

TACS 2024-25 - MDSE Assignment

Phase 2 – Group 5 – Theme 2a

S1. Modeling Domain Analysis

We chose this example from the book “Canonical Abstract Prototypes for Abstract Visual and Interaction Design”:

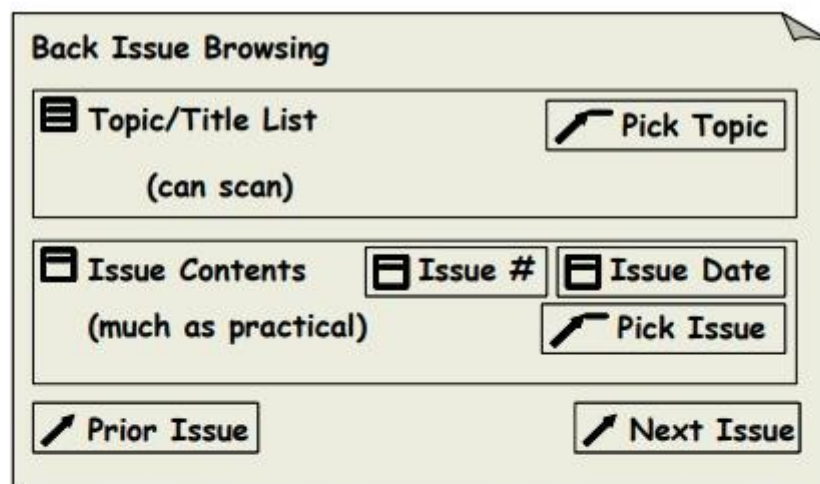


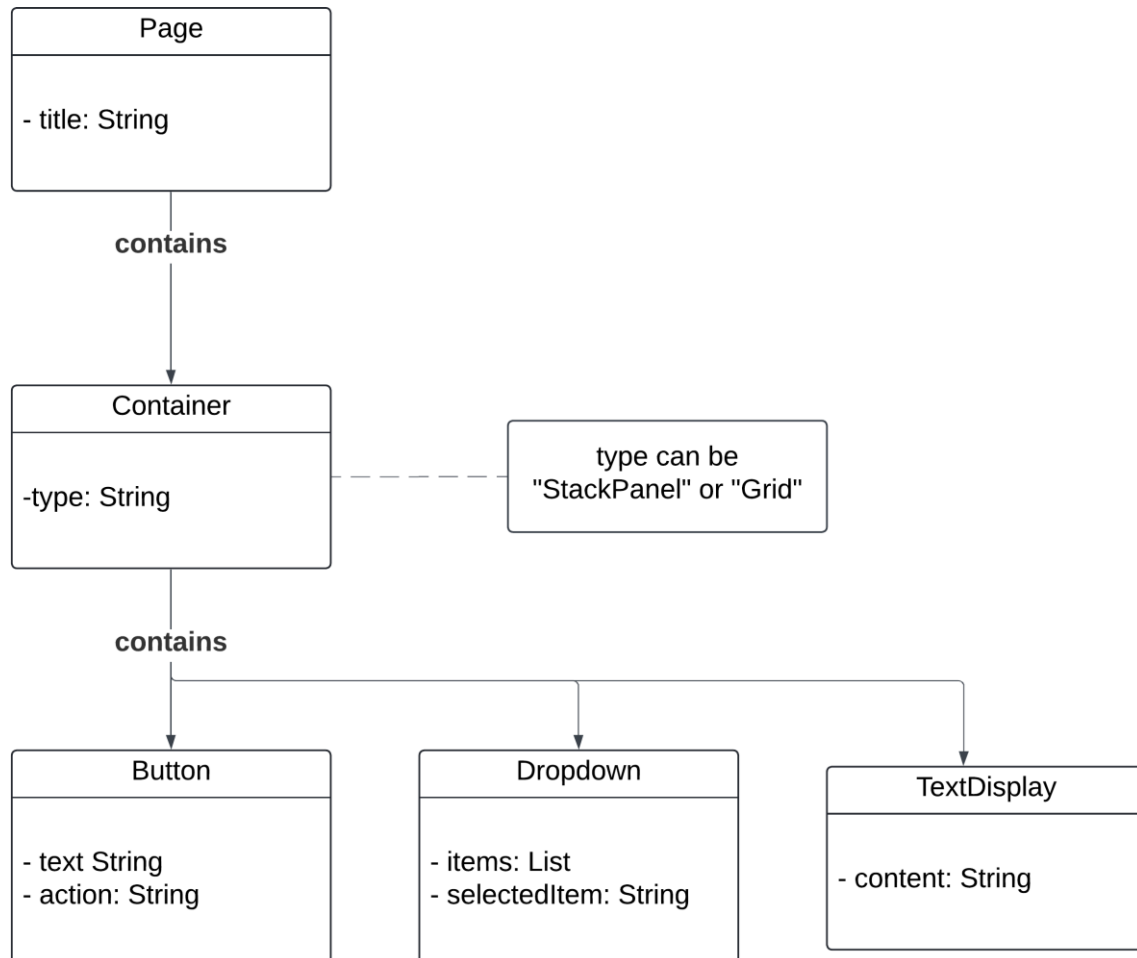
Figure 5 - Example of a Canonical Abstract Prototype for browsing back issues of a corporate newsletter by topic, issue number, or date.



