Ontology Editor Documentation

A component of the NIST, SJU, NMSU CPS System

Written By Matt Bundas (bundasma@nmsu.edu)

Table of Contents

Summary

The ontology editor is a graphical program written in python that allows a user to edit an ontology without having to manually edit an .owl file. After loading from a .owl file, it displays the ontology tree, allows the user to navigate the tree, and allows the user to modify the ontology by interacting with graphical nodes, buttons, and entry windows. Users can edit just about anything contained in the ontology including adding/removing aspects, concerns, properties, components, dependencies, and relations between nodes such as subconcern, addresses concern, and other dependencies.

The ontology editor is created and managed by the OntologyGUI.py script, which uses classes stored in other python scripts to portion the workload of handling the editor's function.

The ontology editor’s input is an existing .owl file, which is loaded in by the editor. The editor can output an .owl file with specified name, which can be used by the ASP solver as part of our overall system, or be used in any other manner you would use an .owl file.

Requirements

The ontology editor is written in python, and makes use of several python packages. To run the editor, you need a working python version and installations of the following packages:

Tkinter - https://docs.python.org/3/library/tk.html
Owlready2 - https://pythonhosted.org/Owlready2/
Networkx - https://pypi.org/project/networkx/
Numpy(comes with python installations) - https://numpy.org/
Matplotlib(comes with python installations) - https://matplotlib.org/
Graphviz - http://www.graphviz.org/
Pygraphviz - http://pygraphviz.github.io/documentation/pygraphviz-1.5/index.html
Pydotplus - https://pypi.org/project/pydotplus/

The dependencies between exact conversions can be incredibly wonky, so rather than trying to massage all of the packages to agree with each other, you can create an anaconda virtual environment from a .yml file. This will help ensure a working combination of packages gets installed. This file is in the GitHub repo, in the interface/testing folder named ontology_GUI_env.yml. You can create a virtual environment, using the command "conda env create -f ontology_GUI_env_windows.yml" or "conda env create -f ontology_GUI_env_mac.yml". You will then need to use the command "dot -c". Then, activate the environment using "conda activate ontology_GUI_windows" or "conda activate ontology_GUI_windows". You should be good to go from there.

Currently all files related to the system can be found here, with the updated branch being "matthew": https://github.com/thanhnh-infinity/Research_CPS

Files to run the editor are in interface/testing/. All files in this directory are needed to run the editor, but none outside of that. Keep all of these files in the same directory.

At present, the ontology editor is functional on all Mac/Linux/Windows, but it is being developed on Windows and may have bugs on other platforms, or not be visually-ideal.

Usage

To run the ontology editor, navigate to the directory containing OntologyGUI.py and its accompanying python scripts. In terminal/command prompt do ***python OntologyGUI.py***, this will run the script and launch the editor.

Interface Layout

The interface is broken up into three main sections, the I/O Button Section, the Information Section, and the Tree Section.

I/O Button Section

Used to load in base and application ontologies, as well as output them to a .owl file

Information Section

Broken up into three sections, Ontology Information, Hovered Information and Most Recent Operation.

Ontology Information

Gives information about the ontology as a whole, including the name of the ontologies used as well as numerical information about the quantity of individuals in the Ontology.

Hovered Information

Shows information about the node which your mouse is hovering over, including its name, type, parent(s), and children.

Most Recent Operation

Shows the most recent operation performed in the editor, where operations are things like loading an ontology, adding a concern, deleting a property etc.

Tree Section

Once an ontology is loaded, it shows the ontology as a tree, allowing the user to navigate the tree and edit the ontology by clicking buttons or directly on nodes. The tree is updated after each edit.

Loading an Ontology

Types of Ontologies

There are two types of ontologies which can be loaded into the editor, a base ontology and application ontology. The base ontology is the ontology which is general for CPS, and contains the aspects/concerns and their subconcern relations. An application ontology is specific to a CPS and contains the properties and components related to a specific application of a CPS. For an application ontology to be used, a compatible base ontology must be already loaded in.

Load Ontology
Ontologies are loaded using the entry windows and corresponding button located in the I/O Button Section. To load an ontology, type the exact ontology file name/path, including .owl at the end and click the corresponding button just below the entry. The entered path is relative to the location of OntologyGUI.py, so if just a filename is entered it will look for an ontology file in the same directory as OntologyGUI.py.

Outputting an Ontology
Ontologies are outputted using the entries and buttons below Output Name in the I/O Button section. To output an ontology, type the exact desired ontology file name/path, including .owl at the end, and click the Output Ontology button just below the entry. There are two separate sets of Entries/Buttons for outputting a base ontology and application ontologies, as they are stored in separate files. The entered path is relative to the location of OntologyGUI.py, so if just a filename is entered it will save in the same directory as OntologyGUI.py.

Tree Visualization

After an ontology is loaded, it will be visualized in tree format in the large area on the right side of the interface.

Nodes/Individuals
Individuals in the ontology are represented as nodes in the graph, which are color and shape coded depending on their type.

Aspects - Blue, traditional rectangle
Concerns - Red, traditional rectangle
Properties - Grey, overly rounded rectangle (if application is loaded)
Formulas - Purple, rounded rectangle (if application is loaded)
Components - Pink, rounded rectangle (if application is loaded)

The ontology is represented in hierarchical order, where uppermost nodes are of higher level than lowermost nodes.Uppermost nodes are dependent on all lowermost children nodes.

Edges/Relations
Relations in the ontology are represented as edges between nodes in the graph, which are coded by edge style and labeled along the edge.

Subconcern - solid line, encoded as includesConcern in .owl base file

addressesConcern - solid line, encoded as propertyAddConcern and formulaAddConcern in .owl application file
memberOf - solid if member of conjunction, dotted if member of disjunction, denoted in black
negMemberOf - solid if member of conjunction, dotted if member of disjunction, denoted in red
Related To - dotted line, encoded as relateToProperty in .owl application file

Navigating Tree

The tree can be navigated using the mousewheel to zoom in and out, and the sliding bars at the bottom.

Editing Ontology

The loaded ontology is edited through clicking nodes of the tree itself in the Tree Section of the interface, and by clicking buttons or the presented in the Tree Section or empty space in the Tree Section, which opens windows to guide the operation. Left clicking on a node allows for adding a new node connected to the clicked on node, editing the node's name, or deleting the node and optionally, its children. Left clicking on empty space opens a window allowing for the addition of new nodes without an initial relation. The relations button allows for the addition and deletion of relations between existing nodes. The dependencies button allows for the creation of dependencies between groups of properties and a concern. The Remove Floaters button deletes all nodes which don't have any relations to other nodes, usually as a result of other editing operations.

