Team Amadeus:
MAD Assembly Builder
Design Review 3

Members:
Wyatt Evans, Kyle Krueger,
Melody Pressley, Evan Russell

Mentor:
Austin Sanders

Sponsors:
Dr. Hélène Coullon & Dr. Frédéric Loulergue

Team Introductions

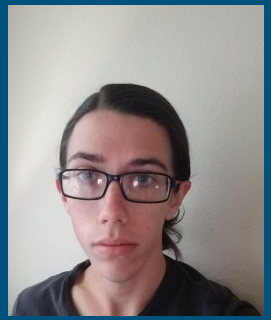
Wyatt Evans

Team Leader

Kyle Krueger

Release Manager

Melody Pressley



Document Architect

Evan Russell

Documenter

Software Deployment

- Deployment of software across multiple devices
- Many interrelated, interconnected activities
- All software is unique
  - Different dependencies
  - Different characteristics
  - Different specifications
  - Deployment process must be unique

Fig. 1: Software Deployment Example

Our Clients

Dr. Frédéric Loulergue

Professor at School of Informatics Computing and Cyber Systems

Dr. Hélène Coullon

Assistant Professor at IMT Atlantique, Inria researcher

Madeus / MAD

- Madeus
  - Theoretical Model for Software Deployment
  - Explicitly Defined Steps and Dependencies
- MAD
  - Madeus Application Deployer
  - Formal Implementation
  - Python

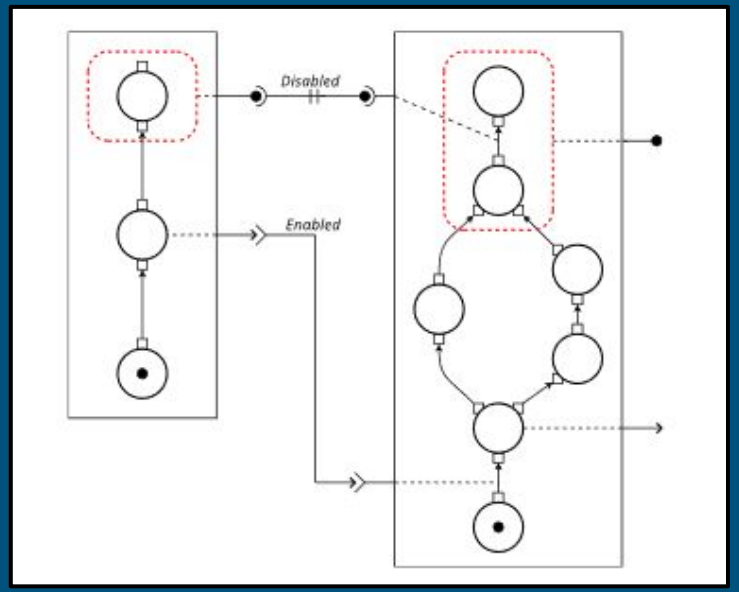
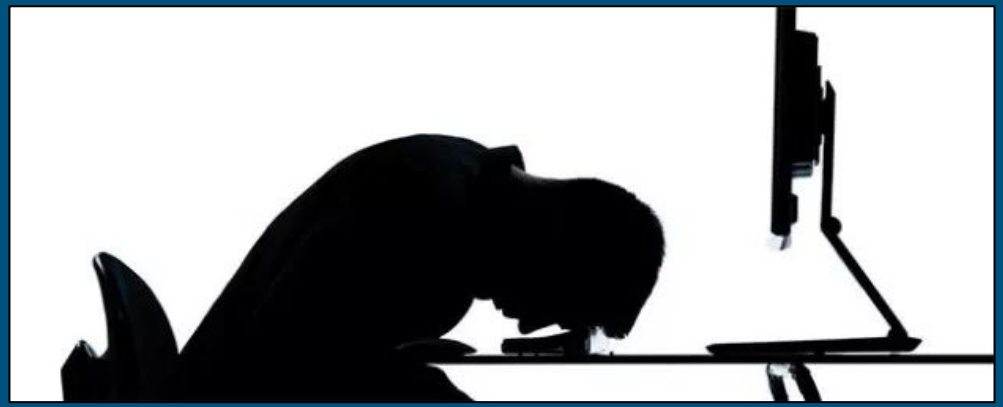


Fig. 2: Basic Madeus Assembly

The Problem

- Current process is slow
- Designing an assembly in code is tedious
- Complex to edit
- Easier to visualize and modify with diagrams



