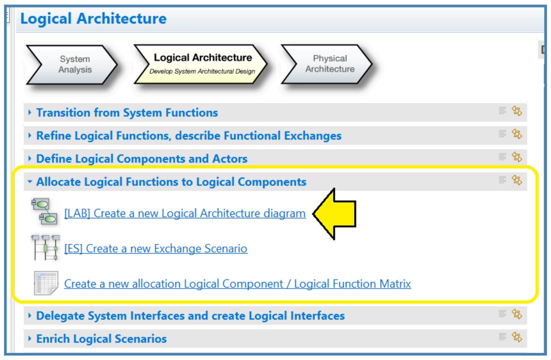
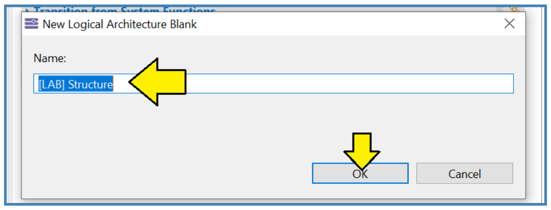

Logical Architecture
System Analysis
Logical Architecture
Develop System Architectural Design
Physical Architecture
Transition from System Functions
Refine Logical Functions, describe Functional Exchanges
Define Logical Components and Actors
Allocate Logical Functions to Logical Components
[LAB] Create a new Logical Architecture diagram
[ES] Create a new Exchange Scenario
Create a new allocation Logical Component / Logical Function Matrix
Delegate System Interfaces and create Logical Interfaces
Enrich Logical Scenarios

Figure 1-16 – Create a new Logical Architecture Diagram

In the "Logical Architecture" section of the expand the "Allocate Logical Functions to Logical Components" activity and select the "[LAB] Create a new Logical Architecture diagram" step.


New Logical Architecture Blank

Name: [LAB] Structure

Figure 1-17 – Accept default name

A panel will appear which will allow you to name the diagram. For the moment, just accept the default name and click "OK".

For a quick start exercise like this one, the default diagram name will be completely adequate. However, we have two suggestions regarding *Arcadia* diagram names:

1) Make the diagram names unique. That is, once you have more than one "[LAB]" diagram, take the time to give each one a unique, meaningful name.
2) Keep the prefix. *Arcadia* and *Capella* will allow you to name the diagrams anything you please. However, experienced users are used to the reading the prefix "[LAB]" and other similar *Arcadia* prefixes to quickly recognize diagram types.

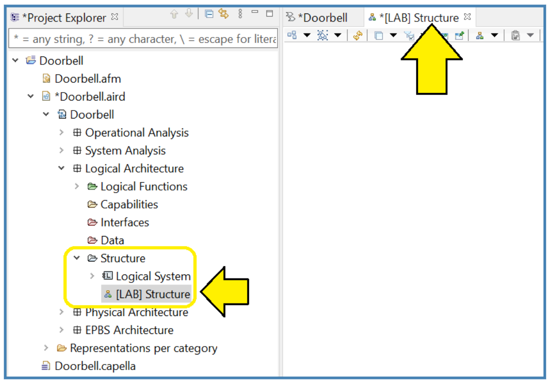


Figure 1-18 – Diagram appears in the Structure package

Capella will automatically place the new diagram in the "Structure" package.

The design of *Arcadia* and *Capella* is rather different than the design of SysML tools. Package names are predefined. Diagrams are automatically placed in predefined locations in the project explorer. Individual model elements are also mostly placed in predefined packages as well, with just a few exceptions that we will be introducing as we encounter them later in the book.

Initially, this approach using predefined packages may seem a little inflexible. The modeler may feel like his individual creativity is being suppressed. However, experience with many models created by many modelers in large organizations shows that modelers allowed to freely create package structures will tend to create a chaotic and bewildering variety of package structures. For large organizations, it is much more efficient to have all the models structured the same way so that everyone knows where to look for a certain type of information regardless of who created the model.

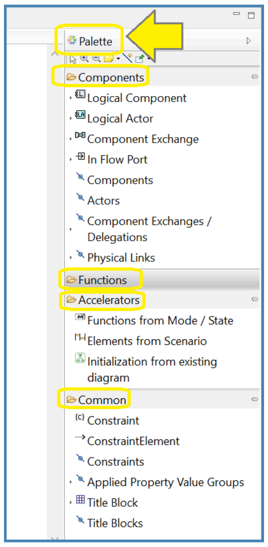
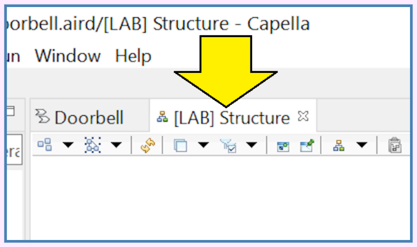


Figure 1-19 – The Palette

The newly created diagram is empty. To the right of the diagram, you will find the **Palette** which contains tools for creating content in the diagram.

Each diagram type has a specific palette. However, the layout is consistent across diagram types. There are several sections which can be expanded or collapsed as needed.

Full-Screen Mode



If you need more screen space to view or work on your diagram, you can simply double-click on the diagram tab to hide most of the windows. Double-clicking on the tab again restores the normal view.

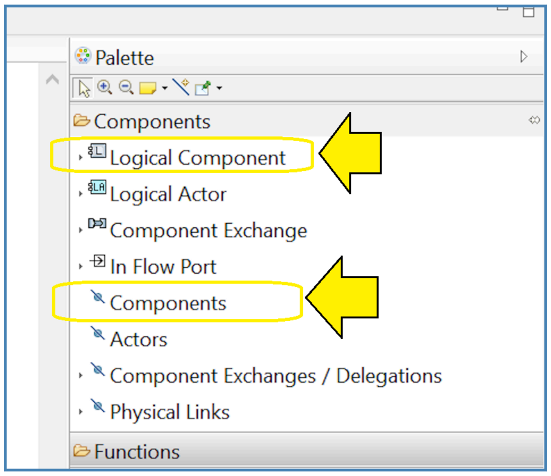
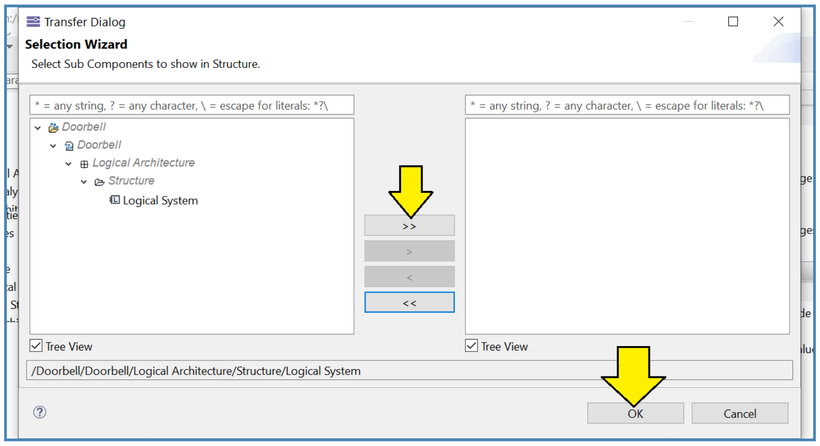


Figure 1-20 – Two kinds of component tools

Looking more closely at the palette we will find a section for components. In this section, there are two different types of tools for a component:

- **"Logical Component" –** This tool creates a new component in the model and also displays it in the diagram.
- **"Components" –** This tool does *not* create a new component in the model. This tool merely adds a graphical representation to the diagram for a component that already exists in the model.