Figure 6 - Example of design mockup for Canonical Abstract Prototype shown in Figure 5 with custom controls for stepping through and selecting issues and topics with synchronized contents, issues, and topics

After choosing the example, we created diagrams to help us visualize the project scope and identify the key concepts that must be a part of each metamodel.

This is the source class diagram:

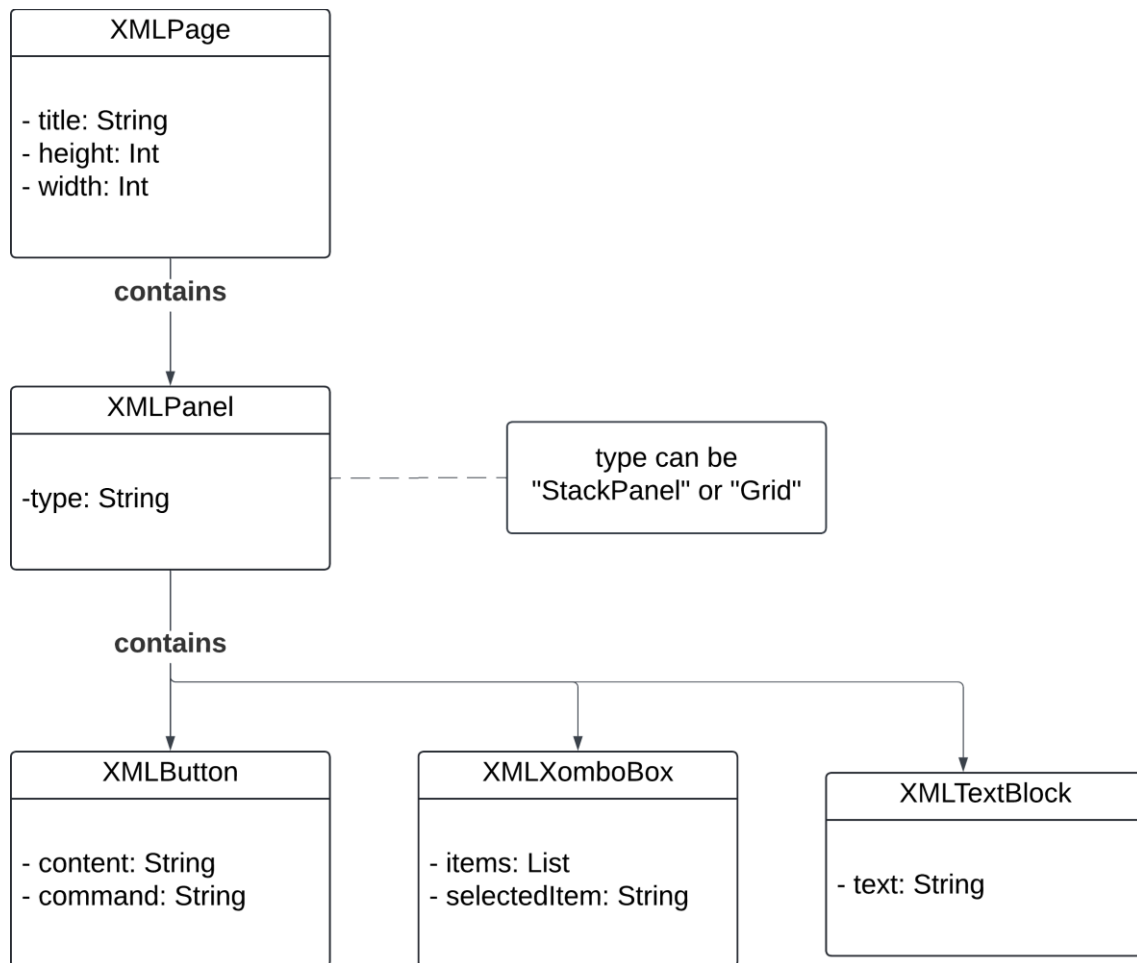


The diagram illustrates a hierarchical component structure for a page layout system. The top-level Page component contains a title property and is connected to a Container element through a "contains" relationship. The Container specifies a "type" property which can be set to either "List" or "Dropdown".

The Container element can also include three different types of UI components:

1. Button components with text and action properties
2. Dropdown components that contain a list of items and optionally a selectedItem property
3. TextDisplay components for showing string content

And this is the target class diagram:



The diagram illustrates a hierarchical XML-based user interface structure. The top-level **XMLPage** component includes three properties: a title (String), height (Int), and width (Int). It connects to an **XMLPanel** element through a "contains" relationship.

