Component Diagram

HOT Architecture Documentation

This document provides an overview of a component diagram, then digs into the mechanics of creating one in LibreOffice. For a general overview of the technical documentation approach for HOT, check out the technical documentation Wiki at https://github.com/hotosm/techdoc/wiki.

When it comes to solution architecture, a component model is a more technical depiction of a logical solution design. The goal of this component diagram is to capture architecturally significant components and their connections. Component diagrams are made up of boxes, lollipops, and lines.

Table of Contents

What is a component diagram?

From our friends at Wikipedia:

A component diagram allows verification that a system's required functionality is acceptable. Programmers and developers use the diagrams to formalize a roadmap for the implementation, allowing for better decision-making about task assignment or needed skill improvements.

Elements of the diagram!

Here is a quick review of how to think about each element on the diagram:

- Each **box** represents an architecturally significant component of the solution architecture. Components can be nested to show sub-components. Each component includes the name of the component and the “type”, or stereotype. More on this later.
- A **lollipop** sticks out the side of a component when a component is providing an interface for other components.
- A **line**, or **Required Interface**, connects those other components to the lollipops to show that the components are using the interfaces.

Some Tips

Before we get started on the mechanics, here are a few tips[1]:

- Keep all the components the same size. It makes your diagram look cleaner and more professional. If you show nested components, the internal components should be the same size as other components on the diagram.
- Use yellow notes during drafting to capture open questions and, minimally, when publishing to provide clarifications.
- Use dependencies, dotted lines with arrows, to indicate a component that is dependent on another component. Try to use these sparingly and, instead, make any necessary dependencies intuitive based on placement.
- Align boxes vertically and horizontally as much as feasible. It makes your diagram look cleaner and more professional.

Drawing with LibreOffice

Getting Started

This assumes you know how to use LibreOffice Draw and provides additional guidance to help you create this specific diagram in LibreOffice. Please checkout these resources to learn more about using LibreOffice:

- https://www.libreoffice.org/get-help/install-howto/
- https://documentation.libreoffice.org/
- https://documentation.libreoffice.org/assets/Uploads/Documentation/en/DG7.5/DG75-DrawGuide.pdf
- https://help.libreoffice.org/latest

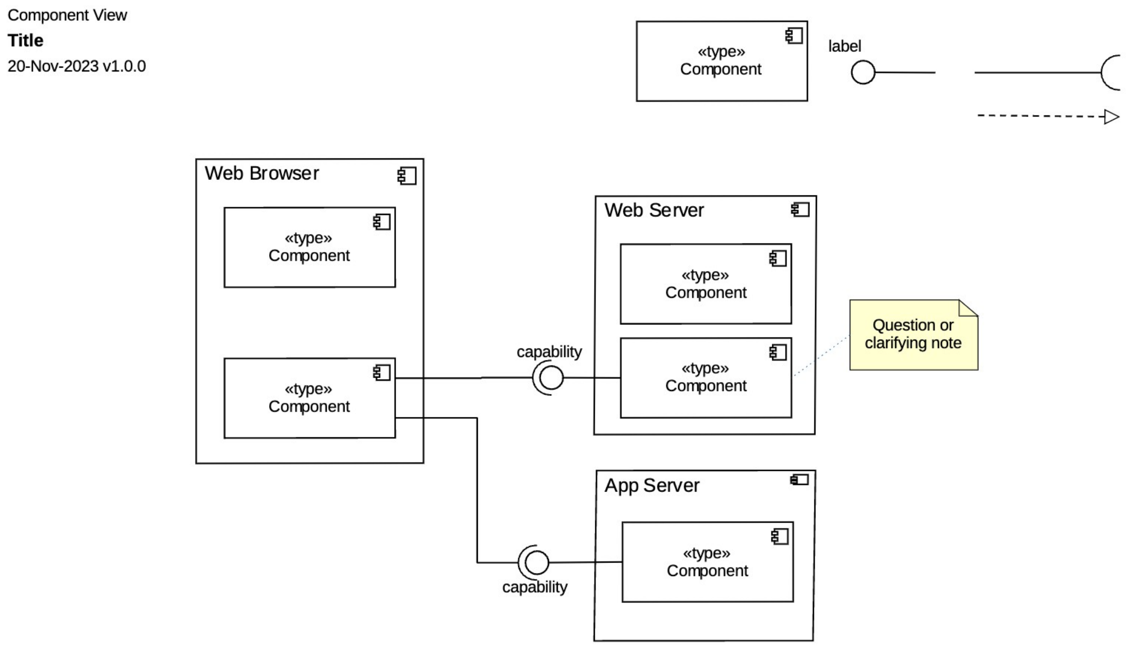
It is also often quickest to grab an existing diagram and edit instead if starting from scratch. You can find an existing component diagram here: https://github.com/hotosm/techdoc/blob/main/overarching-architecture/tasking-manager/Tasking%20Manager%20Component.odg