Our Solution: Develop a GUI

- Visualization
- Simulation
- Easier for users to edit
- Decrease turnaround time on MAD Assembly development

Key Requirements

- Visualize the creation of Madeus assemblies
- Extensible framework that allows for future additions
- Generate MAD code that represents the user’s diagram
- Simulate deployment of an assembly

Architecture Overview

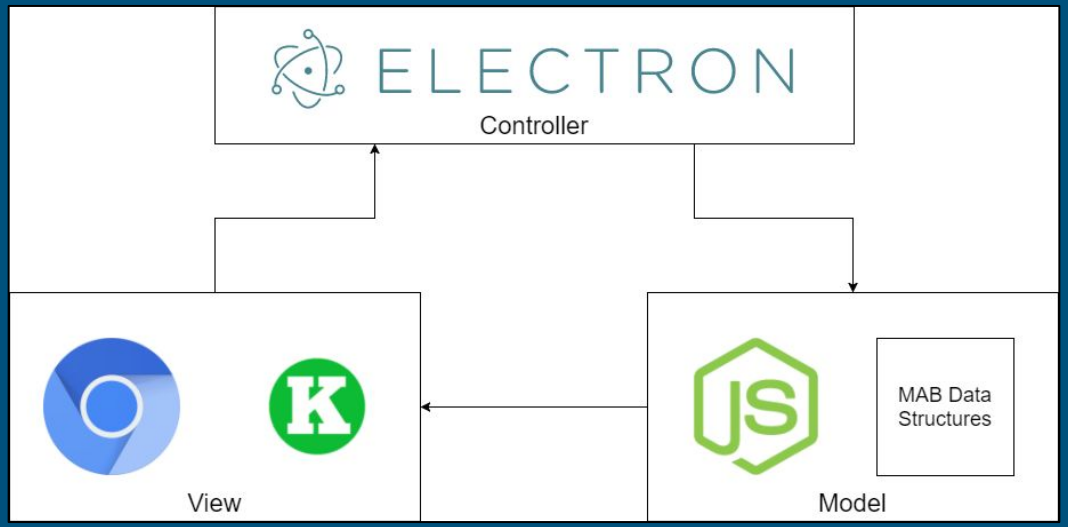


Fig. 3: MVC Architecture

Architecture Overview: Controller

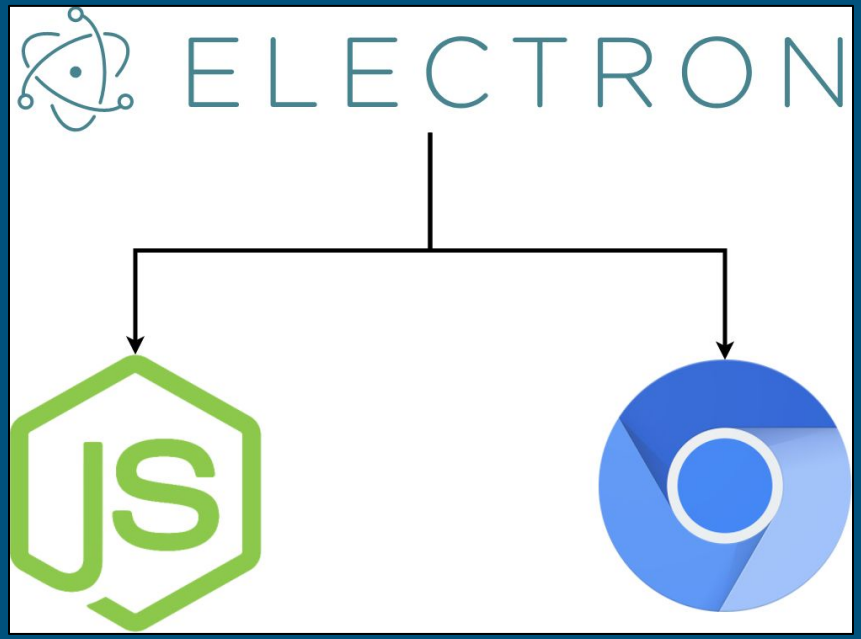