The **XMLPanel** specifies a type property that can be configured as either "StackPanel" or "Grid". This panel can contain three different types of UI components:

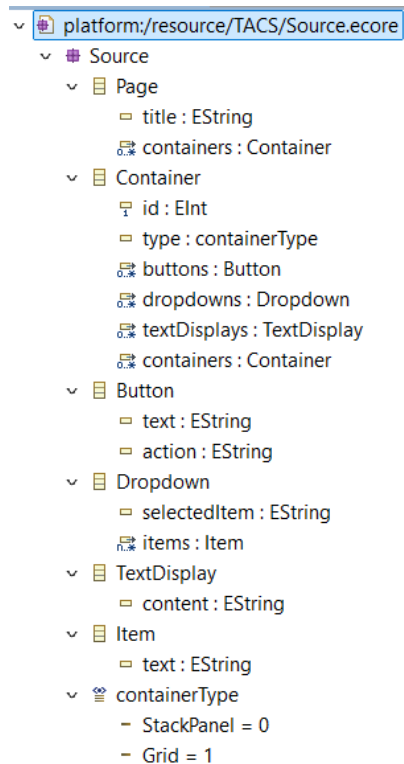
1. **XMLButton** components with content and command properties
2. **XMLComboBox** components that maintain a list of items and optionally a selectedItem property
3. **XMLTextBlock** components for displaying text content

S2. Modeling Language (Metamodel) Design

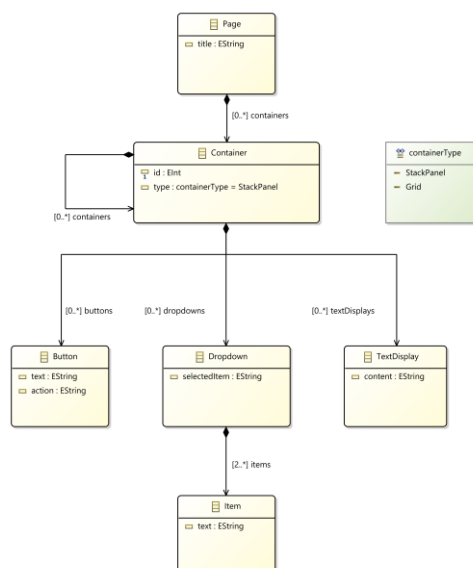
S2a. This part of the report details the creation of source and target metamodels in Ecore based on the hierarchical component structures described. It covers the visual representation of these metamodels, and the integrity constraints implemented using Object Constraint Language (OCL).

- Source Model:

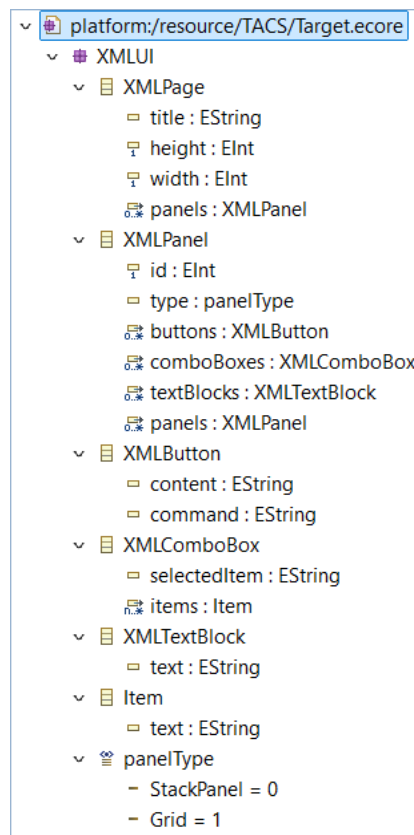
o Ecore Model



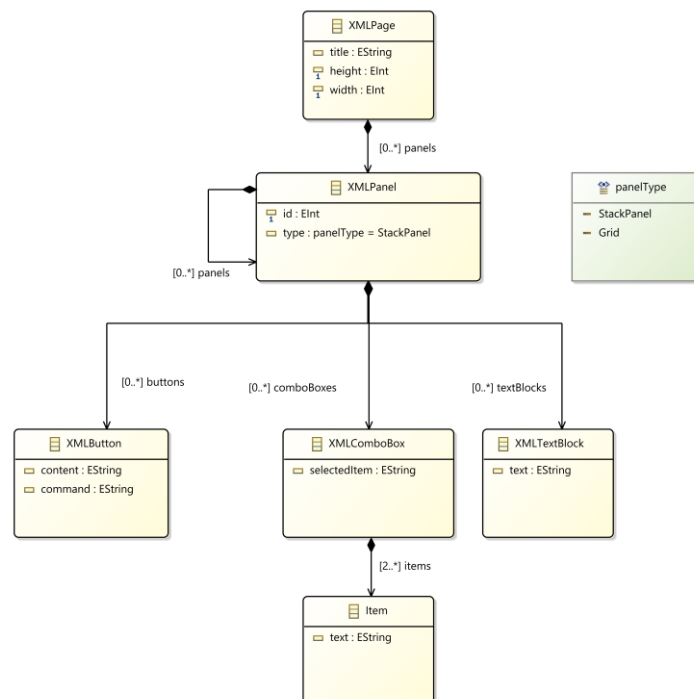
o Diagram



- **Target Model:**
 - o Ecore Model



- o Diagram



In the updated design, we've enhanced the source and target class diagrams to reflect more detailed and flexible component structures.

