**Step-by-Step Guide for Ontology Modeling (G3-A1)**

| **Step** | **Action** | **Example/Code** | **Purpose** |
| --- | --- | --- | --- |
| **1. Define Data Properties** | Create properties to represent dropdown attributes. | **In Protégé:** - maxOptionsAllowed (range: xsd:integer) - hasSearchFunctionality (range: xsd:boolean) | Model attributes like "number of options" and "search functionality." |
| **2. Apply Restrictions to**DropdownComponent | Use General Class Axioms (GCAs) to enforce rules. | **Manchester Syntax:** (maxOptionsAllowed some xsd:integer[> 20]) SubClassOf (hasSearchFunctionality value true) | Enforce that dropdowns with >20 options **must** have search functionality. |
| **3. Create an Instance** | Add a DropdownComponent instance. | **Turtle Syntax:** :CountryDropdown a :DropdownComponent . | Represent a real-world dropdown (e.g., a country selector). |
| **4. Add Data Property Assertions** | Assign literal values to the instance. | **Turtle Syntax:** :CountryDropdown :maxOptionsAllowed 25 ; :hasSearchFunctionality true . | Set the number of options and enable search functionality. |
| **5. Add Object Property Assertions (Optional)** | Link dropdowns to design principles/metrics. | **Turtle Syntax:** :CountryDropdown :requiresPrinciple :EfficiencyPrinciple ; :impactsMetric :TimeOnTaskMetric . | Explain *why* attributes are required (e.g., efficiency). |
| **6. Run Reasoner** | Validate consistency using Protégé’s reasoner. | **Steps:** 1. Go to Reasoner → HermiT. 2. Click Start Reasoner. | Check if the instance violates restrictions (e.g., missing search functionality). |
| **7. SPARQL Query for Validation** | Retrieve compliant dropdowns. | **SPARQL Query:** sparql<br> SELECT ?dropdown<br> WHERE {<br> ?dropdown a :DropdownComponent ;<br> :maxOptionsAllowed ?options ;<br> :hasSearchFunctionality true .<br> FILTER (?options > 20)<br> }<br> | Verify which dropdowns meet the criteria. |

#### SHORT PRESENTATION: **1. Introduction to the Problem**

**Slide 1**:

* **Manual UI/UX Design Challenges**:
  + Time-consuming validation of design rules (e.g., "Do all dropdowns with >20 options have search?").
  + Inconsistent compliance with accessibility/usability standards.
* **Solution**: **Ontology + LLM Integration** to automate rule enforcement, validation, and code generation.

#### **2. Ontology as the Backbone**

**Slide 2**:

* **Defined Class:**DropdownComponent

owl

Copy

Download

Class: DropdownComponent

EquivalentTo:

(hasSearchFunctionality value true) or

(maxOptionsAllowed some xsd:integer[<= 20])

* **Purpose**:
  + **Automated Rule Enforcement**: Ensures dropdowns with >20 options must have search functionality.
  + **Reasoning**: The ontology’s reasoner (e.g., HermiT) classifies valid/invalid components dynamically.

#### **3. How It Automates UI/UX Design**

**Slide 3**:

* **Step 1: Define Rules in Ontology**
  + Example Rule:  
    "Dropdowns with >20 options require search functionality."

SPARQLE :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX onto: <http://www.semanticweb.org/asifcomputer/ontology/2025/ui\_ux\_automation#>

SELECT ?dropdown ?options

WHERE {

?dropdown rdf:type onto:DropdownComponent ;

onto:maxOptionsAllowed ?options ;

onto:hasSearchFunctionality true .

FILTER (?options > 20)

}