vMOF: Virtual Meeting Outcome Facilitator

Team Arbiter

Steve Brewster       Chris Farrell       Andrew Landman
Evan O’Malley

Mitre

Doug Phair, Marc Cannava, Lorin Petersen

Faculty Coach

Ken Martinez

Project Overview

The Virtual Meeting Outcome Facilitator (vMOF) is a web application intended to address common problems encountered during meetings. These problems arise for a number of reasons but primarily because the meeting facilitator lacks the correct experience to enforce a productive meeting. The vMOF system attempts to lighten the burden of meeting facilitation by providing interactive activities that allow the participants to engage in ideation, conflict resolution, decision making, and ice breaking. By shifting the burden from an inexperienced meeting facilitator to a web application, meetings can produce concrete outcomes through their ideation and decision making processes and stay on track using rigid timers to ensure that core activities are completed on time. The final delivery of vMOF contains an icebreaker activity called Two Truths and a Lie which helps everyone get to know each other a little better and get comfortable. Anonymous Brainstorming allows everyone to engage in ideation about a given topic without feeling that they will be judged. Empathy Map is an ideation activity that helps brainstorm about the customer of a product and hopefully learn ways to improve a product or service. Polling and Tiebreaker are final outcome activities that help settle disputes over which option is better, Polling allows for anonymous voting, and the Tiebreaker selects a winner at random. Together these activities provide a general set of activities that are common to most meetings and provide an interactive way to achieve results.

Basic Requirements
vMOF is the virtual meeting outcome facilitator which is designed to run as a web application in two separate views. The facilitator view is to be running on a main screen that is in the physical meeting space. Each attendee view is run on the participates device of choice. Each facilitator and attendees are grouped into separate rooms to have multiple meetings running concurrently. In the application the system must have engaging activities that bring meaning to the meeting and a way to automatically record what has happened in each of the activities.

Non-Functional

- The system must support mobile and desktop users using responsive design to provide a similar experience on all screen sizes
- The interface must be intuitive and easy to use so that the focus can be on meeting outcomes rather than learning new software
- The system must be deployable as a web application
- Activities in the system must be engaging to the participants
- An icebreaker
  - Two truths and a lie, which is set up as a competition to fool the most people and guess the most amount lies.
- Brainstorming
  - Anonymous brainstorming is about solving the problem of the general in the room shooting down people's ideas. This activity will set up a safe environment to have people submit their ideas without the fear of the general shooting it down. During the process attendees should be able to anonymously vote on ideas, comment then assign action items based on what was discussed.
- Empathy Map
  - There needs to be an activity that is all about getting into the customers shoes to understand them. Attendees will then be able to view and rate each post-it note.
- Polling
  - An activity designed to take all of the ideas presented in a room and allow the participants to narrow it down to whichever one they choose.
  - Will notify users of the idea selected or if there is a tie.
- Tiebreaker
  - vMOF should act as an automatic conflict resolver by deciding for the attendees between two or more choices.

Functional

- System must send meeting results in the form of an email
  - The emails will be sent automatically to the participants who signed in with an email address.
  - Results will be formatted and sent using vMOF’s gmail account.
- System shall support running in multiple rooms, each meeting shall act independently from each other
- Application must run on IOS devices
- Application must run on the latest version of internet explorer
- Empathy map should look like a physical map, complete with digital color coded post-its
- Tutorial videos
  - Facilitators and attendees should have access to a tutorial video about each activity supported by vMOF
- Late users should not interrupt activities in progress
  - If an attendee joins late they shall be at the splash screen until the start of the next activity.
- Users should not be able to participate in activities if the facilitator has been disconnected
  - Upon a facilitator leaving the room the current activity will end and all attendees will be sent back to the splash screen.