- Source Model

- Nested Containers - we introduced the possibility for a Container element to contain other Container elements, which allows for a more hierarchical structure. This change enables the creation of nested layouts where containers can hold additional containers, thus supporting more complex page designs.
- Container Type Enumeration - to provide greater flexibility and ensure consistency in defining the type of containers, we added an enumeration (enum) to the Container element. This enum restricts the container type to predefined values, such as "StackPanel" or "Grid", ensuring that only valid container types can be used. This helps maintain the integrity and clarity of the model by providing a controlled set of options for the container type.

- Target Model

- Nested Panels - similar to the source model, we allowed for the XMLPanel element to contain other XMLPanel elements, providing a hierarchical structure for the panels. This change supports complex layouts where panels can hold additional panels, enabling more flexible UI compositions.
- Panel Type Enumeration - to ensure uniformity in defining the types of panels, we added an enum to the XMLPanel element. This enum specifies the possible panel types, such as "StackPanel" or "Grid". By using this enum, we guarantee that only valid panel types can be assigned, which improves the clarity and consistency of the panel's structure and behavior.

These updates reflect our effort to build more flexible, hierarchical user interface layouts while maintaining strict control over the types and relationships between components.

S2b. In this section, we document integrity constraints expressed in the Object Constraint Language (OCL). These constraints ensure the structural and semantic consistency of the models within the defined domain and enforce specific business rules. Below, we provide a detailed explanation of the OCL constraints applied to both the source and target metamodels.

- **Source Model:**

- On Container:

- Unique Button Constraint - guarantees that no two buttons within the same container have identical text and action attributes.

```
invariant uniqueButtonTexts:  
self.buttons->forAll(b1, b2 | b1 <> b2 implies b1.text <> b2.text);  
invariant uniqueButtonActions:  
self.buttons->forAll(b1, b2 | b1 <> b2 implies b1.action <> b2.action);
```

- No Self-Containment Constraint - ensures that no object is contained within itself, either directly or indirectly, within the containment hierarchy.

```
invariant noSelfContainment:  
not self.containers->closure(p | p.containers)->exists(c | c.id =  
self.id);
```

- On Dropdown:

- Selected Item Exists in Items List Constraint - ensures that the selected item in the dropdown is part of the items list.

```
invariant selectedItemMustBeInItems:  
self.items->exists(item | item.text = self.selectedItem);
```

- **Target Model:**

- On XMLPage:

- Valid Height and Width Constraint - restricts the height and width attributes of an XMLPage to be between 50 and 3000.

```
invariant validHeight:  
self.height >= 50 and self.height <= 3000;  
invariant validWidth:  
self.width >= 50 and self.width <= 3000;
```

- On XMLPanel:

- Unique Button Constraint - guarantees that no two buttons in the same panel have identical content and command.

```
invariant uniqueButtonContent:  
self.buttons->forAll(b1, b2 | b1 <> b2 implies b1.content <>  
b2.content);  
invariant uniqueButtonCommand:
```

```
self.buttons->forall(b1, b2 | b1 <> b2 implies b1.command <>
b2.command);
```

- No Self-Containment Constraint - ensures that no object is contained within itself, either directly or indirectly, within the containment hierarchy.

```
invariant noSelfContainment:
not self.panels->closure(p | p.panels)->exists(c | c.id = self.id);
```

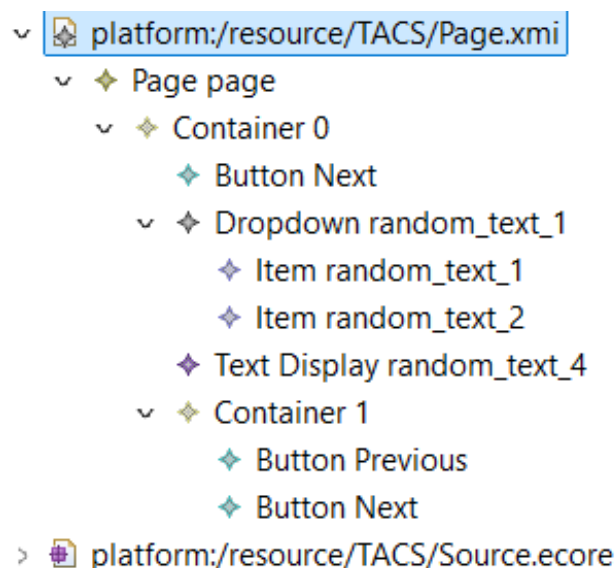
- On XMLComboBox:
 - Selected Item Exists in Items List Constraint - ensures that the selected item in the combo box is part of the items list.

```
invariant selectedItemMustBeInItems:
self.items->exists(item | item.text = self.selectedItem);
```

S3. Modeling Language (Metamodel) Validation

This section presents the validation of the source and target metamodels created in Phase 2. The validation ensures that the metamodels accurately define the intended domain and enforce their structural and behavioral constraints. Validation was performed using encoded examples and additional test cases to comprehensively evaluate all features of the metamodels.

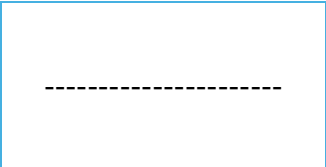
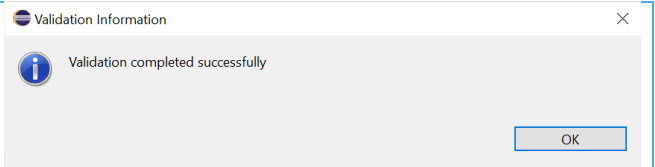
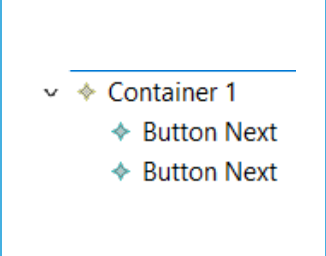
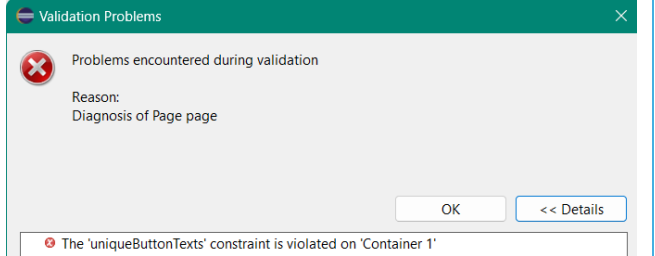
For the Source Model we created a Page with a Container that has a Button, a Dropdown with 2 Items and a Text Display and a Container Grid with two buttons.



The table summarizes the test cases for the source metamodel, highlighting their objectives, modified parameters, expected results, and validation outcomes.

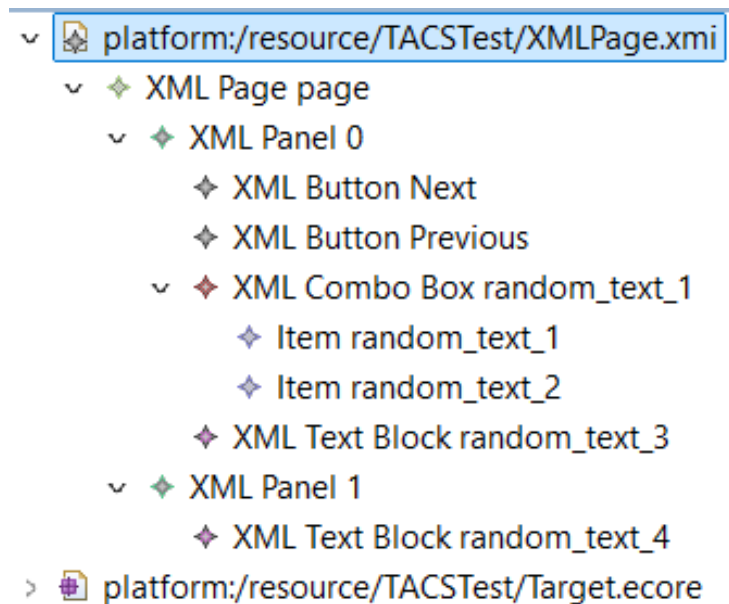
Test Case ID	Description	Changed Parameter	Expected Result	Outcome
S1	Valid configuration of Page with a Container	None	Passes all constraints	Passed
S2	Duplicate Button texts in the same Container	Two Buttons with text="next"	Fails uniqueButtonTexts constraint	Passed (Error Detected)
S3	Duplicate Button actions in the same Container	Two Buttons with action = "previous"	Fails uniqueButtonActions constraint	Passed (Error Detected)
S4	Self-containment cycle in Container	Object added as a container of itself	Fails noSelfContainment constraint	Passed (Error Detected)
S5	Dropdown.selectedItem not in Dropdown.items	The selectedItem attributed isn't part of the items list	Fails selectedItemMustBeInItems constraint	Passed (Error Detected)

The table outlines the test cases with corresponding parameter changes and visual representations of the detected errors during validation.