Let's try out the "Components" tool first.



Selection Wizard

Select Sub Components to show in Structure.

* = any string, ? = any character, \ = escape for literals: *?\

- Doorbell
  - Doorbell
    - Logical Architecture
      - Structure
        - Logical System

* = any string, ? = any character, \ = escape for literals: *?\

/Doorbell/Doorbell/Logical Architecture/Structure/Logical System

Figure 1-21 – Selection Wizard

Select the "Components" tool in the palette and click in the diagram. The "Selection Wizard" will appear. *Capella* has predefined a default "Logical System" component which is visible in the left pane. Click the double arrow in the middle of the wizard to move it to the right pane (and hence into the diagram). Click: "OK".

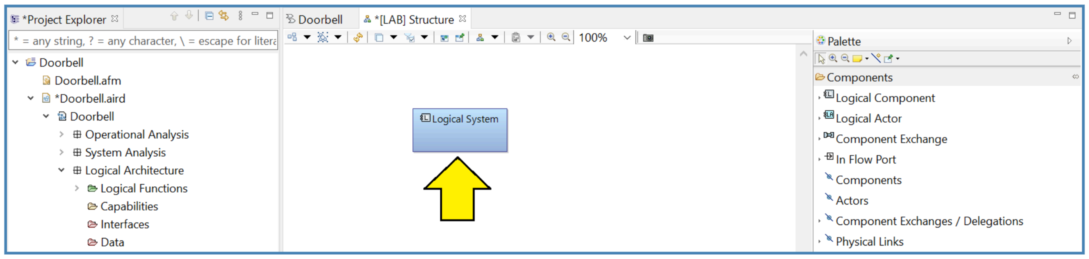


Figure 1-22 – The component appears in the diagram

The "Logical System" will appear in the diagram in the default color and size. The default color is fine, but this thing is our entire system. We are going to want to make it larger so we can put some other things inside of it.

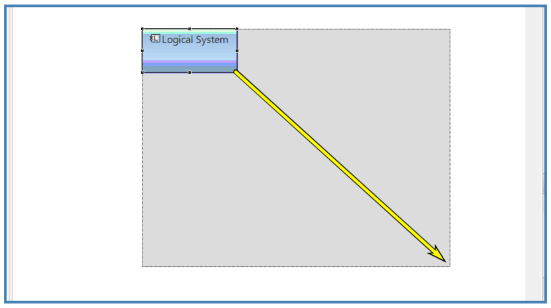


Figure 1-23 – Drag the corner to expand the component

Click to select the component. Drag one corner of the component to make it larger.

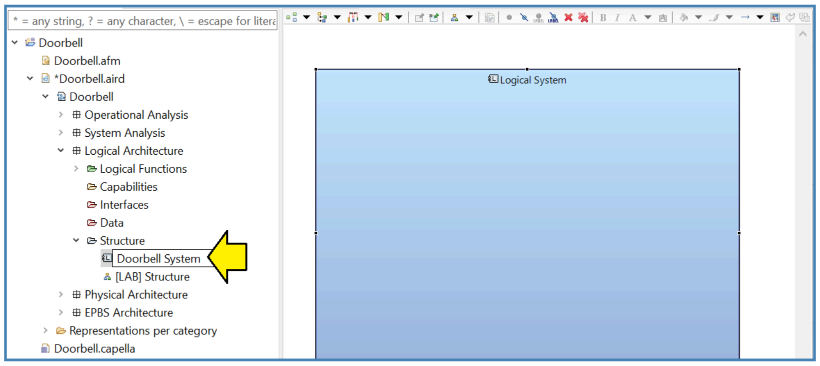


Figure 1-24 – Rename Logical System

Select the logical system in the project explorer, press "F2" and rename it: "Doorbell System".

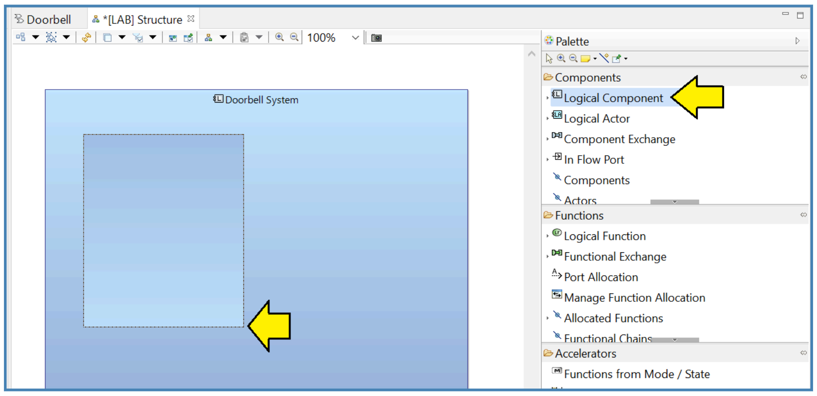


Figure 1-25 – Create nested logical component

Select the "Logical Component" tool in the palette, click in the diagram, and drag to make a larger outline.

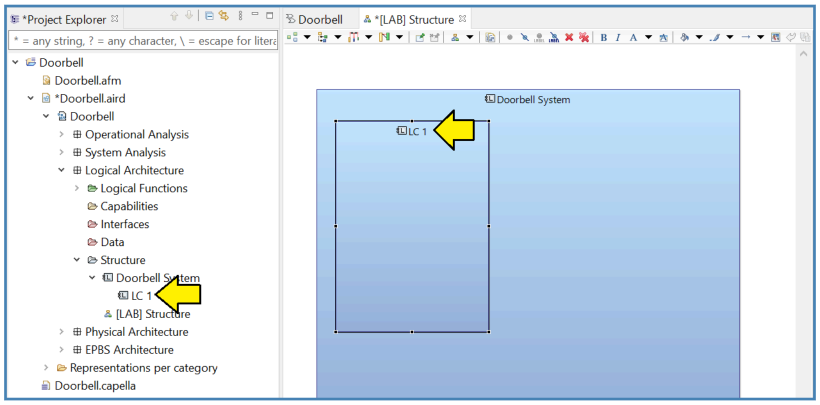


Figure 1-26 – LC1 has been created

Capella has created an element called: "LC1" in both the diagram and the project explorer. In other words, a new model element has been created.