No two nodes can have the same name, if the user attempts to create a node with a name which already exists in the ontology, an error will be displayed and the operation will not proceed.

Add Node

Aspect - Left click on empty space in Tree Section, type in name of the new aspect, click Add Aspect button
Concern - To add a concern with no initial relation, left click on empty space, to add a concern with an initial relation, left click on an existing node you want the new concern to have a relation with. Type in name of new concern, click desired option to add new concern as. Options depend on the selected node.
Property - To add a property with no initial relation, left click on empty space, to add a property with an initial relation, left click on the node you want the new property to have a relation with. Type in name of new property, click desired option to add new property as. Options depend on the selected node.
Dependencies - Click on the Dependencies button, enter the desired rule in the LHS box, then the desired concern the rule should address, then the "Create Dependency" button. More details later in the documentation.

Component - To add a component with no relation, left click on the Tree Section, type in the name of the new component, click add component. To add a component related to a property, click on the desired property, type in the name of the new component, and click add child component.

Edit Node Name

To edit the name of any node, left click on an existing node, type in the desired name in the entry box, and then click the edit name button.

Node Deletions

Delete Single Leaf Node

To delete a leaf node (node with no children), left click on the node and click delete.

Delete Single Nonleaf Node

To delete a node which is not a leaf (node with children), left click on the node and click delete. This will open a window with different deletion options. Click the Delete Selected Only Button. This will delete the selected node, keeping its children and placing them in the top right of the Tree Section.

Delete Node and Descendants

To delete a node and all of its descendants, left click on the node and click delete. This will open a window with different deletion options. Since the selected node has children, deleting it may leave some nodes without a relation. If you want to delete the selected node, and the nodes which will become relentless as a result (but keep the ones which will still have a relation to another node), click Delete Selected + Resulting Relationless Children. To delete all descendants click the Delete Selected + All Children button. A window will appear, displaying all nodes to be deleted, and asking for confirmation of their deletion.

Left Click on Node (Add/Edit/Delete Node)

Left clicking on a node will select it and open up a window containing buttons which allow the user to perform operations on the selected node based on its type. The first few options allow the user to add a new node of a certain type with the name in the entry box. The Edit Name button will change the name of the selected node to be what it is in the entry box. The delete button will do one of two things. If the node is a leaf (at the bottom of the tree), it will be deleted upon pressing the delete button. If the node is not a leaf (has children), a window will be shown allowing for different deletion options

Add/Remove Relations - Relations Button

To perform relational operations, click the Relations button in the top left of the Tree Section. This window allows you to specify the parent and child's relation you wish to operate on. Left clicks on nodes in the Tree Section will select the parent and child, alternating which one it changes. Once a parent and child is selected, click add relation to add a relation between the two nodes. The relation is automatically detected depending on the rules of the ontology. To remove the existing relation between the two nodes, click remove relation. If a resulting node becomes relentless as a result of the removal of a relation, it will move to the top right of the Tree Section. The editor will not allow for relations between nodes which don't make sense, such as between a concern and component.

Remove All Relationless Nodes - Remove Floaters Button

To remove nodes from the ontology which no longer have a relation as a result of editing operations, and are now in the top right of the Tree Frame, click the Remove Floaters button in the top right of the Tree Section.

Dependencies (Formulas and DecompositionFunctions)

Dependencies are the rules which control the complex satisfaction requirements for a concern. For a concern or formula to be satisfied, the formula which addresses it must be satisfied. A formula can either be a conjunction (think "and"), where all members of the formula must be satisfied for formula satisfaction, or a disjunction (think "or"), where only one member of the formula must be satisfied for formula satisfaction. Formulas and properties can also be used in negated form, that is a formula might be satisfied if a property does **not** hold. Members of a formula are other formulas, where individual properties are considered formulas as well. It is worth noting here, that if you select a memberOf relation in the Relations window and try to add a dependency, it will switch it to negMemberOf. Similarly if you select a negMemberOf relation, it will switch it to memberOf.

Dependencies are added to the ontology with the editor using the Dependencies Button on the Tree Frame. This will open a window where a dependency can be entered using IF (property combinations) THEN (concern). The left hand side of the rule is entered using the keyboard, or by clicking on nodes, similarly for the right hand side. Which side the nodes are entered in is controlled by clicking the IF and THEN buttons in the window, where IF is selected by default. Dependencies can only be created using already existing Properties and Concerns.

Entering LHS

The LHS of a dependency is a combination of properties connected by if's and and's. The LHS must be entered in the following format (Property1 and (Property2 or Property3 or Property4) and (Property5 and Property6)). Where each property in a group of parentheses must be connected by the same operator (and/or), in the example Property2, 3 and 4, are connected by or. Each group of properties in parentheses will have a formula declared for it depending on whether the properties are connected by and or or, meaning the rule as a whole needs to have

parentheses wrapped around it, you can't just enter Property7 and Property8, it needs to be (Property1 and Property2). Every pair of properties needs an operator between them, for example you can't enter (Property1 (Property2 or Property3)), you need an operator between Property1 and Property2. To add include a property in a negated sense, type **not** before the property, for example Property1 and not Property2. When using **not**, you still need to include an and/or operator, which matches the operator used in the formula, for example Property1 and Property2 or not Property 3 is not valid, since you are mixing ands and ors in the same formula. A formula can only be a conjunction or disjunction, not both. To add properties to the LHS, you can manually type them in, or click on the nodes in the Tree Frame. Left clicking a node will try to add it as part of a conjunction (and), right clicking will try to add it as a disjunction (or). To enter operators and parentheses, you can make use of the buttons provided in the window, or manually type them in.

Entering RHS

The RHS of a dependency is just a single concern for the time being. This can be entered manually or by clicking on a concern in the tree. To tell the editor your click wants to end up in the RHS of the dependency, click the THEN button.

Create Dependency Button

After the dependency is entered according to the above documentation, click the Create Dependency Button. This will parse the LHS, and based on the parentheses create formulas (disjunctions or conjunctions) containing the properties in the parentheses. Deeper formulas are created first, for example in the above example, a conjunction will be made for (Property5 and Property6), make a name for it like Concern_Condition_1, then a disjunction for (Property2 or Property3 or Property4), call it Concern_Condition_2, then a conjunction for (Property 1 and Condition_1 and Condition_2). It will add these formulas to the ontology, setting up the correct relations between them like memberOf, includesMember, formulaAddConcern, and for the properties involved in the rule, the correct propertyAddConcern will be set automatically, although the Tree may not show it.