Fig. 4: MVC Architecture - Controller

Architecture Overview: View

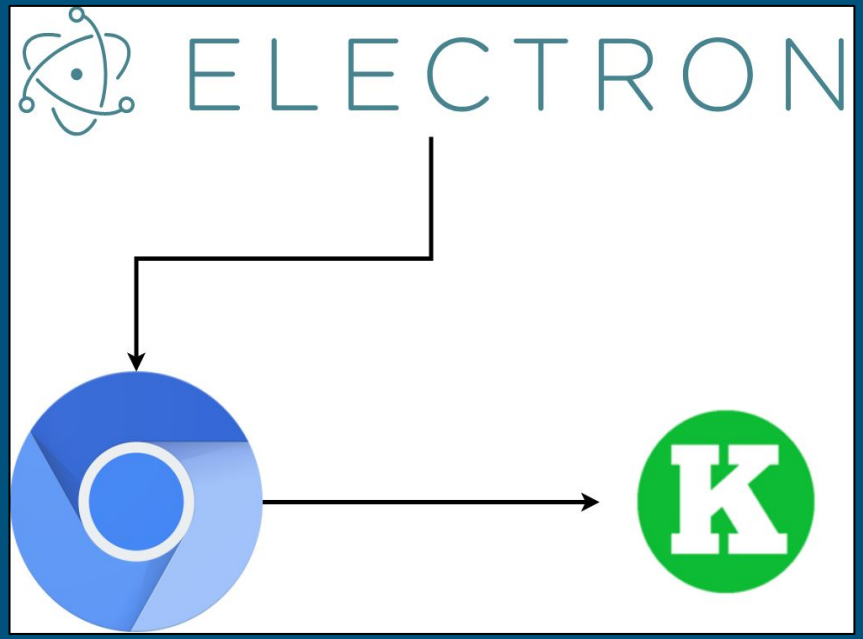


Fig. 5: MVC Architecture - View

Architecture Overview: Model

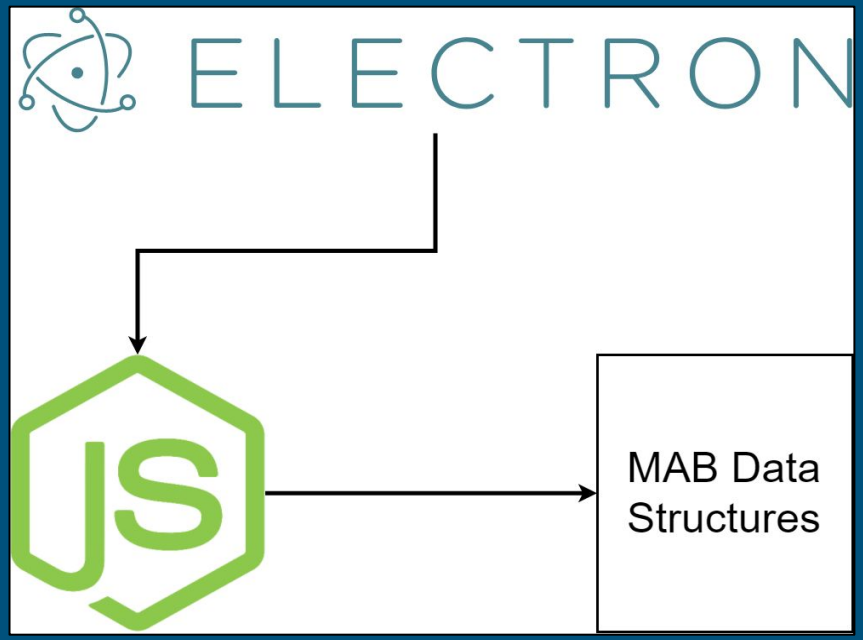


Fig. 6: MVC Architecture - Model

Implementation Overview

- [Model] Data Structures
  - Assembly Component List
    - Contains all user created components in one centralized location for [Controller] use as well as any provided plugins.
  - Connection List
    - Contains all dependency connections between components.