**Constraints**

From the first semester we were to have LYNC integration with our applications. Users could participate in the meeting activities through the use of a command line like interaction through the LYNC client. Because of this we were tied to the Microsoft .NET and C# framework for the first semester of this project. This sprouted other problems such as each team member did not have a dedicated windows device to do development on and instead of development meetings done in person they were done online. Another issue was since this is a real time web application we had to find the right library for communication and we ended up using the SignalR library that was compatible with the .NET stack. SignalR did not natively support grouping the sockets into rooms, this functionality had to be added in by the team. The biggest issue we ran into during development was the team did not have much experience making web application. Most the development time went into learning how to use JS, JQUERY, and the .NET stack. At the time of the interim presentation we had struggled with the current stack to produce nothing more than a concept demo.

One of the last items we agreed upon was that the API for the LYNC integration was not yet completed by Microsoft halfway into the project life cycle. After research was done by the team and evidence presented to MITRE we official had that requirement dropped. This gave us a chance to revisit the tech stack make more informed choices on what we should build our app with. Over the break we looked into different technologies to see what would best fit the problem we were given. In order to make sure that we only had to write in one language we decided to go with a nodejs, express, and react framework. This solved the problem of different development environments. Our server side code and our client side code is all written in JS which we learned on the first semester. With react for the front end we split up the UI into reusable components to build activities quicker with reusable components. To improve the communication between the facilitator and the clients Socket.io was chosen over other socket frameworks. Socket.io has native support for socket grouping by rooms, which is exactly one of the requirements for vMOF.

Because of the tech switch in the middle of the semester we became crunched for time with our requirements. We did have time to get to data persistence with the application which would
have held meeting information on the server to view previous meetings as well as persist users through multiple log ins.

**Development Process**

MITRE requested that a spiral process be used since this is most commonly used by their software teams. Early in the project we had numerous risks that needed to be addressed so we used a risk driven methodology to address our biggest risks in the first two cycles. After the winter break we changed tech stacks and reduced the scope of the project significantly. During the second term our risks were effectively addressed allowing us to move into an evolutionary methodology to focus on rapid prototyping and frequent customer feedback.

Communication was addressed early in the project by the sponsors. We were given access to their Handshake.com portal and we were able to handle communications through the tool. When delivering our initial requirements, mockups and questions to the sponsors, this tool proved useful in allowing for asynchronous communication between our regularly scheduled conference calls.

Our primary roles were Project Manager (Stephen Brewster), Testing Lead (Chris Farrell), Research and Build Engineer (Andrew Landman), Gatekeeper (Evan O‘Malley).

**Project Schedule: Planned and Actual**

Our project was more of an evolutionary prototype with no concrete milestones in place that necessitated a long term schedule. We used a schedule template for each three week cycle where the first week focused on mockups, functionality was written in the second week, and testing and cleanup in the third week. This was more of a guideline than a hard set rule since there were gray areas between the necessary activities in each cycle. Ultimately our goal was to spend as much time on the prototype as possible and continuously improve the product after each round of feedback. Our sponsors reinforced this methodology because it is what they are familiar with as a research company. Their goal is to just keep iterating and improving the product until they have something they’re happy with or time runs out. It became unnecessary to maintain a concrete schedule when our activities boiled down to a basic iterative cycle of implement, demo, then respond to feedback.

**System Design**

At the beginning of the project we started out using ASP.NET MVC. The rationale behind this decision was that we had the requirement to incorporate Microsoft Lync chat integration into the project. We thought Microsoft would have an API that we could use to make it easier to implement this requirement. However, it turned out that Microsoft was still in the process of developing a Lync API that would work with the Office 365 server that we were given by MITRE. In addition, we decided to use a library called SignalR to handle the real-time web socket communication between our server and client. After much research we decided that SignalR was the most mature real-time library available for a Microsoft tech stack. In addition, the library was made by Microsoft themselves.
After trying out a few different Lync APIs, we realized that none existed that worked with a Microsoft Office 365 server. When we conveyed this new found information to MITRE they decided to remove the Lync requirement. Additionally, at this time the team was encountering several issues with the way SignalR worked, including getting users into separate rooms and handling separate rooms concurrently. Furthermore, the team was having trouble building the system we wanted to on top of the scaffolding that the ASP.NET MVC project generated. The framework was more geared towards standard RESTful web applications whereas we wanted to create a multiroom real-time web application.