A logical component is a structural element within the system, which can be connected with other logical components and external actors. A logical component can have one or more **Logical Functions** allocated to it. A logical component can also be sub-divided into sub-components.

Use the F2 key to rename "LC1" to "Doorbell Button Unit".

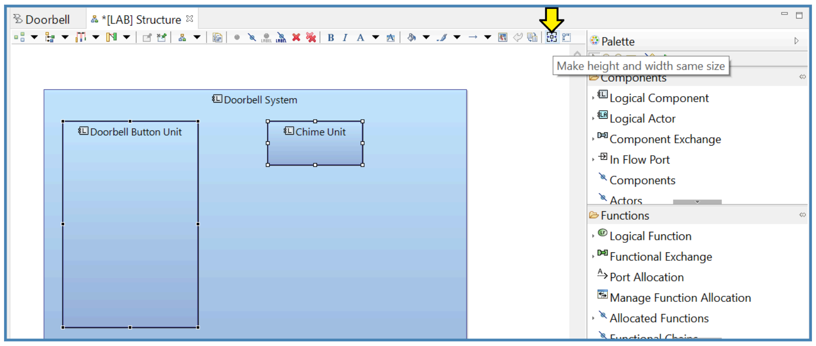


Figure 1-27 – Make Chime Unit same size

Create another logical component called: "Chime Unit". After you create the new component, leave it selected and the doorbell button unit component as well. Near the right of the toolbar along the top of the diagram, you will find an icon whose purpose is to make things the same size. Click this icon.

Notice that the order in which you select things is important. The last item selected will be the reference for the other selected item when you use an alignment or sizing function.

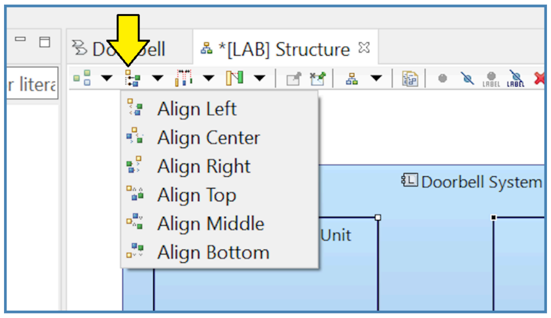


Figure 1-28 – Alignment tool

There is also a tool for aligning elements as well as another tool next to it for spacing elements.

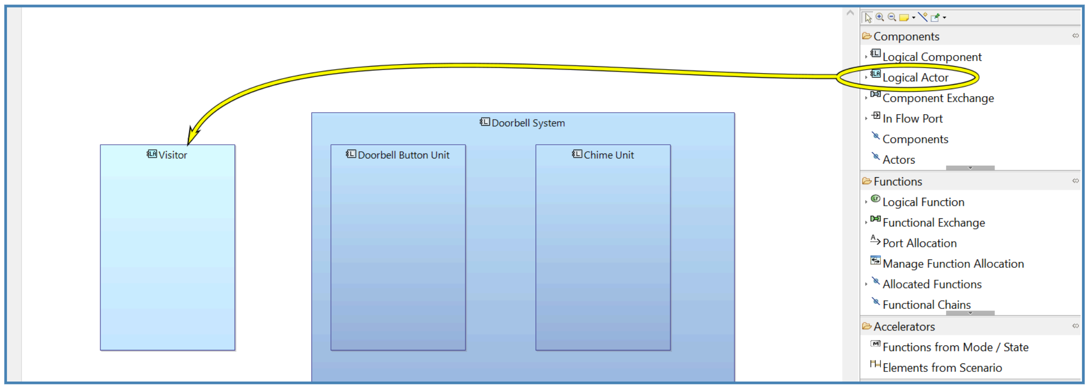


Figure 1-29 – Add visitor

Next, we can add a **Logical Actor** to the diagram. Select the "Logical Actor" tool in the palette and click in the diagram. Name the new element: "Visitor". Make it the same size as the doorbell button and chime units. External actors are always light blue, by default to distinguish them from internal components.

We can now add logical functions inside the logical components and the logical actor.

A logical function represents a behavior or service provided by a logical component or by a logical actor. A logical function owns **Function Ports** that allow it to communicate with the other logical functions. A logical function can be sub-divided into logical sub-functions.

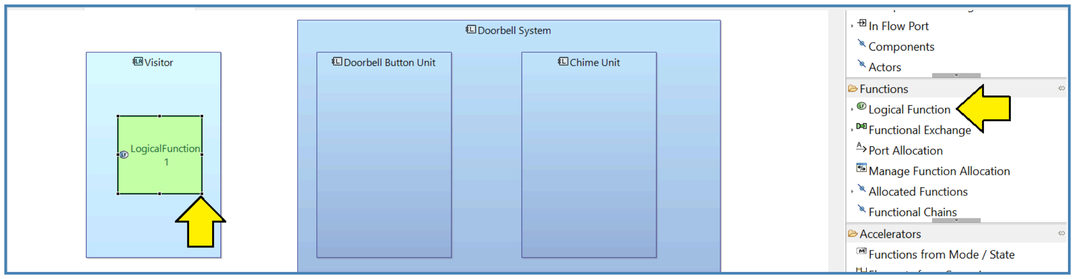


Figure 1-30 – Add Logical Function

Select the "Logical Function" tool in the "Functions" section of the palette. Click inside the visitor element and size the green box as shown.

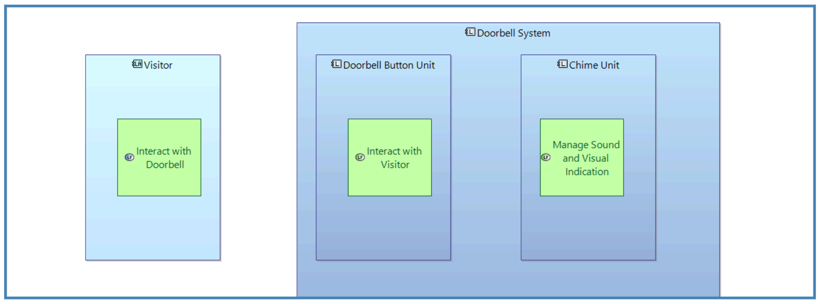


Figure 1-31 – Add two more logical functions

Rename the first logical function: "Interact with Doorbell". Add two more logical functions named: "Interact with Visitor" and "Manage Sound and Visual Indication" inside the doorbell button unit and the chime unit respectively. Make all three logical functions the same size.