Test Case ID	Changed Parameter Image	Image of type of Error Detected on validation
S1		
S2		

S3	<table><tr><th>Property</th><th>Value</th></tr><tr><td>Command</td><td>previous</td></tr><tr><td>Content</td><td>Previous</td></tr></table>	Property	Value	Command	previous	Content	Previous	<div><div>Validation Problems</div><div><div></div><div>Problems encountered during validation</div><div>Reason: Diagnosis of Page page</div></div><div><div>OK</div><div><< Details</div></div><div>The 'uniqueButtonActions' constraint is violated on 'Container 1'</div></div>
Property	Value							
Command	previous							
Content	Previous							
S4	<div>Container 1<ul style="list-style-type: none">Button PreviousButton NextContainer 1</div>	<div><div>Validation Problems</div><div><div></div><div>Problems encountered during validation</div><div>Reason: Diagnosis of Page page</div></div><div><div>OK</div><div><< Details</div></div><div><div>The 'noSelfContainment' constraint is violated on 'Container 1'</div><div>The ID '1' of 'Container 1' collides with that of 'Container 1'</div></div></div>						
S5	<div>Dropdown random_text_1<ul style="list-style-type: none">Item random_text_2Item random_text_3</div>	<div><div>Validation Problems</div><div><div></div><div>Problems encountered during validation</div><div>Reason: Diagnosis of Page page</div></div><div><div>OK</div><div><< Details</div></div><div>The 'selectedItemMustBeInItems' constraint is violated on 'Dropdown random_text_1'</div></div>						

For the Target Model we created a Page with a Panel that has two Buttons, a ComboBox with 2 Items and a Text Block.

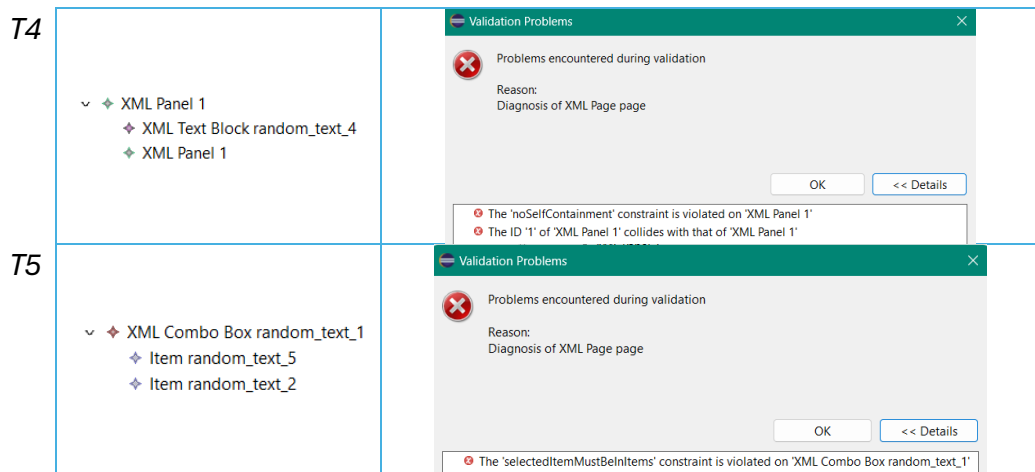


The table summarizes the test cases for the target metamodel, highlighting their objectives, modified parameters, expected results, and validation outcomes.

Test Case ID	Description	Changed Parameter	Expected Result	Outcome
T1	Valid configuration of XMLPage	None	Passes all constraints	Passed
T2	Invalid XMLPage dimensions	height = 4000, width = 6	Fails validHeight and validWidth constraints	Passed (Error Detected)
T3	Duplicate Button in the same XMLPanel	Two Buttons with content = "previous" and command = "back"	Fails uniqueButtonContent constraint	Passed (Error Detected)
T4	Self-containment cycle in Panel	Object added as a panel of itself	Fails noSelfContainment constraint	Passed (Error Detected)
T5	XMLComboBox.selectedItem not in items	selectedItem = "Invalid"	Fails selectedItemMustBeInItems constraint	Passed (Error Detected)

The table outlines the test cases with corresponding parameter changes and visual representations of the detected errors during validation.

Test Case ID	Changed Parameter Image	Image of type of Error Detected on validation								
T1										
T2	<table><thead><tr><th>Property</th><th>Value</th></tr></thead><tbody><tr><td>Height</td><td>500</td></tr><tr><td>Title</td><td>page</td></tr><tr><td>Width</td><td>500</td></tr></tbody></table>	Property	Value	Height	500	Title	page	Width	500	
Property	Value									
Height	500									
Title	page									
Width	500									
T3	<table><thead><tr><th>Property</th><th>Value</th></tr></thead><tbody><tr><td>Command</td><td>previous</td></tr><tr><td>Content</td><td>Back</td></tr></tbody></table>	Property	Value	Command	previous	Content	Back			
Property	Value									
Command	previous									
Content	Back									



S5. Define Model-to-Model (M2M) Transformation with ATL

S5a. M2M Design

The goal of the Model-to-Model (M2M) transformation is to map elements from the source metamodel (Page and its components) to corresponding elements in the target metamodel (XMLPage and its components). In this design, we mapped each element in the source model to an appropriate target model element. The key mappings include:

- **Page → XMLPage:** The Page element in the source model corresponds to the XMLPage element in the target model. This includes transferring attributes such as title, height, and width.
- **Container → XMLPanel:** The Container in the source model is mapped to the XMLPanel in the target model. The type property (with values like "List" or "Dropdown") is mapped to the type property of XMLPanel (with values "StackPanel" or "Grid").
- **Button → XMLButton:** The Button in the source model is mapped to XMLButton, transferring properties like text and action.
- **Dropdown → XMLComboBox:** The Dropdown is mapped to XMLComboBox, including properties like items and selectedItem.
- **TextDisplay → XMLTextBlock:** The TextDisplay in the source model is mapped to XMLTextBlock in the target model, which displays string content.

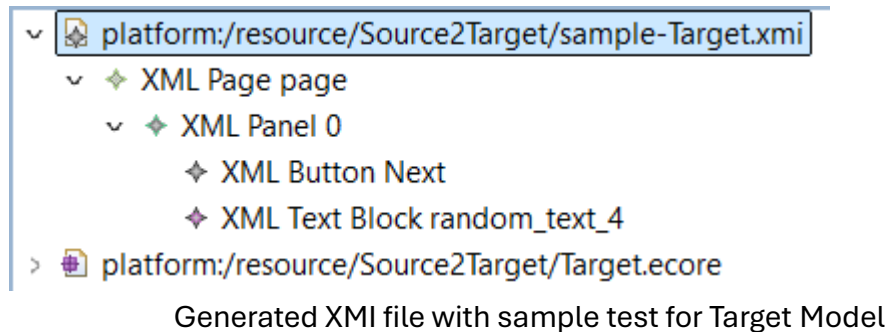
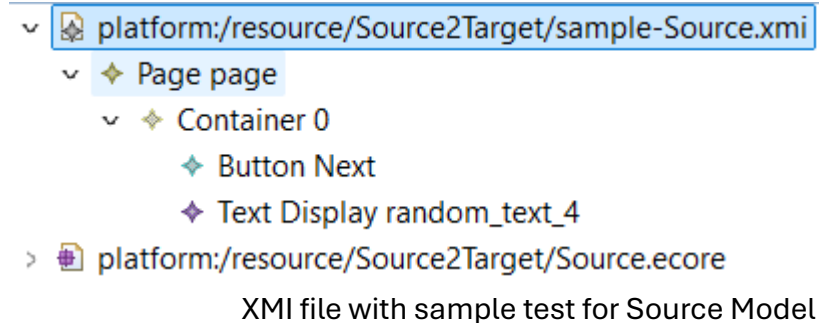
S5b. M2M Implementation

The M2M transformation was implemented using ATL (Atlas Transformation Language). The ATL code reflects the design by defining transformation rules for each key component. These rules convert source model elements into their respective target model elements and assign the corresponding attributes.

S5c. M2M Testing

The M2M transformation was tested using the test case developed in Phase

1.



The test cases ensured that the mappings were correct.

S6. Define Model-to-Text (M2T) Transformation with Acceleo

S6a. M2T Design

The goal of the Model-to-Text (M2T) transformation is to convert the target model elements into a textual representation (e.g., HTML, XAML). The design of the M2T transformation involved specifying how the model elements should be converted into text:

- **XMLPage → HTML:** The XMLPage element in the target model is converted into an HTML page with its attributes (title, height, and width) inserted as HTML tags.
- **XMLPanel → HTML Container:** The XMLPanel type is converted into a corresponding HTML container element (<div>) with its type attribute determining the layout (e.g., StackPanel maps to a div with a vertical layout).

- **XMLButton → HTML Button:** The XMLButton is mapped to an HTML <button> element, where the content is used as the button text and the command is used as a JavaScript action.
- **XMLComboBox → HTML Dropdown:** The XMLComboBox is converted into an HTML <select> element with <option> elements representing the items.

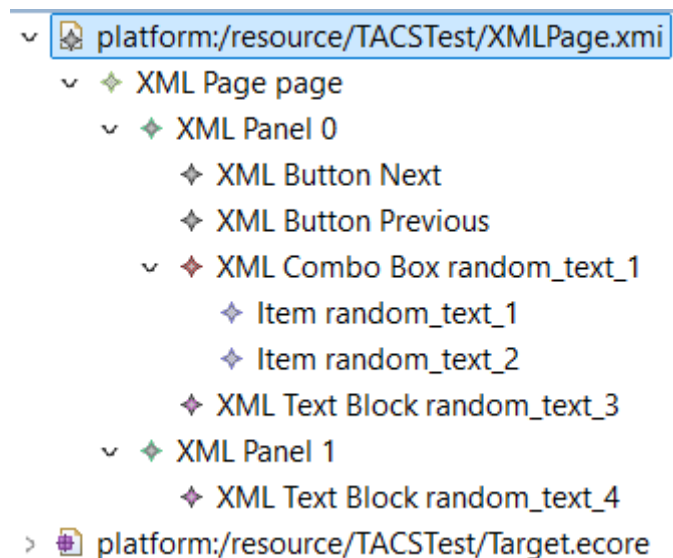
The design was formalized through Acceleo templates, and the conversion rules were defined for each model element.