At this point we decided that a Microsoft tech stack may not be the best choice for us. Over the winter break we took a look at other web technologies that could be applicable to our project. After coming across Node.js and socket.io, and playing around with various github projects that showcased what could be done with these technologies in real-time web applications, we decided to continue down the Node path. Additionally, the team came across the React framework which would allow components to re-render themselves when the state of the component changed. We decided that this behavior would be immensely helpful in our real-time application, so we decided to stick with React.

In order to meet another one of our requirements of having both facilitator and attendee roles in our system, the team decided to create two separate single page applications for each of these roles. We needed some way of bundling our modules together for both of the these single page applications. After looking at tools such as browserify, requirejs, and webpack, the team decided to stick with webpack because it allowed us to generate separate bundles for both of our roles very easily. In addition we decided to use the Express web framework so that we could easily statically request these bundles from our server. It was also very lightweight and didn’t impose any sort of architecture on our system the way that ASP.NET MVC did. Additionally, it gave us much easier control over URL routing than ASP.NET MVC.

At this point we had the technologies that would enable us to build a system that would meet our requirements. We began laying the foundation of our architecture. On the server we created controllers that would handle each of the five separate activities that we would be focusing on. This included the Brainstorming, Two Truths and a Lie, Empathy Map, Polling, and Tiebreaker. Additionally, we create a controller to handle actions associated with rooms such as a user being able to join and leave a room. These controllers would delegate to different activities models we created which all extended an Activity module. Furthermore, we decide create a global collection of rooms wrapped in a Rooms model that would enable us to get the specific room that the user sending the request was in. Each Room object contained references to the users in that room in addition to the activity itself. Our rationale for having this Rooms model was that it would allow us to easily access a specific room and all the information associated in it, from each of the five activities. Additionally, the way socket.io and nodejs worked was that each client request was an atomic operation meaning that there were no concurrency issues that could present themselves from having multiple clients make requests to our server at the same time. One important note about our server side code is that it contained logic for the different activities. Since each activity had vastly different logic, there wasn’t that much reuse of code between each of these activities.
On the client side we built different React components primarily for the benefit of being able to re-render them easily when they changed state. Many of the react components that needed to be rerender contained socket.io listeners that would listen for a message from the server. Upon receipt of a message we would set the state of the component with the new state from the message, causing the component to rerender. This was highly beneficial for us as it allowed us to avoid a lot of DOM manipulation code that could have gotten very messy with the amount of dynamic data that we needed to present to the user. Another benefit of the React components was that some of them could be reused between both the facilitator and attendee roles.

Below is a high level diagram of our final architecture that we have described above:

![High level diagram of the final architecture](image)

### Process and Product Metrics

#### Product:

We calculated an estimate of the number of lines of code that we wrote:

```
$ find ./src -name "*" | xargs wc -l
11724 total
```

This number seems reasonable given that we had a lot of custom unique code for each of the different activities in our system. Perhaps the number may be a little high which could indicate that we could have focused more on code reuse more than we did. Reducing lines of code could have the potential to make our application less error prone.
We calculated the mean per-function cyclomatic complexity using complexity-report for Node.js, which turned out to be: 1.1666562211182232

Since the mean per-function cyclomatic complexity is low, our functions are likely easy to understand.

We calculated the mean per-function logical LOC using complexity-report for Node.js which turned out to be: 3.3190790331284026

This number is small which is an indication that our functions aren't that big and are easy to understand and maintain.