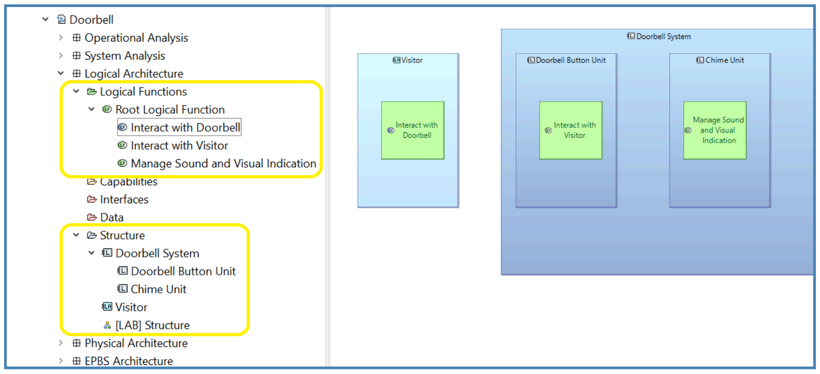


Figure 1-32 – Logical functions in the model

Notice that the logical functions are nested inside the logical components in the diagram, but not in the project explorer. The diagram represents **allocation** rather than containment. That is, the logical functions are allocated to their respective logical components, not contained within them.

If you look very carefully at the logical functions, you will notice that each one contains a small oval icon which in turn contains the letters: "LF". If you look even more closely, you will notice that the LF icon for the actor function is blue, distinguishing it from the other LF function icons which are green.

Next, we will introduce the concept of a **Functional Exchange**. A functional exchange represents a unidirectional exchange of information or matter between two functions.

If we were really doing formal systems engineering on a large system, we would carefully derive the sequence of steps in a functional exchange from the system operational concept. In later chapters, we will introduce some more rigorous techniques for making this sort of air-tight derivation. Right now, however, we are just interested in getting familiar with the tool, so we will use the following simple outline of the visitor/doorbell interaction to model the functional exchange:

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.

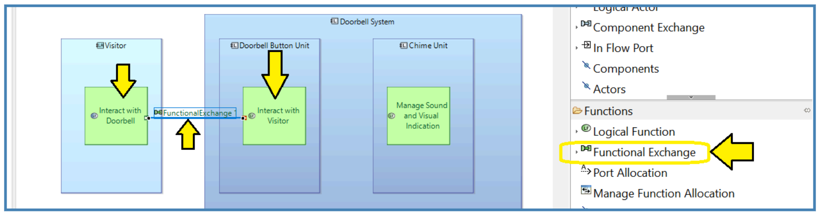


Figure 1-33 – Draw a functional exchange

Select “Functional Exchange” in the palette. Next, click once inside the logical function “Interact with Doorbell” and once inside the logical function “Interact with Visitor” to draw a functional exchange between the two.

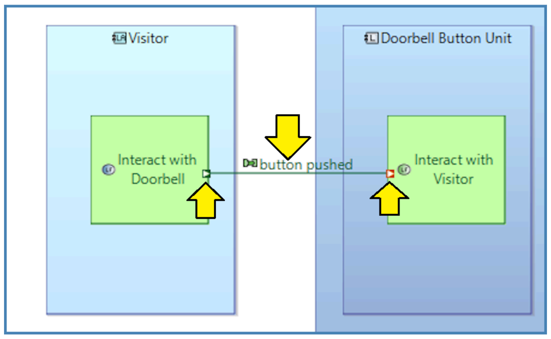


Figure 1-34 – Rename functional flow

Rename the functional flow: “button pushed”. Notice:

- *Capella* has created ports on each logical function for the functional flow.
- Each port contains a small arrow to indicate the direction of the flow.

When creating functional flows, always click the “from” logical function first and the “to” logical function second.

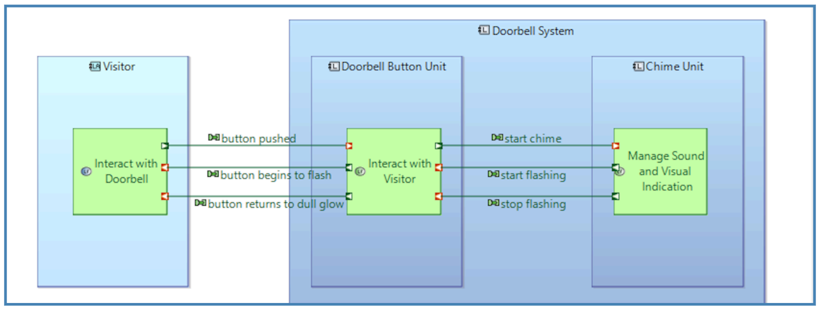
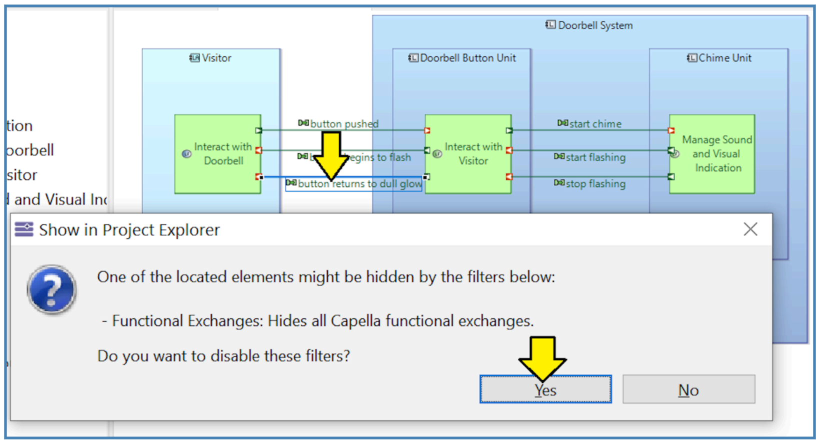


Figure 1-35 – Completed functional exchanges

Using the simple outline, complete a set of functional exchanges to support the desired behavior of the doorbell system.

If we now look at the project explorer again, we will not (initially) find the functional exchanges or the ports. In order to prevent the project explorer view from becoming overwhelmingly complicated, *Capella* has filter functions with default settings that suppress some detail – such as ports and functional exchanges. These default filter settings can be modified to show or hide these details in the project explorer.