Design

The Ontology Editor is written in Python and makes use of several Python packages to make the coding and operations with the ontologies relatively simple. Several python scripts and classes make up the editor. OntologyGUI.py is the main script which handles the GUI and makes use of classes stored in other scripts to facilitate the ontology editing process. When an ontology is loaded or edited, these class individuals are built from scratch, storing information about the ontology, constructing the graph, and refreshing the graph in the GUI.

GUI

The GUI side of the Ontology Editor is handled by a Python package called Tkinter, and is contained in the OntologyGUI.py file. The GUI is class-oriented, so each instance of the Ontology Editor is an object of the OntologyGUI class. In this way, everything the GUI needs to know about such as the base ontology, operating system, zoom level, which windows are open etc. are all data members of the class.

Tkinter works by setting up frames and canvases, which are just portions of the GUI which you can attach other widgets to, like buttons and entry frames. Frames are set up to be proportional, so they take up the same relative space on every machine. Buttons are placed on frames, where when the button is clicked, it calls a function which can handle the operation the button is designed to enact.Buttons call functions which are always member functions of OntologyGUI, which handle the desired operation and in turn call functions in other classes, like the base ontology or graph.

Handling of Ontology

The loading, editing, and saving of the ontology itself is completely done by a package called owlready2. This package allows you to load an ontology from a specified .owl path, and after that treat the ontology as a python object. Owlready2 has all kinds of functions and features which make modifying the ontology as simple as one line. In this way, the editor never has to mess with an .owl file directly, it lets Owlready2 handle all of that. To help the editor function, there are also classes which help keep track of everything the Editor needs to know about the ontologies it is working with. These are stored in owlBase.py and owlApplicaton.py. These classes have a data member which is the Owlready2 ontology object, and several other data members to assist the editor. These classes make use of another class, owlNode, where each instance of this class is an individual in the ontology. This class stores a reference to the owlready2 object of the individual, and also information that the owlready2 object doesn't have like the individual's parents, children etc. The owlBase and owlApplication classes look at all individuals in the owlready ontology objects, and construct arrays of The Ontology Editor

primarily works with owlNodes instead of owlready2 objects because they have extra information.

Tree Graphing

The entire graph of the ontologies used in the editor is stored as part of a class in a separate file, owlGraph.py, and makes use of networkx to do the drawing of the graph itself. The owlGraph class has references to the owlBase and owlApplication class instances which represent the ontologies and store their owlNodes. The owlGraph class looks at the nodes in the base and application ontologies, and adds them to a networkx graph. Edges are added to the graph by looking at each node's children, and adding an edge between each one. Each node and edge are added to a list of common type, so groups of nodes can be graphed with different properties like color and shape. Positions of the nodes are automated at the moment, making use of pygraphviz. Pygraphviz is fed a networkx graph object with nodes and edges inside, and pygraphviz figures out the correct positions to draw the ontology as a tree. An instance of owlGraph is used by OntologyGUI to show the tree in the editor. It is reconstructed from scratch every time an operation is performed on the ontology such as adding or removing a node.

Files/Class Summary

OntologyGUI - Master class to handle GUI, facilitate ontology operations, makes use of all below classes with references to each, calls functions from other classes.

owlBase - Stores information about base ontology, and reference to owlready ontology object. Holds array(s) of all owlNode objects relevant to ontology.

owlApplication - Stores information about application ontology, and reference to owlready ontology object. Holds array(s) of all owlNode objects relevant to ontology.

owlGraph - Handles graph of ontology as a tree. Has references to ontology class objects. Looks at these and constructs a networkx graph.

dependencyCalculatorEntry - Handles what is seen in the Dependency window, including the buttons and functions called upon pressing.

parseDependency - Handles parsing of dependency text, detecting rules within dependencies and creating corresponding owlFormula objects.

owlFormula - Stores information about a single formula, including its members, its type, and operator.

owlNode - Stores information about a single ontology individual. Its name, type, parents, children, and reference to owlready object.

owlFunctions - Stores common functions used by multiple classes.

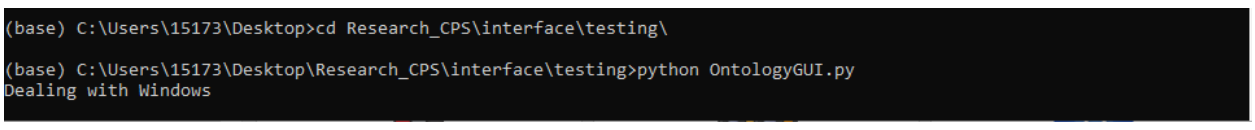
Connection to System

The purpose of the Ontology Editor is to make it easy for someone to edit an ontology without having to manually edit a .owl file, and without having to understand exactly how they work. It serves to give a basic visualization of the structure of the ontology, and provide an interactive and efficient way to edit an ontology. With regards to the system as a whole, it allows a user to edit an ontology which can then be used in whatever purpose they desire, whether that be connected to the system we have built or not. Specifically talking about our system, ontologies outputted from the Ontology Editor can be used by the ASP solver to be analyzed and eventually viewed in a much more meaningful capacity in Tableau.

