The SlideMeister Concept

# Idea

There are several occasions when one architectural design, one image or even just one diagram is required to be shown and several phases of an activity needs to be shown.

Examples are:

* Animated circuitry of switch positions
* Process description with blue and red lines depending on state of the system
* Parallel State machines

The application is successful when a non-IT person can create a slide with some toggling of pictures being controlled by a small script to support a pre-defined animation.

## Technical constraints

As a base, the WPF framework will be used. This allows the usage on Windows 7 computers which are quite common in the industry. Using Windows 10 UWP platform might only target private users since corporate networks have no opportunity.

The application will run on .Net Platform to ease development for myself. I’m not so familiar with Java, C++ and other frameworks.

# Concept

## Use Cases

The application supports the following use-cases:

### Manual triggering

One diagram will be shown and the user can directly modify the diagram by clicking on certain elements. This will allow the user to explain schematics depending on certain switch positions and internal states.

### Automatic sequence

The diagram and its switch position will be predefined by the user and can be shown as a sequence within the screen.

The user can decide whether each transition will be forwarded automatically or if he can navigate manually between several transitions.

## Glossary

### Machine

The non-static image that will always be shown and is overlayed by the elements which can have several states and their corresponding image.

Each machine has a name, a version number which is shown in the title of the application and in one of the borders of the application.

### OverlayItem

Defines the item that is overlaying the background and can be in several states.

### OverlayType

A definition if possible states and their corresponding visual look of an overlay item. This will allow the reuse of types of the items without the need to redefine them at every occurrence

### OverlayImage

One of the possible overlaytypes of an item. Each state is visually represented by a certain image.

### OverlayState

Defines one possible state of the item. A state is connected with a very specific image that will be shown to the user.

### Transition

A transition moves the OverlayItem from one state to another state. A transition can be triggered automatically by a transition sequence or can be triggered manually by the user

### Transition Set

A set of transitions that will be applied to several items on the background. If one item does not have a transition with the set, this item will keep as it is.

### Transition Sequence

Several transition sets will be executed sequentially within one transition sequence. A transition sequence is usually user-defined and can be shown for demonstration purposes

### Technical definition

The application will register the script files to the “.slidemeister”.

## Definition language

### Philosophy and elements

Every user with some technical background (capable to write text files in a basic text editor) needs to be able to create a .slidemeister script. A json format is considered as sufficiently easy to be written y users.

The .json file stores the complete definition of a machine, including states, sequences and items. Each file stores one complete machine which is located at the root-level of the file.

The definition shall not have any requirement upon the order of the elements.

#### Machine

The root element with the following attributes:

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| items | OverlayItem\* | Defines the set of overlayitems of the machine |
| types | OverlayType\* | Defines possible types of the items |
| sequences | TransitionSequence\* |  |
| name | String | Name of the machine being shown in the title |
| version | String | Version of the machine to be shown as clear reference |
| transitions | TransitionSet\* | The transition sets being defined in the machine |

#### OverlayType

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| \* | String | Unique name of the overlay type |
| type | String | Type of the overlaytype. If there is no definition, image will be assumed |
| states | OverlayState\* | An enumeration of possible states of the overlaytype |

#### OverlayState

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| \* | String | Unique name of the overlay state |
| image | String | Relative path to the image representing the state |

#### OverlayItem

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| \* | String | Unique name of the overlay items being used for references |
| x | Double [0 … 1.0] | Horizontal position of the item as a proportion to the width of the machine |
| y | Double [0 … 1.0] | Vertical position of the item as a proportion to the height of the machine |
| width | Double [0 … 1.0] | Width of the item as a proportion |
| height | Double [0 … 1.0] | Height of the item as a proportion |
| type | String | Name of the type of the overlay item as defined above |

#### TransitionSequence

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| \* | String? | Unique name of the transition set |
| steps | TransitionSequenceStep\* | The steps that will be performed within the given transition |

#### TransitionSequenceStep

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| \* | String | Unique name of the overlay type |
| Transition | String | Name of the TransitionSet to be performed |
| Duration | Double | Duration of the step as it is shown to the user |

#### TransitionSet

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| \* | String | Unique name of the overlay type |

The value of the transition set is a dictionary mapping the item to be changed to its corresponding final state as given in the following example:

"ToYellow":

{

"Red": "off",

"Yellow": "on",

"Green": "off"

},

### Example

The following script is an example for a traffic light definition.

{

"name": "Traffic Light",

"version": "1.0",

"types":

{

"RedLight":

{

"states":

{

"on":{

"image": "redlight\_on.png"

},

"off":{

"image": "redlight\_off.png"

}

}

},

"YellowLight":

{

"states":

{

"on":{

"image": "yellowlight\_on.png"

},

"off":{

"image": "yellowlight\_off.png"

}

}

},

"GreenLight":

{

"states":

{

"on":{

"image": "greenlight\_on.png"

},

"off":{

"image": "greenlight\_off.png"

}

}

}

},

"items":

{

"Red":{

"x": 0,

"y": 0,

"width": 1,

"height": 0.33,

"type": "RedLight"

},

"Yellow":{

"x": 0,

"y": 0.33,

"width": 1,

"height": 0.33,

"type": "YellowLight"

},

"Green":{

"x": 0,

"y": 0.67,

"width": 1,

"height": 0.33,

"type": "GreenLight"

}

},

"transitions":

{

"ToRed":

{

"Red": "on",

"Yellow": "off",

"Green": "off"

},

"ToYellow":

{

"Red": "off",

"Yellow": "on",

"Green": "off"

},

"ToGreen":

{

"Red": "off",

"Yellow": "off",

"Green": "on"

}

},

"sequences":

{

"FromRedToGreen":

{

"steps":

{

"Begin":

{

"Transition": "ToRed",

"Duration": 1

},

"GoingToYellow":

{

"Transition": "ToYellow",

"Duration": 1

},

"NowGreen":

{

"Transition": "ToGreen",

"Duration": 1

}

}

}

}

}

# Milestones

## Milestone I

The necessary classes for the data elements are existing and transitions can be reversed and forwarded without any graphical interface. In addition, some debug routines exist which bring the current state of the machine to the console and/or as a user-readabe string

## Milestone II

The application itself has a WPF window and a hard-programmed example is shown where the user can change the state of the overlay items.

## Milestone III

The script language is defined and the user can define a machine, the overlayimages and their states by using the script. The application will be extended to be able to load a script. If possible, the application registers itself to a file extension.

## Milestone IIIa

The script interface can define a transition sequence and the application supports the execution of a transition sequence by clicking forward or backwards. In addition, an automatic execution is supported including the definition of time between two steps.