One of the located elements might be hidden by the filters below:

- Functional Exchanges: Hides all Capella functional exchanges.

Do you want to disable these filters?

Figure 1-36 – Display functional exchanges

Select one of the functional exchanges in the diagram. Press F8. A panel will appear that allows you to disable the filter that suppresses the display of functional exchanges in the project explorer. Click: "Yes".

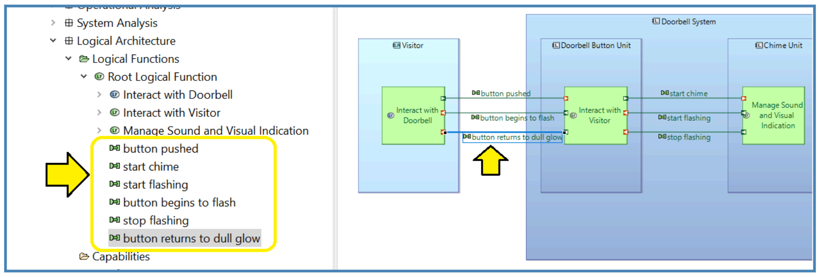


Figure 1-37 – Functional exchanges are now visible

The functional exchanges are now visible in the project explorer.

Now, let's take a closer look at the semantic browser. The semantic browser becomes more and more useful as the model gets larger. However, we can already see that it provides much more than the project explorer. For example, the allocation relationship is not visible in the project explorer because the project explorer is designed to only display containment and the allocation relationship is not a containment relationship. Likewise, inputs and outputs of functions are not shown in the project explorer either. The semantic browser is useful for exploring this sort of relationship detail. Selecting any model element – either in a diagram or in the project explorer – will cause the semantic browser to present a visualization of all the relevant model links.

Let's use the semantic browser to explore the details of a functional exchange. If the semantic browser is not open, use the "Show View" menu to display it as shown in *What if the Project Explorer isn't there?* on page 6.

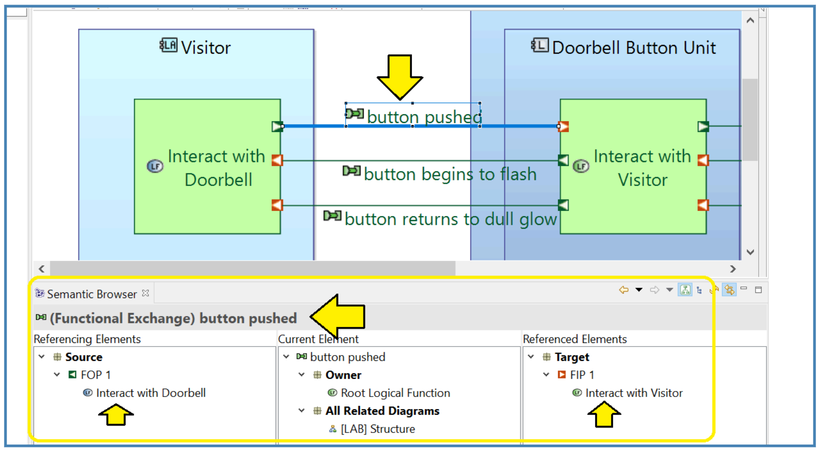


Figure 1-38 – Semantic browser for a functional exchange

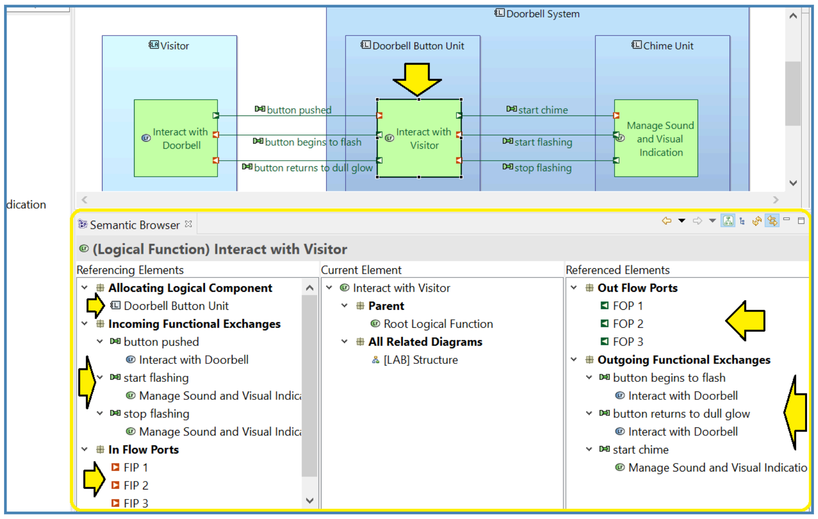
Select the "button pushed" functional exchange. We can see that the name of the selected model element is displayed at the top of the semantic browser. Underneath the selected element, the semantic browser shows three columns:

1) **Referencing Elements –** In the first column, we have a list of model elements that reference the current model element. In this case the "Interact with Doorbell" logical function has an outgoing port that connects to the current element.

2) **Current Element –** The second column shows information about the current element.

3) **Referenced Elements –** In the third column, we have a list of model elements that are referenced by the current model element. In this case the "Interact with Visitor" logical function has an incoming port that the current element connects to.

Next, let's take a look at the semantic browser display for a logical function.



(Logical Function) Interact with Visitor

```
Referencing Elements                      Current Element             Referenced Elements
v Allocating Logical Component            v Interact with Visitor     v Out Flow Ports
    Doorbell Button Unit                    v Parent                      FOP 1
v Incoming Functional Exchanges                 Root Logical Function     FOP 2
  v button pushed                           v All Related Diagrams        FOP 3
      Interact with Doorbell                    [LAB] Structure       v Outgoing Functional Exchanges
  v start flashing                                                      v button begins to flash
      Manage Sound and Visual Indic                                         Interact with Doorbell
  v stop flashing                                                       v button returns to dull glow
      Manage Sound and Visual Indic                                         Interact with Doorbell
v In Flow Ports                                                         v start chime
    FIP 1                                                                   Manage Sound and Visual Indicatio
    FIP 2
    FIP 3
```

Figure 1-39 – Semantic browser for a logical function

Select the "Interact with Visitor" logical function. Again, we see that the name of the selected model element is displayed at the top of the semantic browser with three columns underneath:

1) **Referencing Elements –**
   - The allocation relationship for the logical function is shown here.
   - Incoming functional exchanges are shown here.
   - Input ports that belong to the logical function are shown here.

2) **Current Element –** The second column shows information about the current element.

3) **Referenced Elements –**
   - Outgoing functional exchanges are shown here.
   - Output ports that belong to the logical function are shown here.