Otherwise, start by using the file menu to Create a New Drawing.

Page Setup

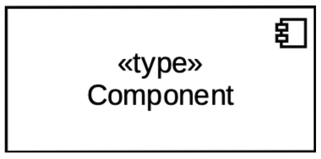
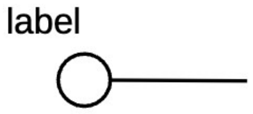
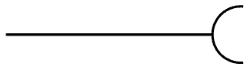
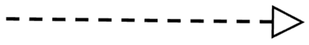
If you are not using an existing diagram to start, make a copy of the Component Diagram template and rename it to the name of your solution, e.g. “Tasking Manager Component Diagram.odg.”

1 Scott Ambler’s book “Elements of UML Style 2.0” has some great general diagramming tips.



In the upper left, is our standard **title block**. Change the word "title" to the title of your diagram. It should describe the scope of your diagram. For example, "Tasking Manager."

In the upper right are 4 shapes you may use to create the diagram:

	The **component** is the most common box on the diagram and is used to identify architecturally significant components. The icon in the upper right identifies the box as a component. It is labeled with a logical name and classified using a type (aka. "stereotype). You can nest components as required to depict parent-child relationships along with a lower level of detail.
	The **Provided Interface**, or "lollipop," is attached to a component to signify an interface provided by that component for other components to consume. The label indicates the capability provided by that interface. On a logical level diagram where we show services as components, the capability provided is the protocol used to interact with that component. It is only useful to identify the protocol for industry standard protocols, so for any custom protocol, label the lollipop "native".
	The **Required Interface** is added to a component to show that the component requires another component to provide a specific interface to operate. We use this to connect a component to an interface that is provided by another component.
	A **Dependency**, a dotted arrow, indicates that the component where the arrow originates is dependent on the component at which the arrow terminates.

To use these, simply make a copy, rename and position appropriately. When you are done with the diagram you can delete the shapes off of the top.

Stereotyping Components

It’s helpful to have a common language for stereotypes to keep component diagrams simple and widely understandable by the entire team. These stereotypes are able to handle most situations, however, if absolutely necessary, there can be exception cases.

Application	A software program or executable component; can be something that runs on another runtime (like a configuration that runs in an application environment or virtual machine).
Datastore	A persistent storage location for data.
File	Self explanatory.
Node	A “logical” environment within which other components can run. They are the outer components in a nested component stack.
Service	A discrete unit of functionality that can be accessed remotely by other components. You can think of a service as anything offering an API for other components.
Job	Any batch run process, usually scheduled but not always. Also a “discrete unit of functionality”. Usually, if it’s invoked with an API, it’s a service; if it isn’t, it’s a job.
Library	An object or function library used by other components on this list.
Package	A component representing multiples of some type (any of the above) where the details are not necessary, e.g. “Scanned Documents.”

Connecting Components

Once you have some components added to the diagram, copy and paste the lollipops and connect the straight end to the component providing the interface. Label the lollipop end with the industry-standard capability provided, such as “rest” or “http”.

Once you have all your interfacing lollipops in place, begin to connect those with components using the Required Interface line. The straight end of the line should connect to the component using the interface and the U-shaped end of the line should rest against the lollipop end of the component providing the interface.

Clean Up

Once you have your whole diagram laid out, select components and move connectors to make the diagram as simple to read as possible. Connectors should never cross each other and components should be aligned vertically and horizontally, where possible.