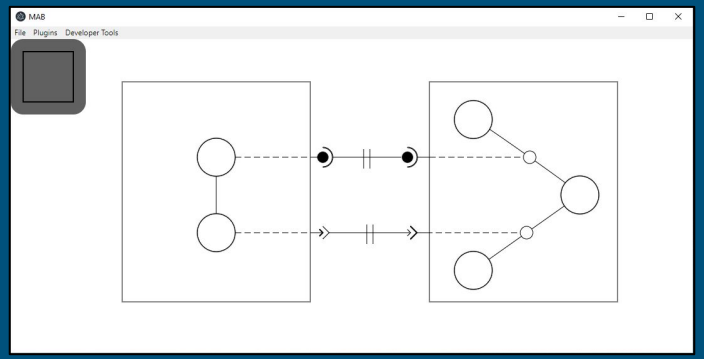


Fig. 7: Complete Assembly Example

Prototype Demo

- (1) - Component Creation

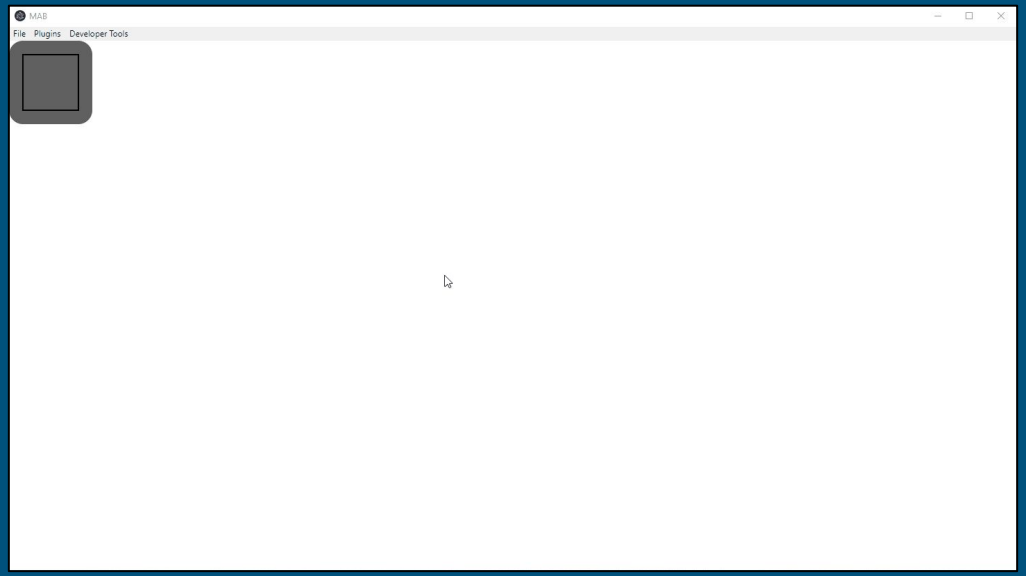


Fig. 8: Component Creation

Prototype Demo Cont.

- (2) - Place Creation

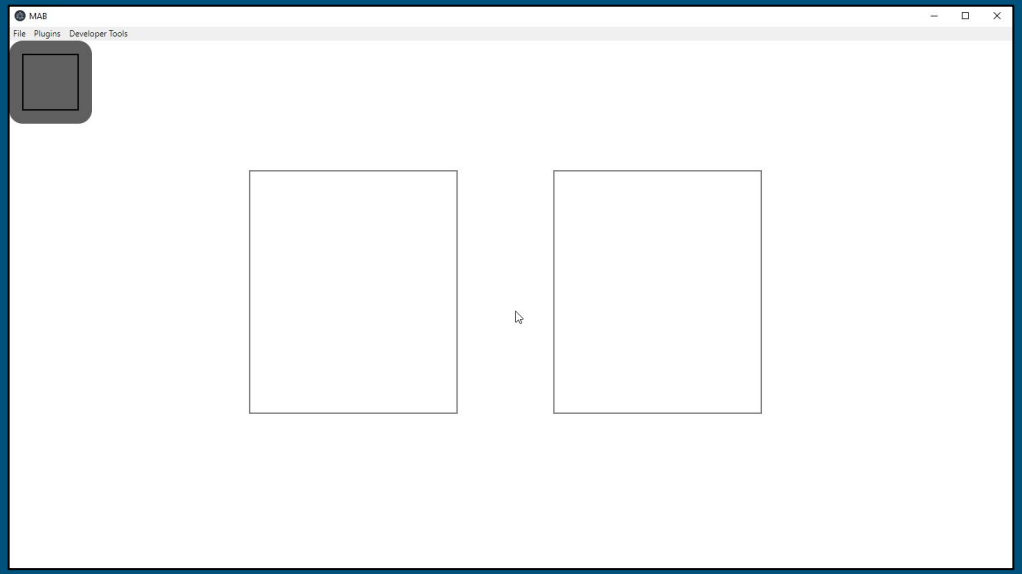


Fig. 9: Place Creation

Prototype Demo Cont.