Notice that the left/center/right columns of the semantic browser are in *semantic* order. Just because the outgoing ports are in the rightmost column of the semantic browser does *not* mean that those ports are on the right side of the model element in the diagram!

Adding a Scenario Diagram

Now we are ready to add a second diagram in our model: a **scenario diagram**. Once again, *Arcadia* defines many types of scenario diagrams. We will be creating an **Exchange Scenario [ES]** diagram. This diagram will represent a simple sequence of messages between the visitor and the two logical components.

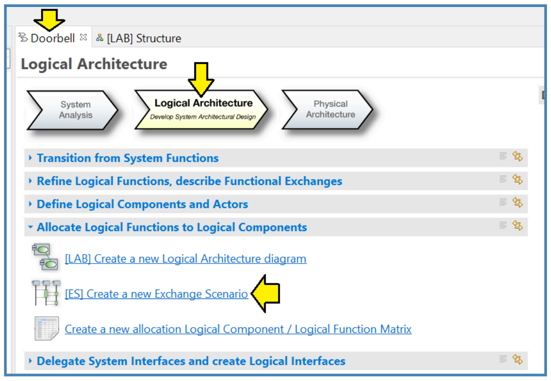

Doorbell
[LAB] Structure
Logical Architecture
System Analysis
Logical Architecture
Develop System Architectural Design
Physical Architecture
Transition from System Functions
Refine Logical Functions, describe Functional Exchanges
Define Logical Components and Actors
Allocate Logical Functions to Logical Components
[LAB] Create a new Logical Architecture diagram
[ES] Create a new Exchange Scenario
Create a new allocation Logical Component / Logical Function Matrix
Delegate System Interfaces and create Logical Interfaces

Figure 1-40 – Create exchange scenario

1) Click the "Doorbell" tab to return to the activity explorer. (1)
2) Make sure that you are still in the "Logical Architecture" section.
3) Expand the "Allocate Logical Functions to Logical Components" activity list.
4) Click on: "[ES] Create a new Exchange Scenario".

A window will appear to allow you to name the diagram. What should we name it? There are several factors we should consider in choosing a name for an exchange scenario:

1) *Capella* scenarios are tightly tied to the **capability** concept. Scenarios must belong to capabilities and will be stored in the related capability section of the project explorer.
2) In most system designs, we will create several different scenarios for each system capability, such as "Main Success Scenario", "Alternate Scenario", and "Error Scenario".
3) When you create a scenario, if you are not adding the scenario to an existing capability, *Capella* automatically creates a capability to own the scenario and gives the capability and its exchange scenario a default name of: "CapabilityRealization 1" or just "Capability 1" depending on which level of analysis you are working at.

(1)If you have accidentally closed the activity explorer, see *What if the Activity Explorer isn't there?* on page 9.

4) Having the capability and its scenarios all identified with the same name is probably not what we will want in our final model. We will be back to clean that up in a moment. For the moment, we will want to give the scenario a name that fits an activity, typically a verb phrase.

5) Experienced *Capella* users recognize the "[ES]" prefix. In fact, we should enhance the prefix by adding a "L" for "Logical" making it: "[LES]". (2)

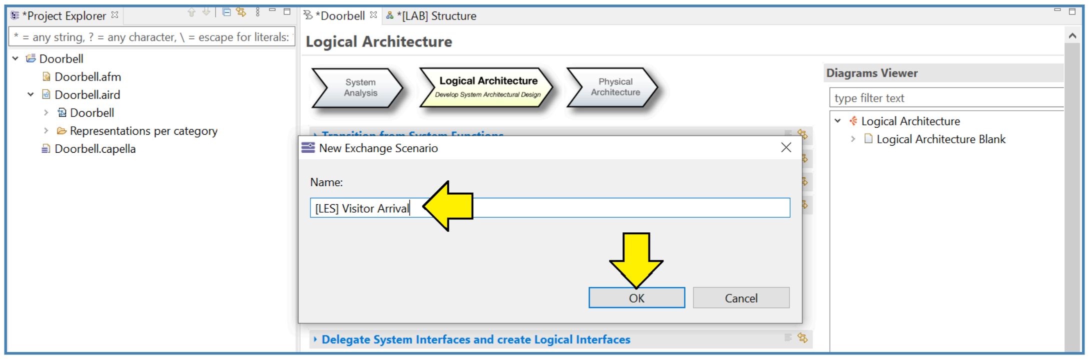


Figure 1-41 – Name the scenario

Name the scenario: "[LES] Visitor Arrival". Click: "OK".

We will discuss capabilities in more detail in later chapters of this book. For the moment, let's clean up the less-than-meaningful default name that was assigned by *Capella*.

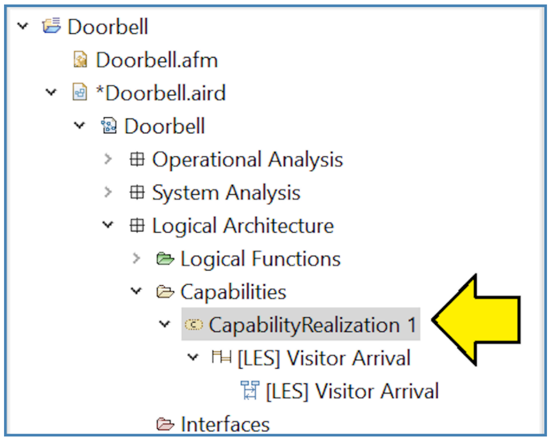


Figure 1-42 – Rename the capability

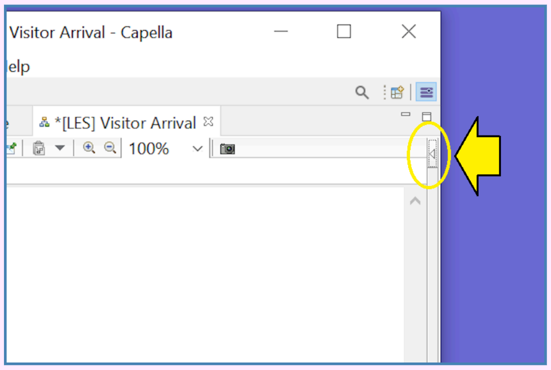
Find the capability in the project explorer and rename it: "Visitor Handling". The choice of capability name is a personal preference, but many architects use noun phrases for capabilities to distinguish them from the verb phrases used to name functions.

(2) Here we are actually working around a small inconsistency in *Capella*. In other areas of the tool, the letter for the architectural level would be added to the prefix automatically.

Note: We will cover capabilities in more detail in later chapters of the book.

What if the Palette isn't there?

