**Just Write**

**Use Case Document**

**April 19, 2012**

**IAT 355**: Developing Design Tools

Project Phase 2

- Lanz Singbeil

- Ethan Johanson

- Nathan Tsang

**Table of contents**

3 Use Case Diagrams

4 D01: Create Node

5 D02: Edit Node

6 Edit Node UI Sketch

7 I01: Write Simple Text

8 I02: Edit Paragraph

9 Edit Paragraph Macro example

10 V01: Change Branching of a WrittenElement

11 D03: Split an Event

12 Split an Event UI Sketch

13 D04: Merge 2 Events

14 T01: Change Time index of an event UI Example

15 Change Time index of an event UI Example

**Our use cases are labeled with different Letters:**

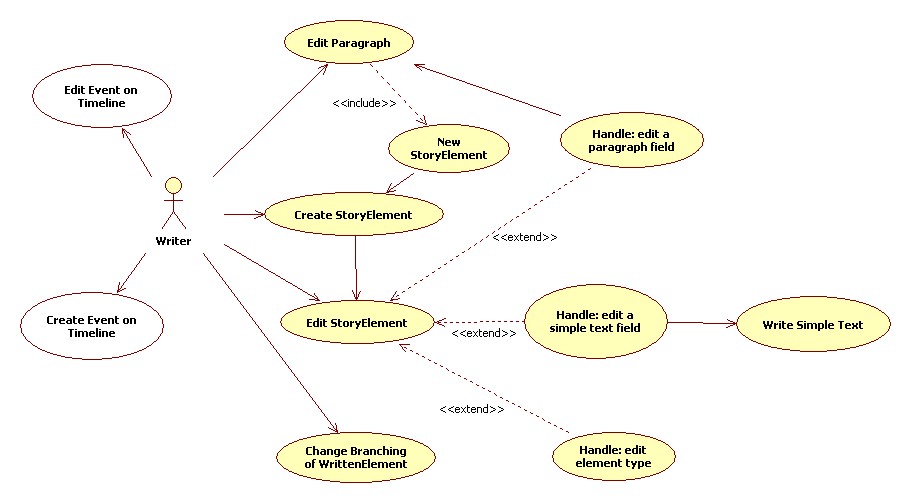
**“D”** use cases are primarily concerned with the manipulation of domain objects

**“I”** use cases are primarily concerned with interaction

“V” use cases are primarily concerned with how the view is managed

“T” use cases are primarily concerned with interaction with the timeline

**Use Case Diagram:**

****

**Use Case D01: Create Story Element**  
Importance: High  
Design Scope: System Level  
Trigger: Writer gives the system the name of a new story element and the command to create one.  
  
Include: Edit Story Element  
Extend: ...  
  
Brief Description:  This use case demonstrates how a regular story element is created.  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer chooses to create a new story element with a given name  
2. System checks if that element already exists.  
    If StoryElement already exists:  
        the S-1: Handle Name Issue sub-flow is performed  
    If StoryElement doesn’t already exist:  
        Continue use case at step 3.  
3. Trigger the Edit Story Element use case  
  
Sub-Flow:  
S-1: Handle Name Issue  
    1. System asks user if they wish to overwrite the old story element  
        If user decides to overwrite:  
            the system replaces the previous node with the new one.  
        If user decides to cancel:  
            Trigger Exception a.  
    2. Continue use case at step 3.  
  
Exceptions:  
  
a. Writer cancels the operation:

1. System erases all content that was written during this operation.

2. End of use case

**Use Case D02: Edit Story Element**  
Importance: High  
Design Scope: System Level  
Trigger: Writer chooses to edit this element from somewhere else in the system  
  
Brief Description: This use case demonstrates how a story element is edited. Depending on the type of element, they may have different properties. When significant, these are handled in alternate flows.  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer chooses to edit a story element  
2. System opens a window displaying this node’s current properties  
3. Writer chooses the fields to change and edits them.  
4. step 3 may be repeated.  
5. Writer confirms that they are finished. System stores changes  
  
Alternate Flows:  
  
3a. The writer edits a paragraph field: the Edit Paragraph use case is performed  
3b. The writer edits a text description field: the Write Simple Textuse case is performed  
3c. The writer edits the node’s type:  
1. The system updates the fields according to the new node type  
    2. use case resumes at step 2 with new fields.  
  
Exceptions:  
  
2a. Writer closes the element window:

1. System asks writer to save or discard any changes.

    If the Writer chooses to save changes...

        the changes to properties are stored and the window is closed

    If the Writer chooses to discard changes...

        the window is closed and all properties revert to previous values.