- (3) - Transition Creation

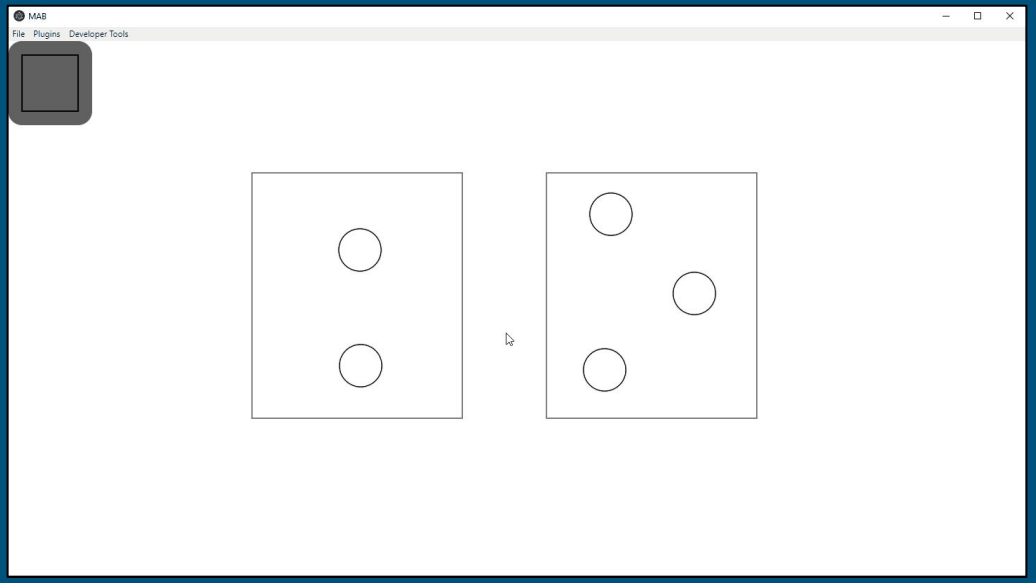


Fig. 10: Transition Creation

Prototype Demo Cont.

- (4) - Dependency Creation

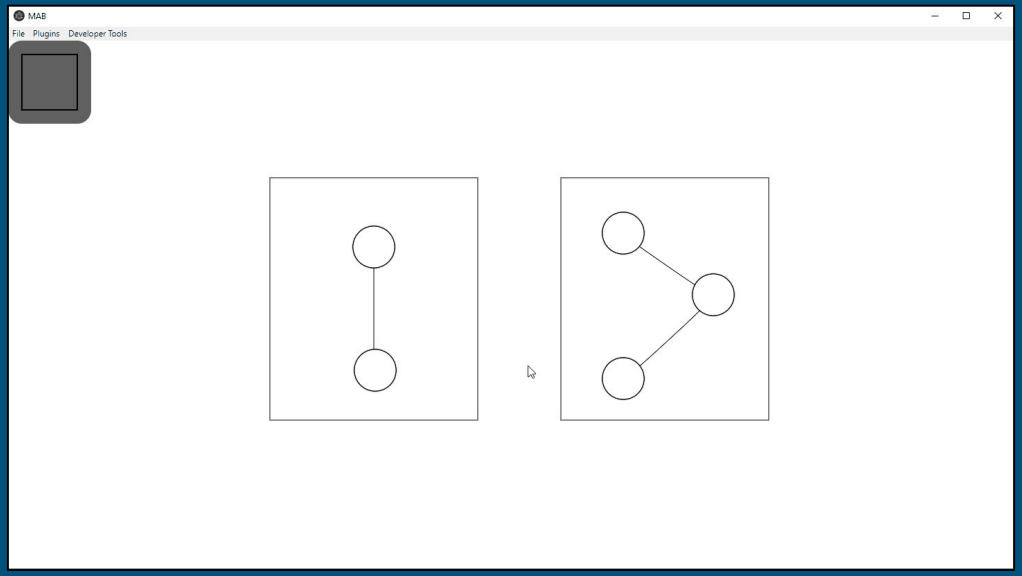


Fig. 11: Dependency Creation

Prototype Demo Cont.

- (5) - Connection Creation

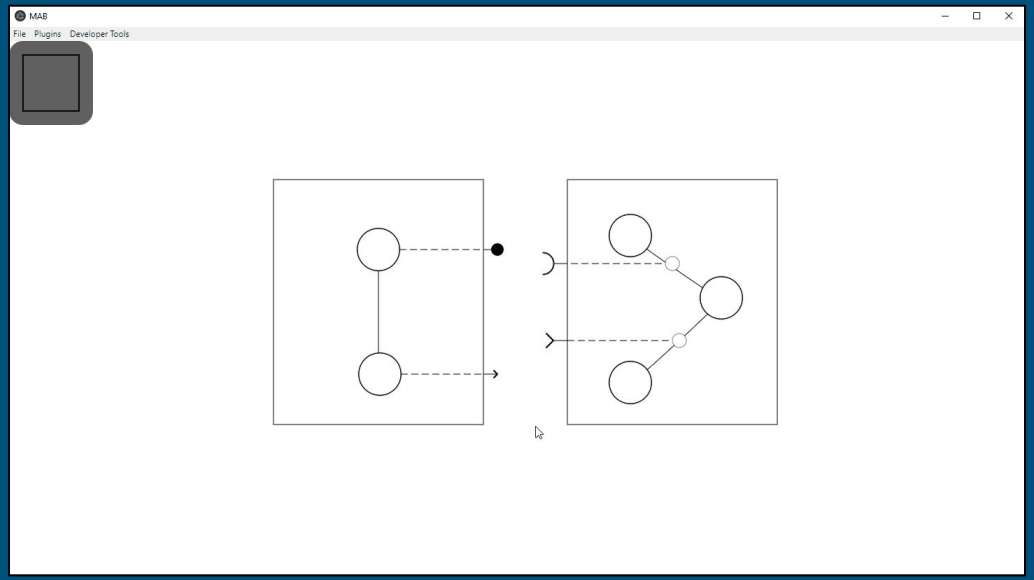


Fig. 12: Connection Creation

Prototype Demo Cont.

- Basic Assembly Manipulation

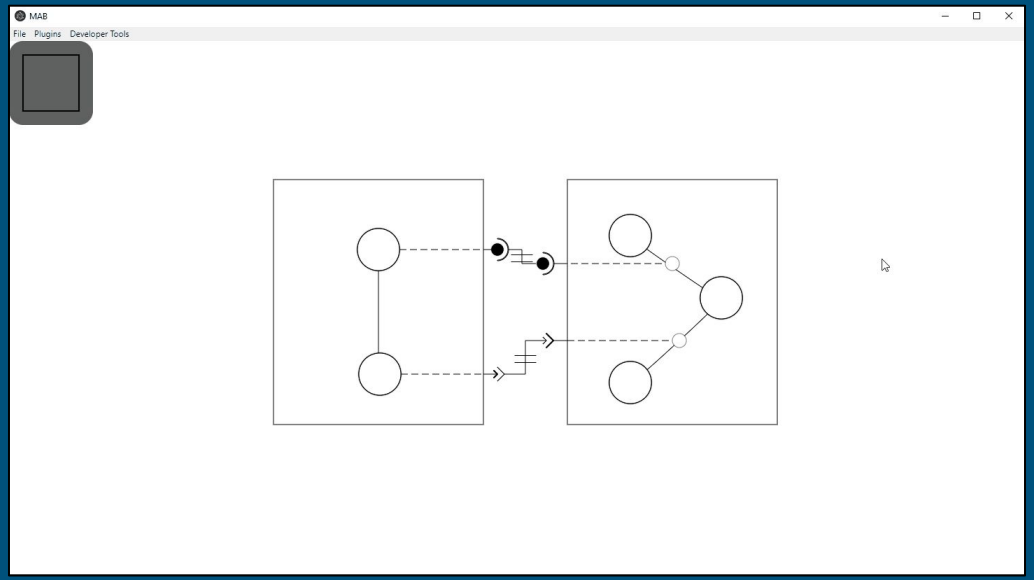


Fig. 13: Assembly Manipulation

Prototype Demo Cont.

- Assembly Simulation

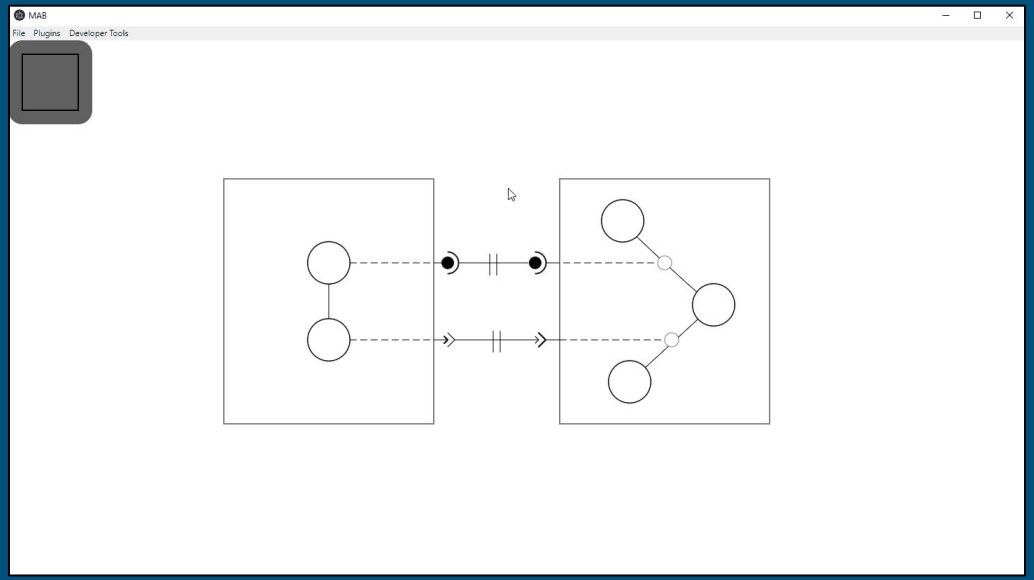


Fig. 14: Assembly Manipulation

Prototype Demo Cont.

- Code Generation

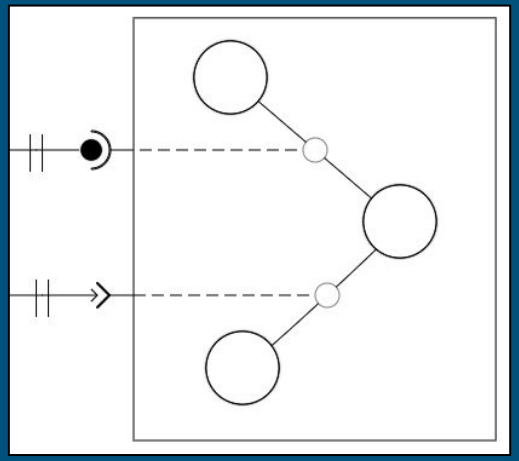


Fig. 15: Component_2

```
from mad import *
import time

class Component_2(Component):
    def create(self):
        self.places = [
            'Place_1',
            'Place_2',
            'Place_3'
        ]

        self.transitions = {
            'Transition_1': ('Place_1', 'Place_2', self.defaultFunction_1),
            'Transition_2': ('Place_2', 'Place_3', self.defaultFunction_2)
        }

        self.dependencies = {
            'Dependency_1': (DepType.DATA_USE, ['Transition_1']),
            'Dependency_2': (DepType.USE, ['Transition_2'])
        }

    def defaultFunction_1(self):
        time.sleep(8)

    def defaultFunction_2(self):
        time.sleep(2)
```

Fig. 16: Component_2 Generated Code

Challenges and Resolutions

- Limitations with Kivy Python framework
  - Switching over to Electron (Node.js and Chromium)
  - Electron framework behind Atom, Visual Studio Code, Slack, and Discord
- Saving and Loading of User Created Assemblies
  - Amended our Data-structure to serialize and store the Konva objects/groups
  - Saving will capture all objects and their attributes (size, position)
  - Loading will build an assembly from the serialized data-structure
  - User created assembly and data-structure generated assembly
- Deployment Simulation through GSAP
  - Simulation mode
  - GSAP or Greensock Animation Platform

Schedule

Gantt Chart / Development Schedule

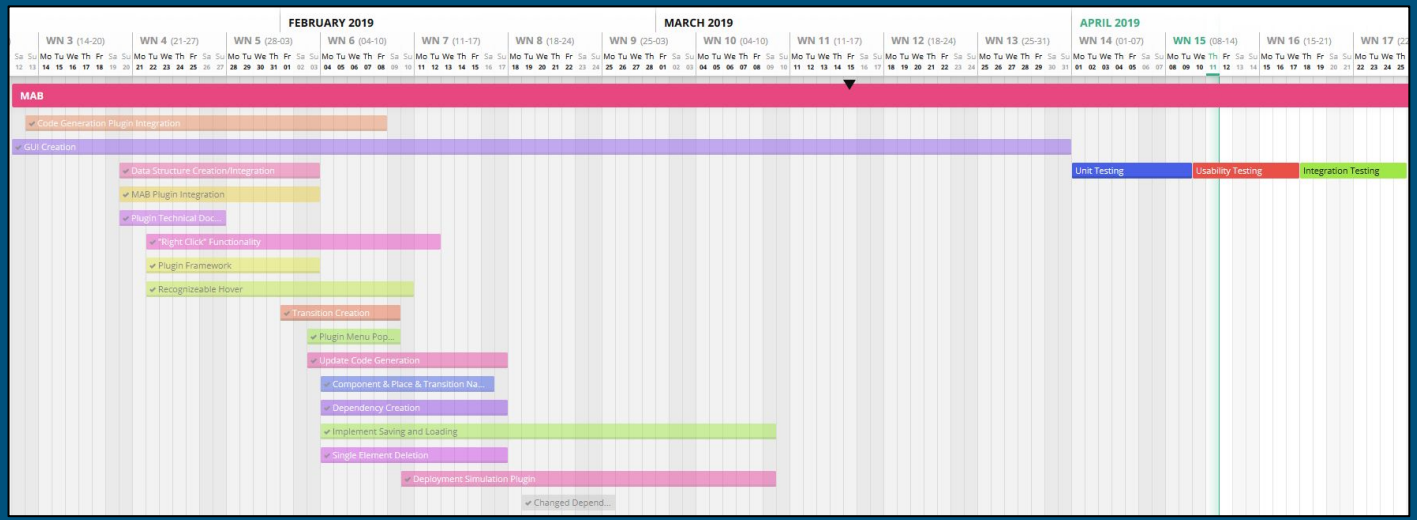


Fig. 17: Gantt Chart

Testing Plan

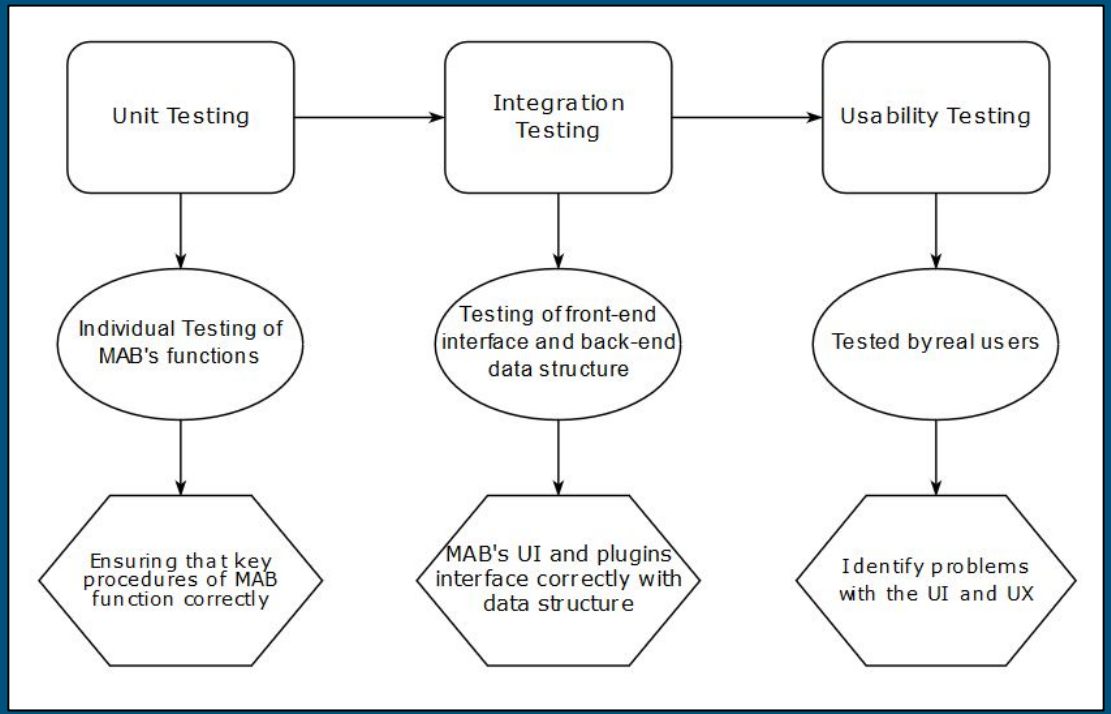


Fig. 18: Testing Flowchart

Conclusion

- The Problem
  - MAD software results in good deployment performance but is tedious and complicated to implement
  - Need a way to help visualize software deployments
- Our Solution Vision
  - Develop a Graphical User Interface
    i. Help Visualize a Madeus Assembly
    ii. Accurately Simulate Software Deployment via animation
    iii. Automate the Generation of Madeus Application Deployer Code
    iv. Allow for Saving and Loading of a user created Assembly
- Our Plan
  - Testing Phase

Thank you!