**Process:**

**Product State at Time of Delivery**

The delivered product had all the activities required by the sponsor and without any negative feedback from them we assume that they are of acceptable quality. Emailing of the results for the major input activities; Empathy Map, Brainstorming, and Polling, was also completed. There was a reach goal offered by the sponsors to persist data for the users that login and track their activity which was not achieved. In addition to the sponsor required activities we incorporated some features such as room titles and splash screens which we thought would enhance the product. There were no features that were promised and not delivered in the final release aside from the reach goal. Some small features that would have improved the usability experience did not make it in, but they were suggestions and not requirements.

**Errors uncovered before release of the software**

Before the final release just after the presentation we hit all of our intended functionality a couple weeks prior. In the weeks up the presentation we conducted functionality testing within the group. During each round of testing the issues and errors were recorded in a google doc. The total number of issues found was 52 issues according to the outstanding issues document. We were able to address 45 of these issues. Given the amount of time, the team did a good job identifying and hammering out the major issues. Each item was given a priority and all the high priority items were hit.

**Time Tracker**

Andrew Landman: 393 hours
Chris Farrell: 192 hours
Evan O'Malley: 285 hours
Steve Brewster: 265 hours

**Project Reflection**
What Went Right

With the tech stack change a lot of our problems went away. The learning curve for Node was a lot smaller than it was for .Net. This lower learning curve allowed for us to churn out better prototypes quicker than we were able to with .Net. In addition to the lower learning curve, we were also able to generate easier UI mock-ups with Node since it was HTML. By generating mock-ups this way we were able to give the customer a more accurate idea of what the final product would look like and any changes that we made to them would be accurately represented.

During the switch from .Net to Node we also switched the communication technology that we were using. When we switched from SignalR to socket.io a lot of the problems that we were having with the application crashing or data going missing were alleviated. Due to a difficult to understand documentation SignalR was a big problem for the team in the first semester. After switching to socket.io the problem with messages being lost/not sent and timers not setting properly. This change in tech allowed us a lot more freedom with creating our application as the documentation was easier to understand and clearer in how to do what we wanted to.

In the final cycles of the project we were able to solidify the vision for the project and worked well as a team to implement the various activities and features to the sponsor’s satisfaction. Once we got into the rhythm of the cycles and became familiar with the technologies we were able to quickly produce the final set of features and fix the outstanding issues from the customer feedback.

What Went Wrong

We had a great deal of difficulty in the first term at all stages of the project. Our initial approach to eliciting the requirements for the product took much longer than they should have and delayed the start of our implementation by a couple weeks. Essentially we were trying to extract solid requirements from the sponsors but they were only willing to give rough guidelines for what they wanted the project to do. We should have seen what they were driving at sooner and sought a rough outline of required features rather than seeking explicit requirements for the entire project.

There were some communication errors when delivering the initial user stories and mockups because the methods that are taught at RIT differ from the expectations of the MITRE team. We should have been a little more precise with our questions to determine exactly what they were looking for from these artifacts.

Process was difficult for us to maintain in a disciplined and consistent manner. This was in part due to the project itself and team preferences. We were able to track our time and report progress and blockers each week, as well as stick to a basic schedule for the cycles. However we did not have the documentation and structured team meetings necessary to engage in other activities that would have improved the project. It would have been useful for us to engage in
more paired programming, group design, and code reviews to ensure that the code base was of acceptable quality and to help all team members get on the same page.

Our initial requirement for Lync integration into the project set the project back by at least four weeks. Many man hours going into researching its feasibility and setting up a tech stack that paired well with Lync. Once this requirement was removed we were compelled to change to a set of technologies that worked better for our purposes. We should have tackled the Lync research as a team to eliminate it as a possibility prior to choosing a tech stack and beginning development.

The overall team dynamic wasn’t as cohesive as we would have liked. We got along well and had fun along the way, but we lacked a strong unified vision of the expectations and vision for the project. It would have been beneficial to engage in more team building activities through games or going out for drinks.