S6b. M2T Implementation

The M2T transformation was implemented using Acceleo templates. This Acceleo code takes elements from the target model and generates HTML code. Each element is converted based on its attributes, ensuring the correct output format.

S6c. M2T Testing

The M2T transformation was tested using the test cases from Phase 1.



XML file with sample test for Target Model

```

<?xml version="1.0" encoding="UTF-8"?>
<TACS:XMLPage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:TACS="http://TACS"
  title="page"
  height="500"
  width="500">

  <panels>
    <TACS:XMLPanel id="0" type="Grid">
      <buttons>
        <TACS:XMLButton content="Next" command="next"/>
        <TACS:XMLButton content="Previous" command="previous"/>
      </buttons>

      <comboBoxes>
        <TACS:XMLComboBox selectedItem="random_text_1">
          <items>
            <TACS:Item text="random_text_1"/>
            <TACS:Item text="random_text_2"/>
          </items>
        </TACS:XMLComboBox>
      </comboBoxes>

      <textBlocks>
        <TACS:XMLTextBlock text="random_text_3"/>
      </textBlocks>

    </TACS:XMLPanel>
    <TACS:XMLPanel id="1" type="StackPanel">

      <textBlocks>
        <TACS:XMLTextBlock text="random_text_4"/>
      </textBlocks>

    </TACS:XMLPanel>
  </panels>
</TACS:XMLPage>

```

XAML file with the generated code

Any discrepancies between the expected output and generated text were promptly addressed by adjusting the Acceleo templates.

Phase 1 Updates and Refinements

The main changes in Phase 1 focused on improving the structure and organization of the report to enhance clarity and readability. While the core information remained consistent, it was reformatted for better presentation. The following refinements were made to the metamodel:

- Enumeration for Container and XMLPanel Types - we introduced an enumeration (enum) for both the Container and XMLPanel elements to specify the type attribute. Instead of allowing any arbitrary string, the type attribute is now restricted to predefined values. The valid types are

- "StackPanel" and "Grid". This change improves the consistency of the model and ensures that only valid, predefined values are used for the type attribute.
- Support for Nested Containers - we implemented support for nested Containers and nested Panels, allowing a Container to contain other Container elements, and an XMLPanel to contain other XMLPanel elements. This enhances the flexibility in designing hierarchical layouts for both containers and panels. To maintain model consistency and prevent logical errors, we added constraints to ensure that neither a Container nor an XMLPanel can reference itself, either directly or indirectly. This prevents circular references and preserves the integrity of the model, while still enabling more complex layout configurations.

Team Contributions, Achievements, and Challenges

The project tasks were divided equally among the team. Rita and José Carlos worked on the source metamodel, while Sofia and Luís focused on the target metamodel. For the second part, Sofia and José Carlos handled the Model-to-Model (M2M) transformation, and Rita and Luís worked on the Model-to-Text (M2T) transformation. Despite the division, everyone contributed across all areas due to the difficulties encountered with Eclipse.

We successfully created both the source and target metamodels, implemented the transformations, and learned about model-driven development, Eclipse tools, and validation techniques. However, working with Eclipse was the main challenge. Although we understood the theoretical concepts, the tool often didn't function as expected, leading to frustration and delays. Despite these setbacks, we were able to complete the project through effective collaboration and perseverance.

References

1. Design and Modeling Approaches

- a. *Canonical Abstract Prototypes for Abstract Visual and Interaction Design*. Available at:
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=29672236c11e4e2dbb51670f7f07a04c05adf035>

2. Eclipse Modeling Framework (EMF)

- a. *EMF Tutorial*. Available at:
<https://eclipsesource.com/blogs/tutorials/emf-tutorial/>

3. Model Transformation using ATL

- a. *ATL Tutorials: Create a Simple ATL Transformation*. Available at:
[https://wiki.eclipse.org/ATL/Tutorials -
_Create_a_simple_ATL_transformation](https://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation)

4. Object Constraint Language (OCL)

- a. *OCL Tutorials*. Available at:
[https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.ocl.doc
%2Fhelp%2FTutorials.html](https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FTutorials.html)

5. Acceleo Code Generation

- a. *Acceleo User Guide*. Available at:
https://wiki.eclipse.org/Acceleo/User_Guide

Group - G5

UP number	Name	Contribution
202008569	Ana Rita de Oliveira Carneiro	25%
201806629	Ana Sofia Oliveira Teixeira	25%
201905451	José Carlos Cardiano Mota Gonçalves da Cunha	25%
201306340	Luís Paulo da Rocha Miranda	25%