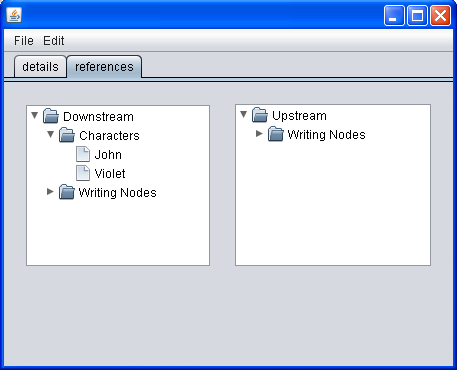
            If no previous values existed...

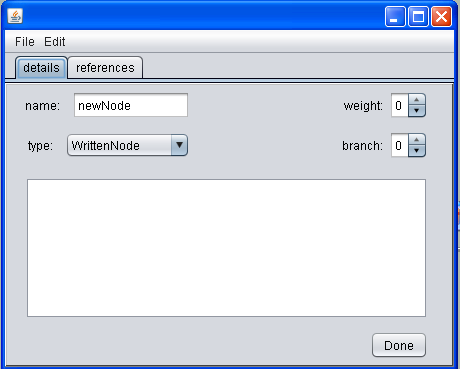
                the story element is deleted

    If the Writer chooses to cancel their action...

        the window remains open.

**Edit Node UI Example**

****

****

**Use Case I01: Write Simple Text**  
Importance: Low  
Design Scope: System Level  
Trigger: Writer activates a block of text to edit  
  
Brief Description:  This use case demonstrates how a simple text block behaves when edited.  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer selects an editable simple text block and system displays it as active  
2. Writer adds or removes content from the text block  
3. Writer confirms that they are finished. System stores content.  
4. 1 to 3 may be repeated.  
  
Exceptions:  
  
a. Writer cancels the operation:

1. System erases all content that was written during this operation. Previous content remains.

2. End of use case

**Use Case I02: Edit Paragraph**  
Importance: High  
Design Scope: System Level  
Trigger: Writer activates a paragraph to edit  
  
Include:  
Extend: Create Node  
  
Brief Description: This use case demonstrates how a paragraph block behaves when edited.  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer selects a paragraph block and the system displays it as active  
2. Writer adds or removes content from the paragraph    
3. Writer confirms that they are finished. System stores content.  
4. 1 to 3 may be repeated.  
  
Sub-Flows:  
  
SA-1. New Node  
    1. System executes the Create Node use case  
    2. System associates the node with this paragraph  
  
Alternate Flows:  
  
2a. Add an in-line node to the paragraph:  
    1. System checks the name of this in-line node with all others  
        If the node’s name doesn’t already exist...

the S-1: New Node sub-flow is performed.

If the node’s name matches an existing node:

    System associates the node with this paragraph

    2. Use case continues with step 3. of Main Flow.

Exceptions:  
  
\_a. Writer cancels the operation:

1. System erases all content that was written during this operation. Previous content remains.

2. End of use case

2a. Writer deletes an in-line reference to a node:

1. System removes this paragraph’s association to that node.

2. Use case resumes step 2.

**Edit Paragraph Macro Example:**

|  |
| --- |
| Event 1 <name>: <list[0].name>, …, <list[n].name>  In this chapter, we join our [hero bob] in his amazing adventures in [funville] as he combats the evil [hunger-beast antagonist]. He first needed to visit the [bakery] to see if he could get any bread to use as a defense against the [beast antagonist]. |

case 1: [hero bob] “hero” is not already a node but adding the word “bob” after a space separator lets the user link to that pre-defined node if it exists. In this case it does and is already associated with the event (it is one of the elements in the unordered list). That node is now locked and cannot be moved from this event.

case 2: [funville] “funville” is the name of a node that is already associated with this event. That node in the unordered list is now locked and cannot be moved from this event.

case 3: [hunger-beast antagonist] similar to case 1 except that “antagonist” isn’t associated with this event. It exists in the public list of available nodes and so is added to the events unordered list and locked in.

case 4: [bakery] this word doesn’t refer to any nodes associated with this event and it also doesn’t refer to any public nodes that exist. A new node is created and added to the event as well as to the public list. The node is locked by default since it is attached to a word.