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



If the palette is hidden, you will find a small arrow icon in the upper right-hand corner of the diagram pane. Clicking on this arrow icon will expand the palette again.

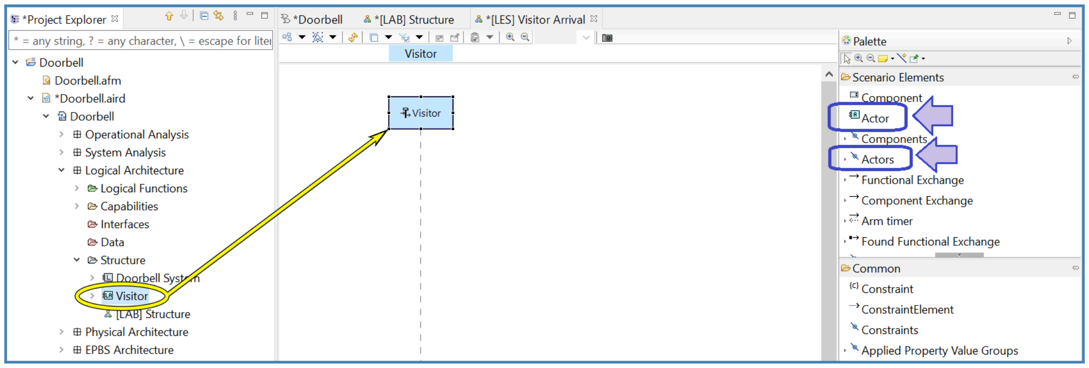


Figure 1-43 – Drag visitor to diagram

As before, icons are available in the palette to either create new model elements or add references to existing model elements in the diagram. In this case, however, we are going to use another diagramming technique: simply click on the visitor actor in the palette and drag the element into the diagram.

Capella has created something called an **instance role** for the visitor element. The *Arcadia* "instance role" concept is similar to the "lifeline" concept of UML and SysML.

- At the top is the element itself.
- Below that is a dashed line. The dashed line indicates the passage of time in the "life" of the element, starting at the top and progressing downward in time.

Note: in this book we will be mostly referring to the dashed line portion of this diagramming element. We will be using the somewhat more intuitive UML/SysML term ***lifeline*** rather than the formally correct, but less understandable *Arcadia* term: "instance role".

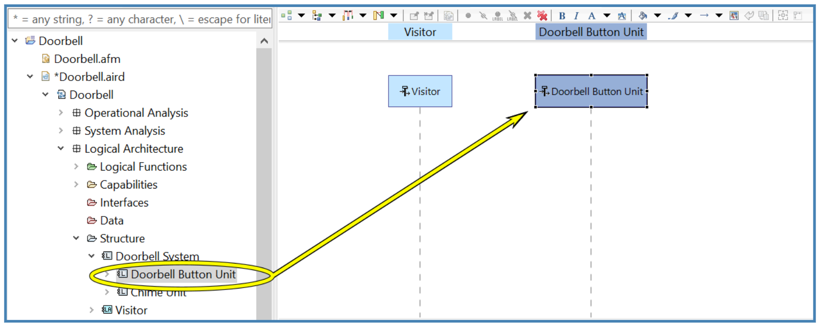


Figure 1-44 – Drag the doorbell button unit to the diagram

Drag the doorbell button unit to the diagram. The instance role (the top of the lifeline) for a component looks a little different than the instance for the actor: dark blue instead of light blue. The colors and their meanings match the colors shown in Figure 1-29 on page 19. You can resize the header box of the lifeline by selecting it and dragging the corners of the box around as needed. You can also change the horizontal position of the lifeline by dragging it back-and-forth as needed.

Drag the chime unit block to the diagram as well. We now have three instance roles (lifelines) for three elements.

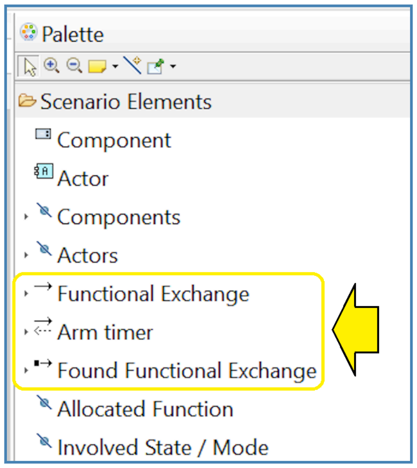


Figure 1-45 – Message types

Now we are ready to start creating the actual interaction between these three elements. In the palette you will find icons for several different types of messages. We will cover the differences between these types of messages in more detail in the chapter on scenario diagrams. For now, we will use the first type: "Functional Exchange". Each message in the scenario diagram will refer to a functional exchange between logical functions allocated to the relevant component or actor.

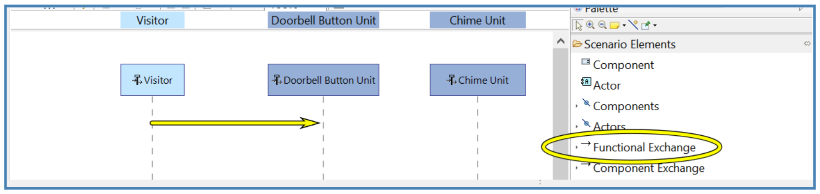
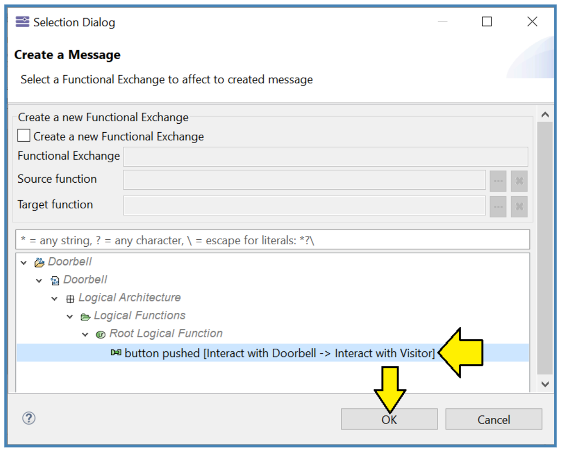


Figure 1-46 – Draw a functional exchange message

Select "Functional Exchange" in the palette, click on the lifeline for the visitor, and drag rightward to draw a message arrow from the visitor to the doorbell button unit.

Capella automatically opens a window prompting you to select one of the existing functional exchanges. In our case, only one functional exchange is relevant: the one that goes from "Interact with DoorBell" (which is allocated to "Visitor") to "Interact with Visitor" (which is allocated to "DoorBell Button Unit").