Example

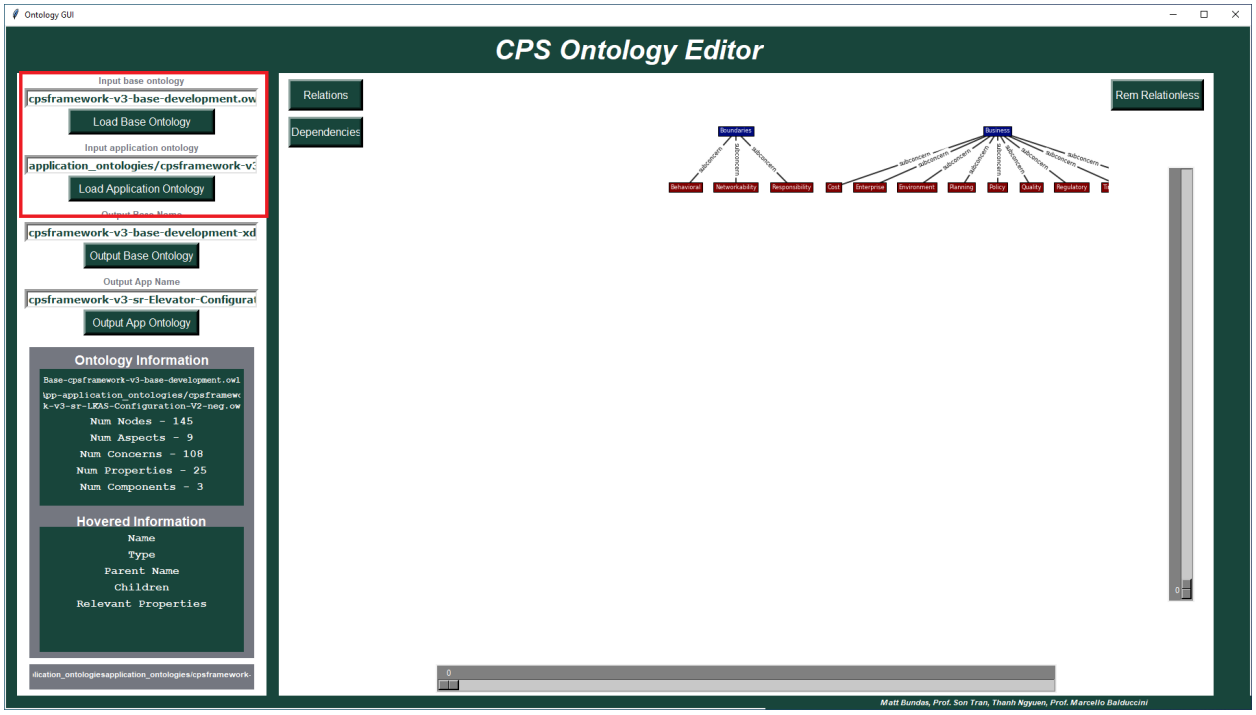
This section will be a walkthrough of using the ontology editor and serve as a bit of a tutorial. In the example, we will load in base and application ontologies, and edit them making use of all of the editor's main features, then finally output our newly edited ontologies. To make sure everything is covered, we will create an entire new Aspect tree, including the Aspect itself, its subconcerns, properties, components, and dependencies between them. Also, to keep things general and simple, I will be using general names for individuals, such as NewAspect, NewConcern1, NewConcern2 etc, but you can type in whichever names you'd like.

1. Launch the Editor
Be sure that you have fulfilled all of the requirements in the Requirements section, then go ahead and clone/download the Github repo https://github.com/thanhnh-infinity/Research_CPS which contains all of the files related to our project. Inside the repo, navigate to the directory containing OntologyGui.py, by default this will be /interface/testing, once there run OntologyGui.py, in terminal that looks like **python OntologyGUI.py**



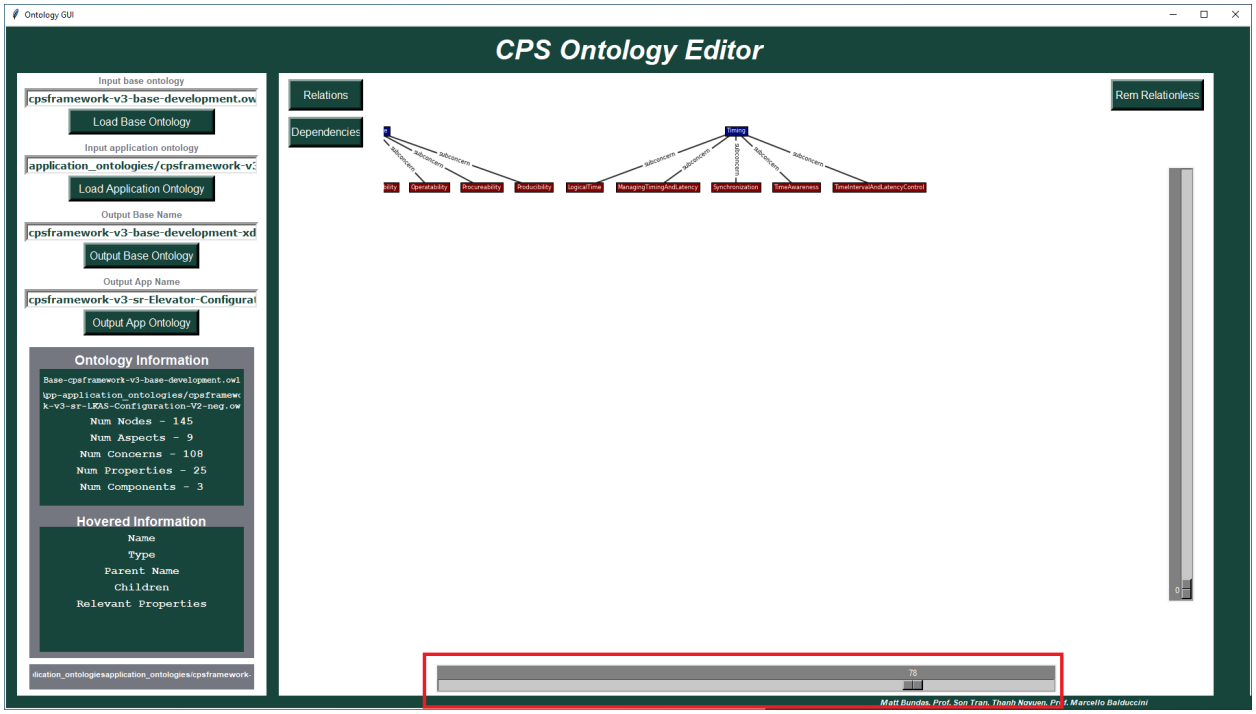
2. Load Ontologies

By default, the boxes to enter filepaths to the ontologies will be filled with operational ontologies, click **Load Base Ontology** then **Load Application Ontology**, this will load in the ontologies and display them in the editor.