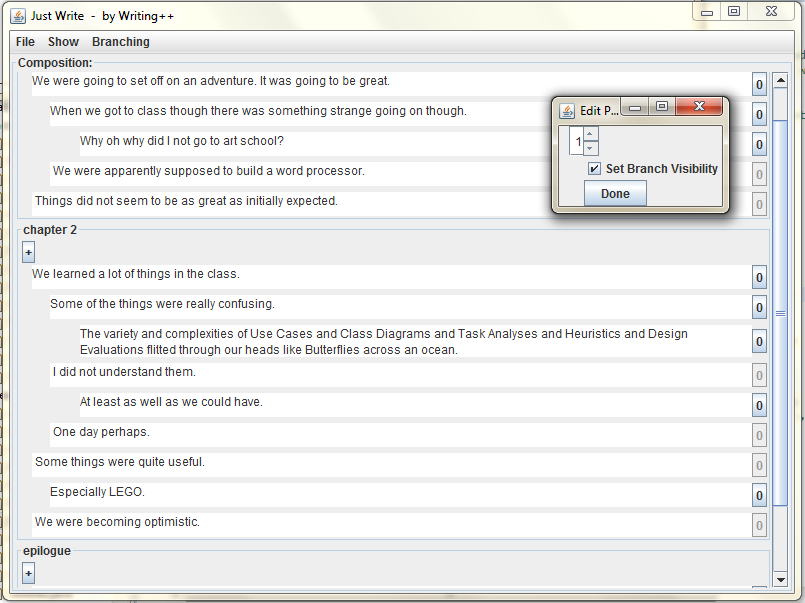
**Use Case V01: Change Branching of a Written Element**  
Importance: High  
Design Scope: System Level  
Trigger: Writer specifies they want to adjust branch settings for this written element

Brief Description: This use case demonstrates how the user changes the settings within the BranchEditor and how that will affect the model and view.  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer selects a written element with a branch they want to change  
2. System displays a BranchEditor that allows them to change the Element’s branch or adjust visibility of that element and confirms. (The BranchEditor closes)

3. The System checks if any changes need to be made. If so, it adjusts the model.

4. The view will update to hide and show branches as needed.

Exceptions:  
  
2a. Writer cancels the operation: End of use case



**Use Case D03: Split an Event**

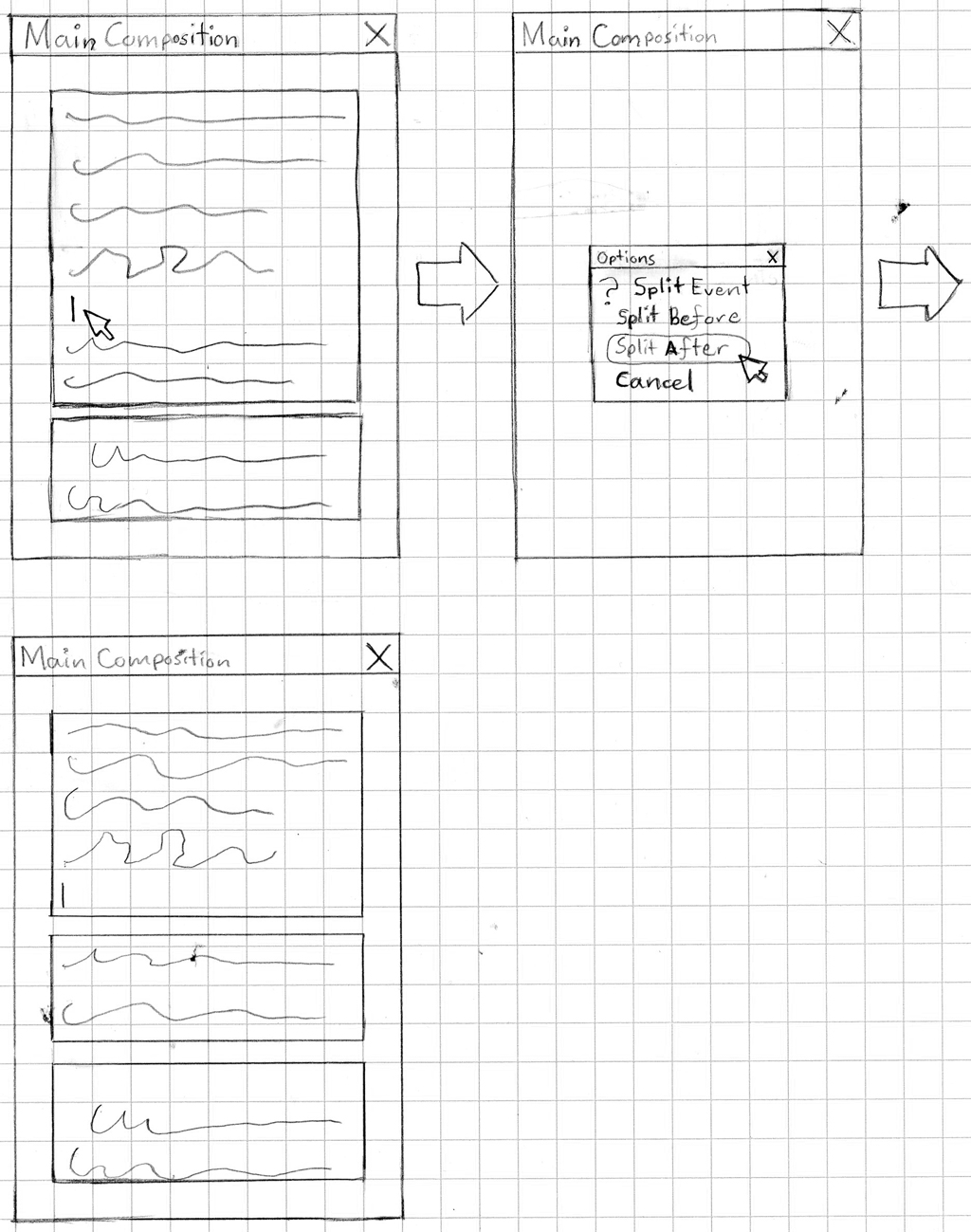
Importance: Medium  
Design Scope: System Level  
Trigger: the Writer specifies that they want to split an event at a specific location.  
  