Selection Dialog

Create a Message

Select a Functional Exchange to affect to created message

Create a new Functional Exchange
☐ Create a new Functional Exchange
Functional Exchange
Source function
Target function

* = any string, ? = any character, \ = escape for literals: *?\

- Doorbell
  - Doorbell
    - Logical Architecture
      - Logical Functions
        - Root Logical Function
          - button pushed [Interact with Doorbell -> Interact with Visitor]

Figure 1-47 – Select the relevant functional exchange

Select the relevant functional exchange. Click: "OK".

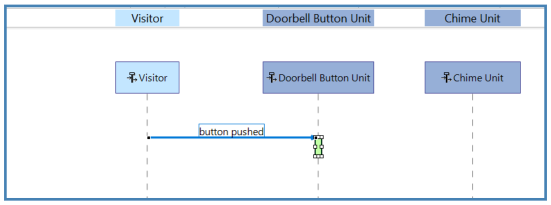


Figure 1-48 – Message created

Capella has created an arrow from the visitor to the doorbell button unit. The tool has also created a green rectangle on the lifeline of the doorbell button unit, called "Execution". We will look at this concept more carefully in the chapter on scenario diagrams.

This green box is similar to the "execution specification" concept in SysML and UML. However, *Capella* has a nice feature that is not supported in most SysML tools: we can link the green box to an actual function in the model.

Note: *Arcadia* does note really seem to have a formal name for this green box. The UML/SysML term "execution specification" is a bit arcane. We will refer to this green box as the ***execution bar.***

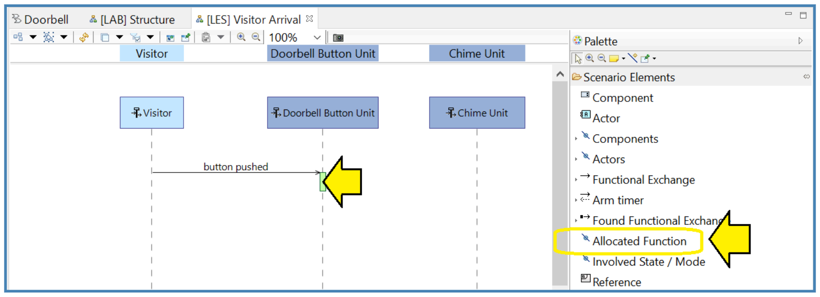
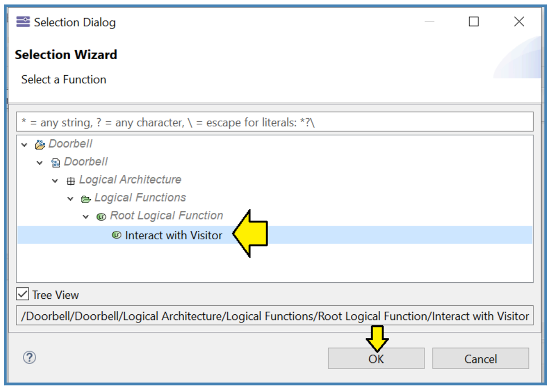


Figure 1-49 – Add allocated function

In the palette, select the "Allocated Function" tool. Click in the execution bar (green box) to add the allocated function to the lifeline.



/Doorbell/Doorbell/Logical Architecture/Logical Functions/Root Logical Function/Interact with Visitor

Figure 1-50 – Select a relevant allocated function

Capella will only allow functions that are allocated to the lifeline you have selected to be inserted. In this case, only the logical function: "Interact with Visitor" is available. Select that function and click: "OK".

Note that the text of the allocated function may initially appear to be truncated. In order to resize the allocated function box in the diagram, you may need to first increase the length of the (green) execution bar underneath the green allocated function box.

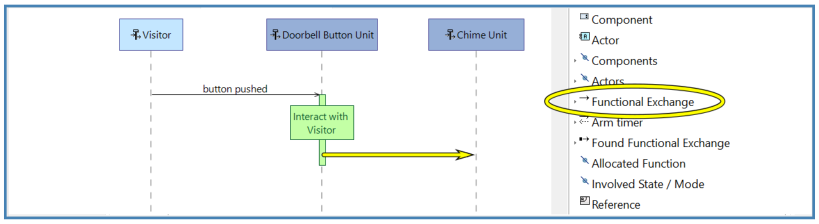


Figure 1-51 – Draw another functional exchange message

Draw a second functional exchange message from the doorbell button unit to the chime unit. Be sure to click inside the execution bar on the doorbell button unit lifeline to indicate that this new message is an output of the allocated function. Again, *Capella* will only allow you to select the existing "start chime" functional exchange.

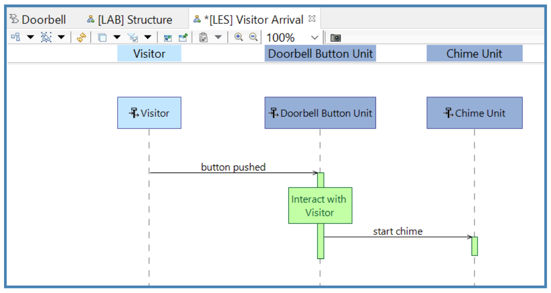


Figure 1-52 – Second message is now visible

The Second message is now visible.

Add some additional messages and functions to complete the modeling of the user story:

- Insert the "Manage Sound and Visual Indication" function on the lifeline for the chime unit.
- Add a message that references the functional exchange "start flashing" from the chime unit to the doorbell button unit.
- Add a message that references the functional exchange "button begins to flash" from the doorbell button unit to the visitor. The flashing is just a visual indication but it is a message, nonetheless.
- Add a message that references the functional exchange "stop flashing" from the chime unit to the doorbell button unit.
- Add a message that references the functional exchange "button returns to dull glow" from the doorbell button unit to the visitor. The sudden absence of flashing is itself a message and can be modeled as such.

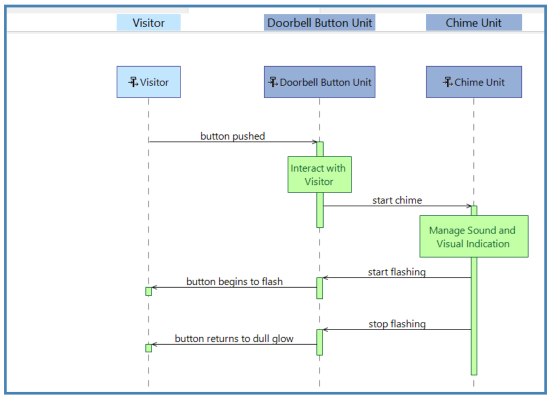


Figure 1-53 – Scenario is complete

The finished scenario should look like Figure 1-53.

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