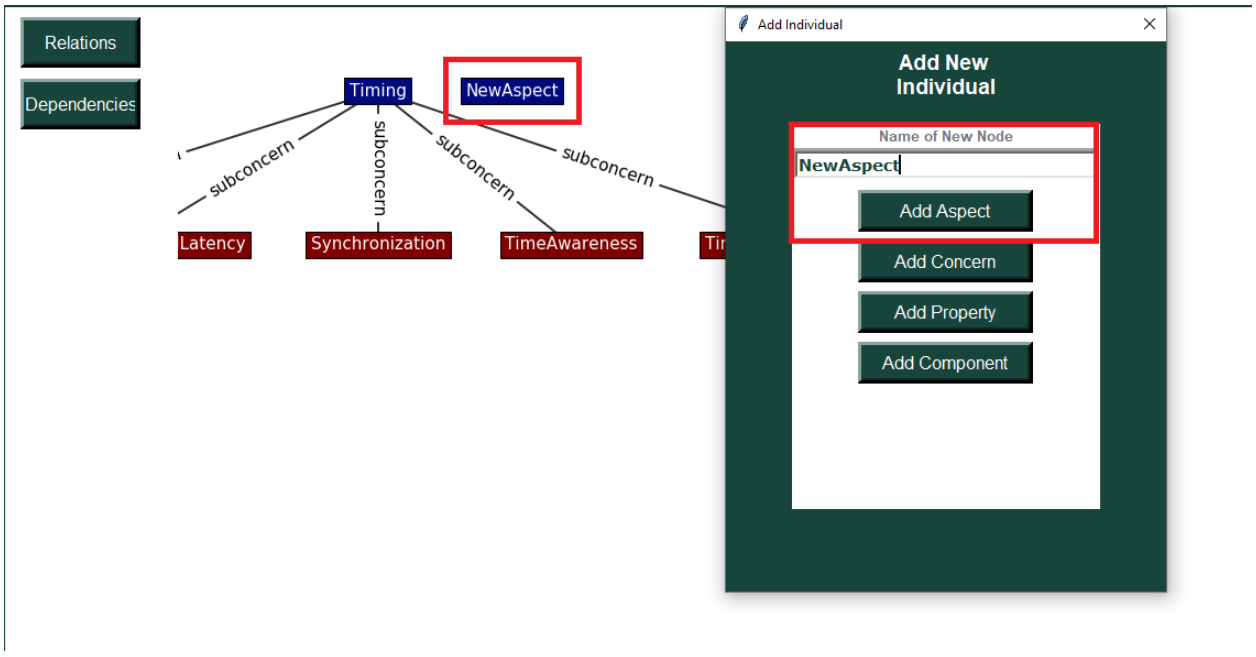
3. Scroll to Far Right

Newly created individuals/nodes who don't have any relations yet will show up on the far right of the tree, so lets scroll all the way to the right using the scroll bar at the bottom.



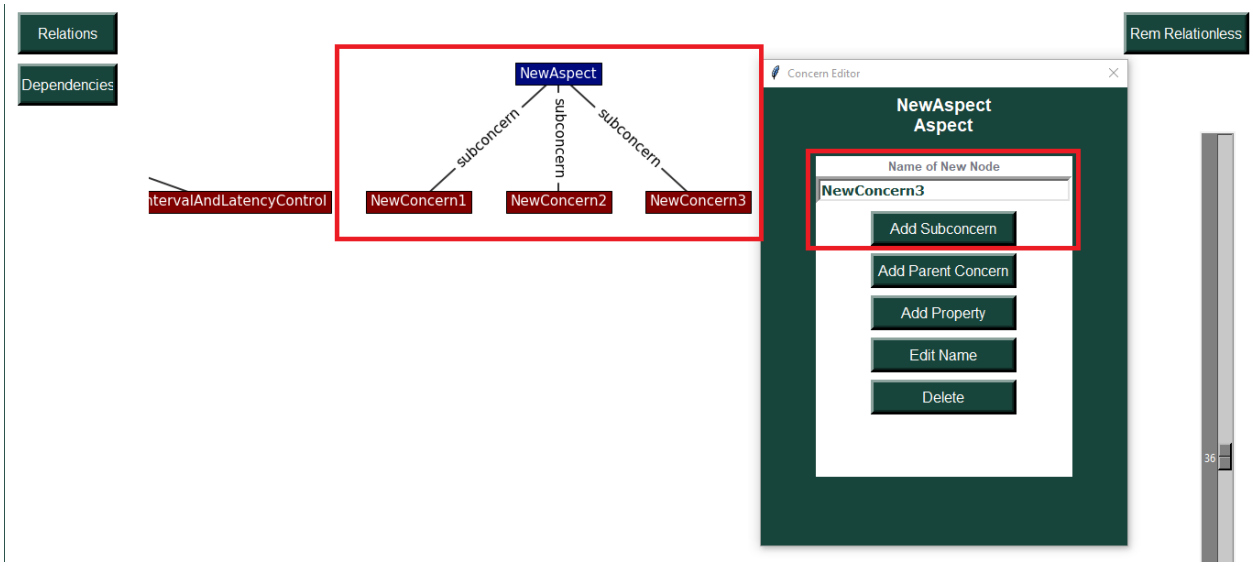
4. Add New Aspect

Left click on any empty space in the Tree Section. This will open up a window where you can add new individuals. We are going to type in the box NewAspect, and then click Add Aspect to add a new aspect. Zoom in using the scroll wheel if you'd like to make things bigger.



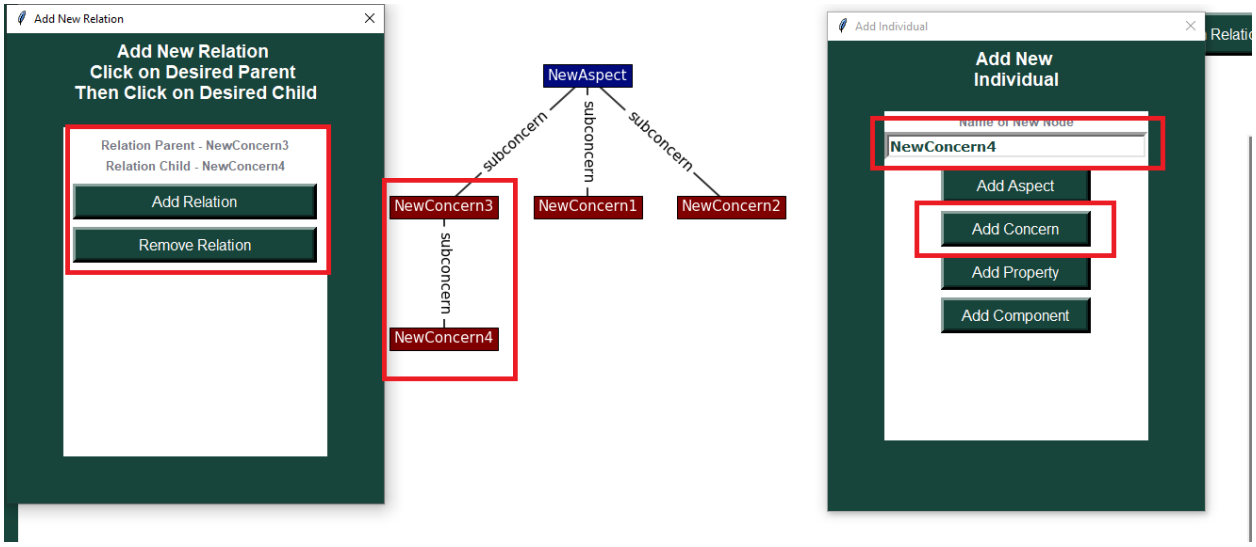
5. Add Subconcerns of New Aspect

Left click on the NewAspect node in the tree, now at the far right of the tree. This will bring up a window where you can do operations on NewAspect, like edit the name, delete it, or add related individuals to it. We are going to add subconcerns, so in the box type in NewConcern1, then click Add Subconcern, and to add more repeat the process, typing ing NewConcern2 then clicking Add Suboncern, finally NewConcern3 then clicking Add Subconcern.



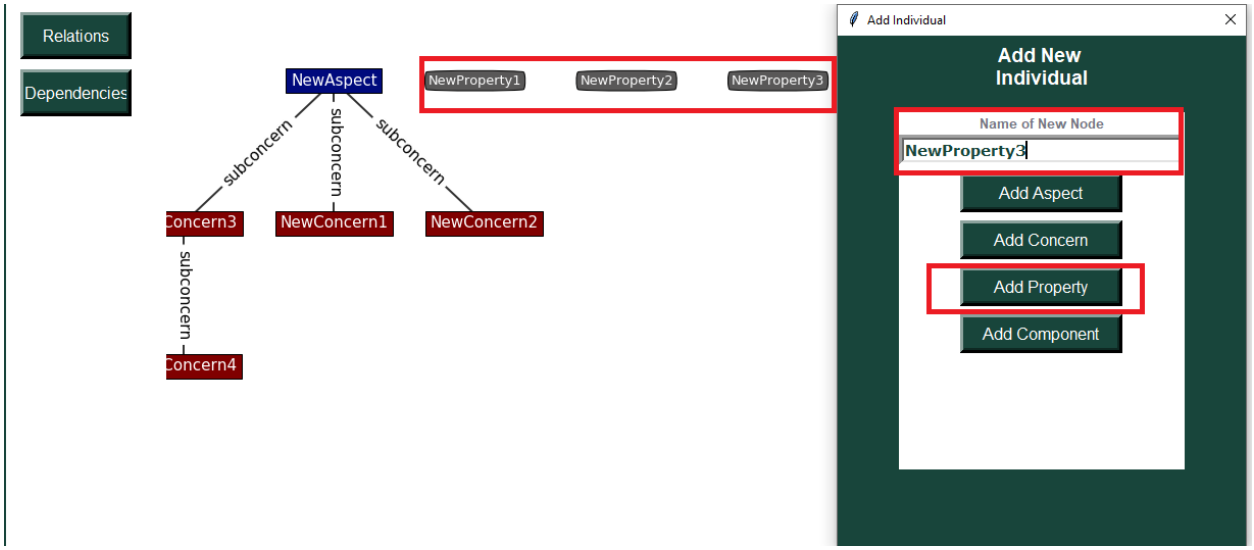
6. Add Subconcerns of New Concern

Now we are going to add subconcerns for NewConcern1. To do this, we could use the same method as step 5, but let's make use of the Relations button instead. Left click on some empty space in the tree, type in NewConcern4 and click the Add Subconcern button. To make the newconcern a suboncern of NewConcern3, click the relations button. Then click on NewConcern3 to select it as the parent, then NewConcern4 to select it as the child. Click the Add Relation button to add the subconcern relation.



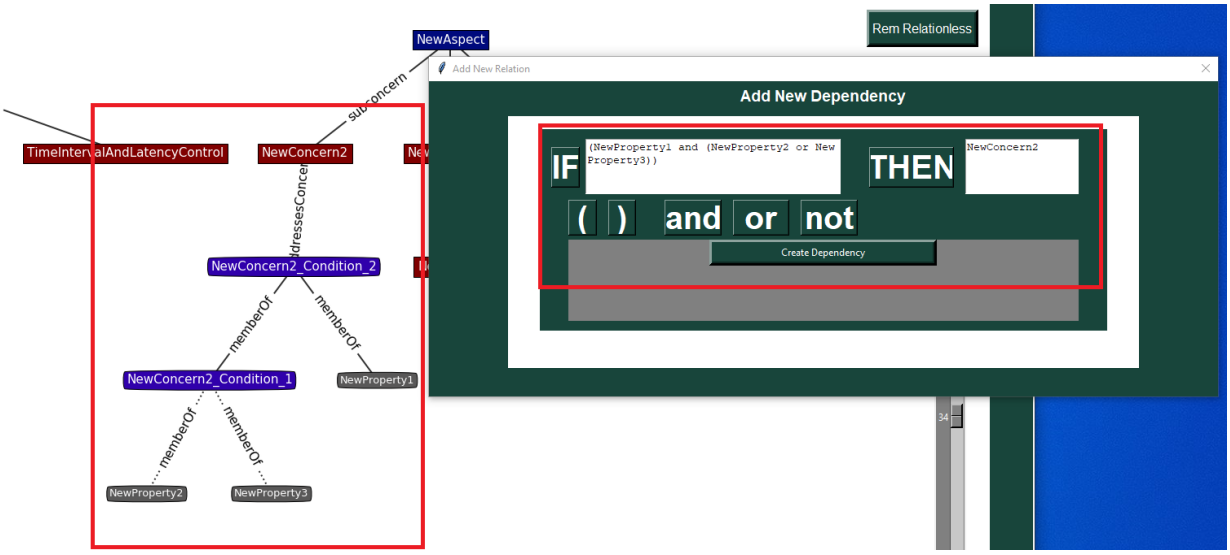
7. Add New Properties

Now lets add some new Properties to eventually address our new concerns. Left click on empty space, and add 3 new properties named NewProp1, NewProp2 and NewProp3 using the entry box and Add Property button.



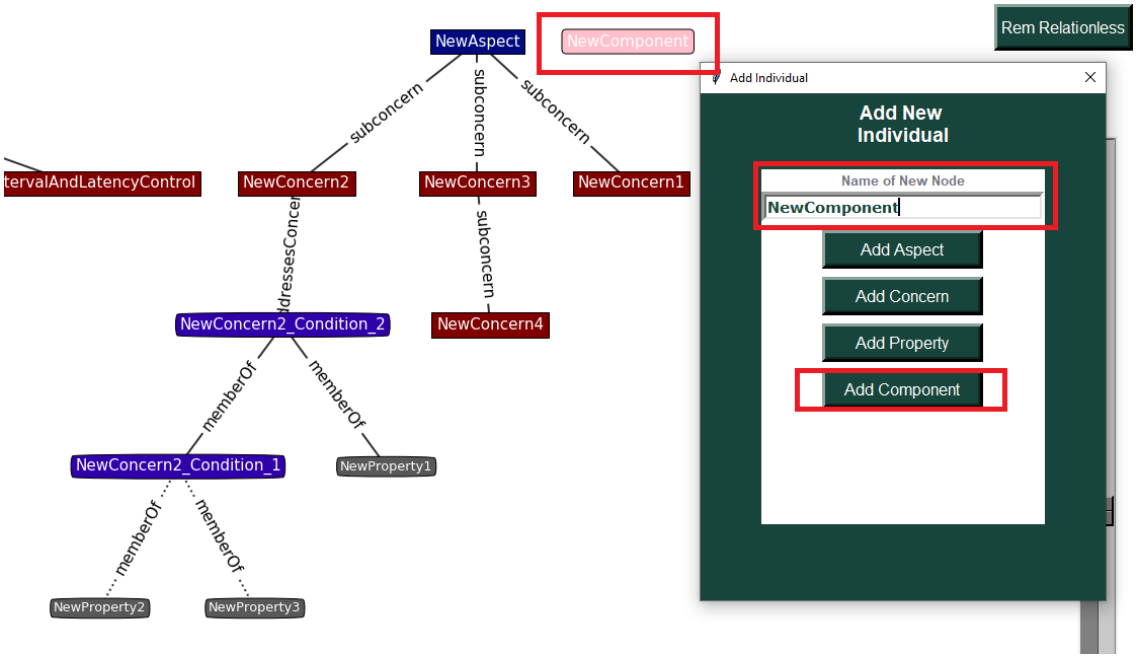
8 Add Dependency

At the moment, our properties aren't related to or addressing any concerns. For an example, let's create a dependency that if NewProp1 is satisfied and either NewProp2 or NewProp3 are satisfied, then NewConcern2 will be satisfied. In terms of a proper dependency, this will look like IF (NewProp1 and (NewProp2 or NewProp3)) THEN NewConcern2. To make this a reality, click the Dependencies button, which will open up a window to create dependencies. In the left entry window, we need (NewProp1 and (NewProp2 or NewProp3)) and in the right window we need NewConcern2. You can either type this in manually, or use the buttons in the window to add operators, and clicking on tree nodes to add properties to the entry.



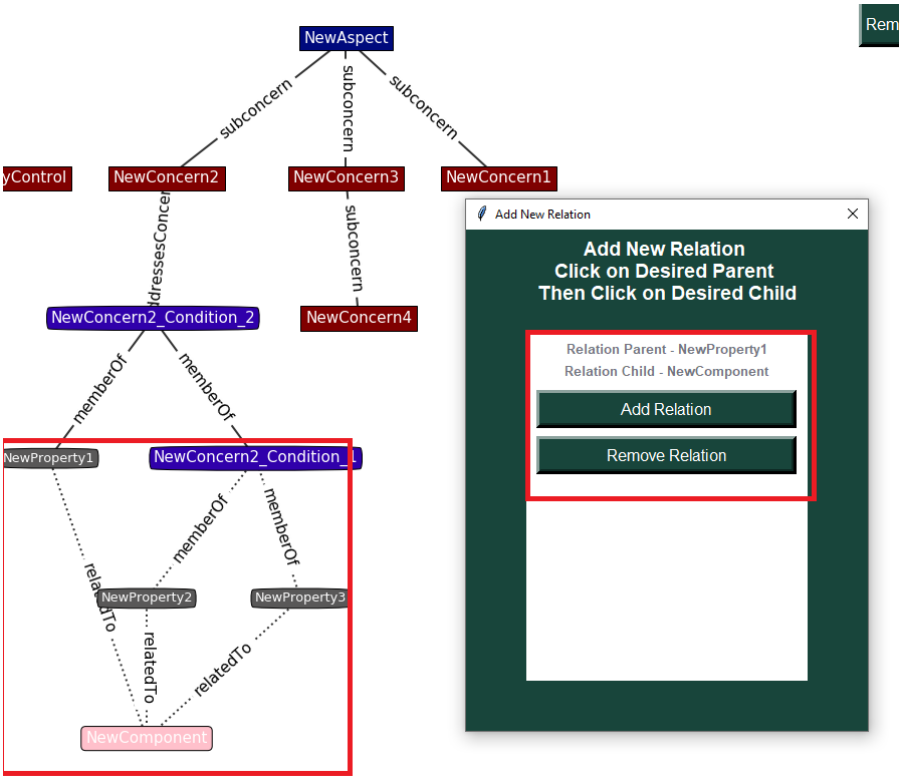
9. Add New Component

Since properties practically can't exist on their own, let's add a component to have the properties we just created. To create a new component, left click on empty space, type in NewComponent, and click Add Component.



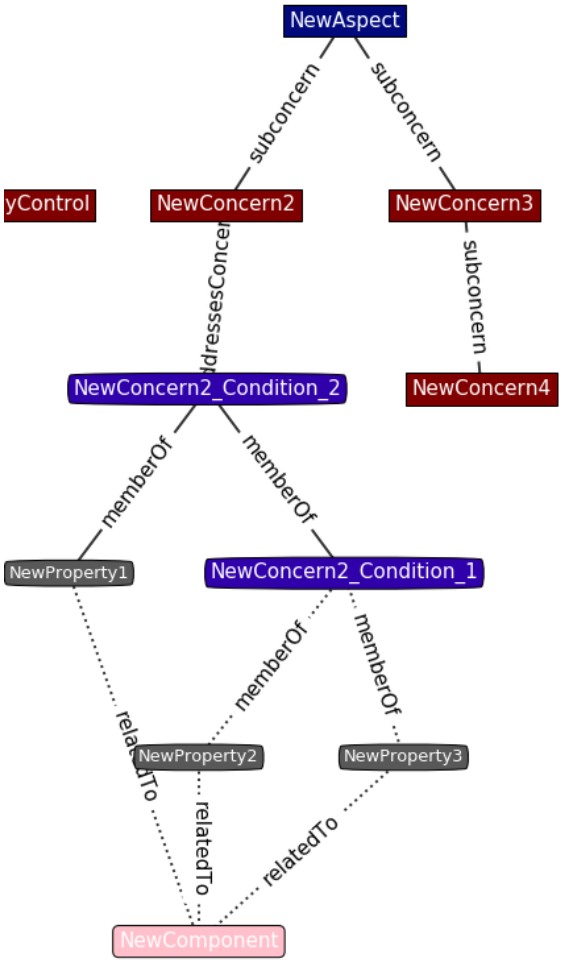
10. Relate Component to Properties

Now let's actually connect the properties and components, that is make the components related to the new properties. To do this, open up the Relations window, and left click on NewProp1 to make it the parent (in the tree), and then left click on NewComponent to make it the child, then the Add Relation button to make the connection. Similarly for NewProp2, click on the property then the component, then the Add Relation button to connect those as well.



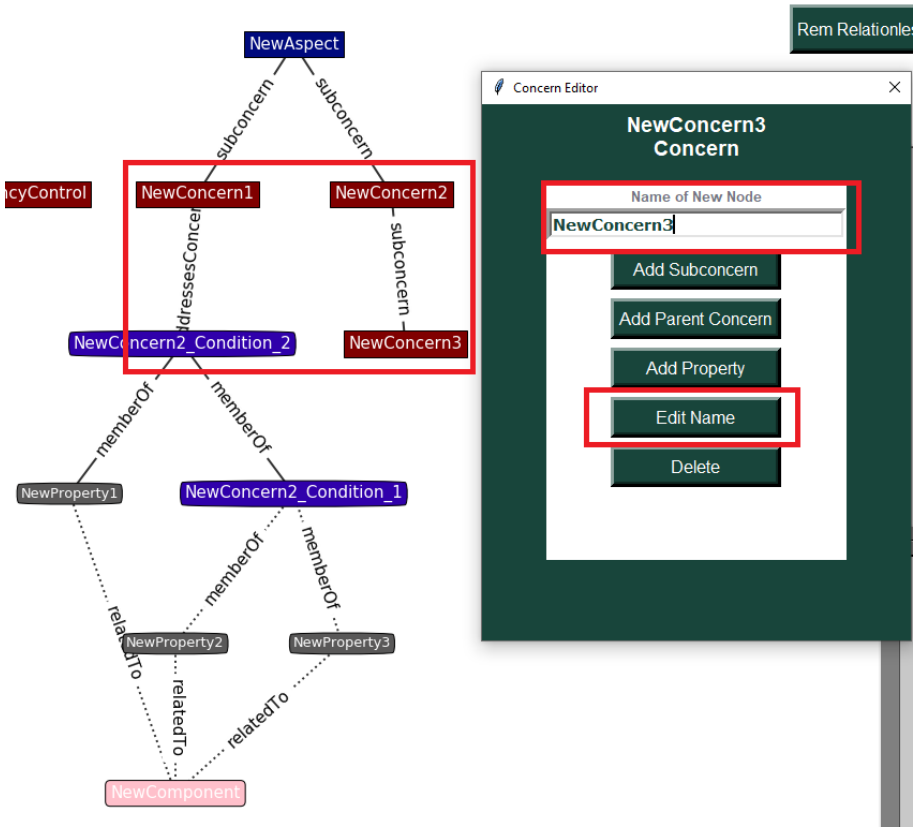
11. Delete Individual

For the example's sake, let's say we want to delete NewConcern1. Click on NewConcern1, then click Delete. Since NewConcern1 doesn't have any children it will be deleted right away with no other interaction.



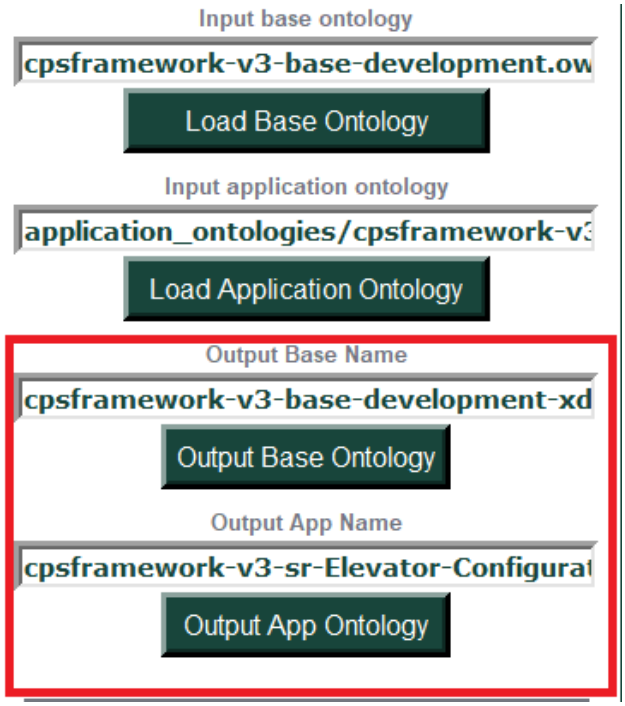
12. Edit Name of Individual

Now that NewConcern1, is no longer with us, we might want to rename NewConcern's 2-4, to go from 1-3. To edit the name of NewConcern2, left click on NewCocern2, type in the name you want to change it to, such as NewConcern1, and then click the Edit Name button. Similarly for NewConcern3 and 4, you can change their names in the same way.



13. Output Ontologies

That pretty much sums up the editor! Now that we are all done editing our ontologies, we can output them to an .owl file using the entry boxes and buttons on the left side of the editor. You can type in a filepath/name of your choice, or use the default ones in there. Click Output Base Ontology to output the base ontology, and Output Application Ontology to output the Application Ontology.


Input base ontology
cpsframework-v3-base-development.ow
Load Base Ontology
Input application ontology
application_ontologies/cpsframework-v3
Load Application Ontology
Output Base Name
cpsframework-v3-base-development-xd
Output Base Ontology
Output App Name
cpsframework-v3-sr-Elevator-Configurat
Output App Ontology