Include:  
Extend:  
  
Brief Description: This use case demonstrates how an event can be split, divided into two new events that contain divided portions of the previous event’s content.  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer specifies a position to split an event  
2. System asks the user if they want to split before or after the current line position

3. Writer selects their choice and the appropriate division occurs.

4. System updates view to show two events where there once was one.

Exceptions:  
  
2a. Writer cancels the operation: End of use case

**Split an Event UI Example**

****

**Use Case D04: Merge 2 Events**

Importance: Medium  
Design Scope: System Level  
Trigger: the Writer specifies that they want to merge 2 events into a single event.  
  
Include:  
Extend:  
  
Brief Description: This use case demonstrates how 2 events are merged  
  
Primary Actor: Writer  
  
Main Flow:  
1. Writer specifies two events they would like to merge

2. System asks the user for confirmation on this action.

If writer confirms, continue use case at step 3.

If writer cancels, end use case.

3. Out of the two events being merged, the writer picks which one is the dominant event.

4. System copies all content from the passive event into the dominant event (Text content is appended at the end) and then discards the passive event.

5. System updates view to show a single event where there once were two.

**Use Case T01: Change Time index of an event**

Importance: High  
Design Scope: System Level  
Trigger: direct interaction with an event in the timeline window  
  
Include:  
Extend:  
  
Brief Description: This use case demonstrates how an event’s time index can be changed by the user very easily when they use the timeline.

Primary Actor: Writer  
  
Main Flow:  
1. Writer selects the visual representation of an event on the timeline

2. Writer moves it to a new position

3. System updates the model event’s time index

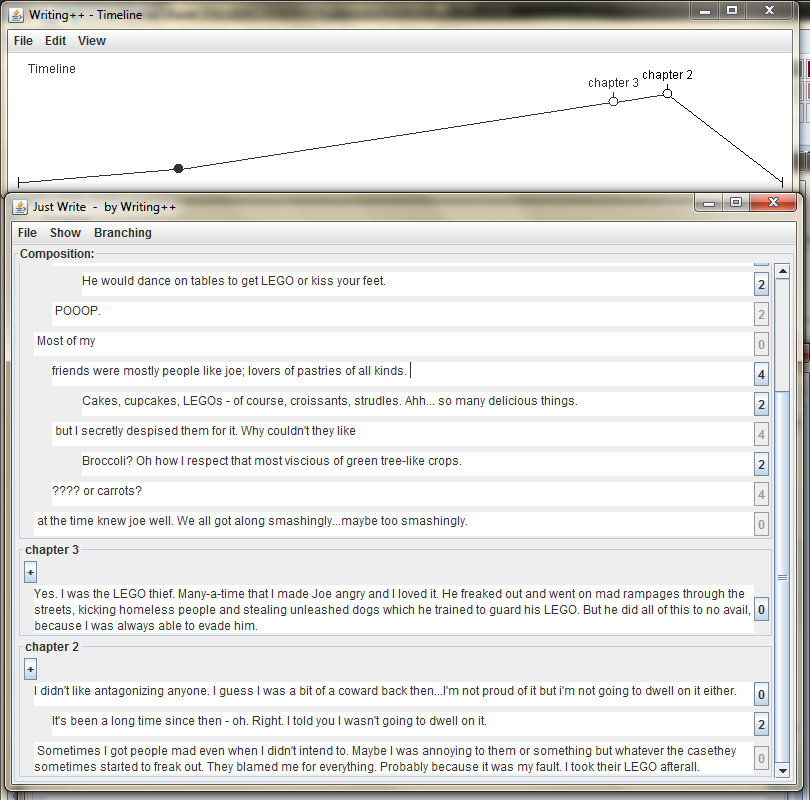
4. System checks if the event order in the model has been changed as a result of the movement.

If yes: System notifies controllers that the display of events should be updated

System updates the display to show proper event ordering in all views.

**Change Time index of an event UI Example**

**Before:**



After:

