# OpenMDM Client Technologies Overview

# Table of Contents

1. Technological Approach ..... 2
   1.1. Full Web Stack ..... 2
   1.2. Full Desktop Stack ..... 2
   1.3. Web Stack with Device Helpers ..... 2
   1.4. Shared Web and Desktop Stack ..... 2
   1.5. Separate Web and Desktop Stack ..... 3
2. Client Features ..... 4
   2.1. Security ..... 4
   2.2. Ubiquity ..... 4
   2.3. Deployment ..... 4
   2.4. Updates ..... 4
   2.5. Device Communication ..... 5
3. User Feedback ..... 6
   3.1. From BMW ..... 6
4. Evaluation ..... 7

This document presents an overview of the different technological stacks that can be used to implement a client for the OpenMDM technology. Alongside it describes some key aspects for each client stack that should be used to decide which client stack should be used for implementing OpenMDM applications. Finally, it incorporates a summary of our findings after conducting user interviews.

# Chapter 1. Technological Approach

Given recent feedback from users of the current system we have determined that the original technology stack set forth for both Web and Desktop clients may not be the best one, specially looking forward to a platform that should last for the next 5 years at the very least. Usage patterns of the different type of users involved with the current system will help us decide which of the following technology options is best suited to continue.

## 1.1. Full Web Stack

Regardless of the implementing technology (Eclipse RAP, JavaEE + JSF, Grails, etc) a web application delivers the comfort of easy deployments paired with quick updates whenever they are needed. Mobile clients can also benefit from this mode, as long as they stay connected. The usage of a full web stack option is less appealing where direct contact with specialised devices (such as the benchmark measurement machines) is mandatory. Thus, this stack makes total sense for test engineers and test managers but not so for test executives that could leverage automatic import/export of data from measurements machines when possible.

## 1.2. Full Desktop Stack

Desktop clients can exploit direct device communication without jumping through hoops like web clients do (most likely through a Java applet which brings in a host of different problems). A desktop client based on Eclipse 4 platform will do the expected job. Or perhaps a non OSGi technology such as a full JavaFX, non-eclipse solution. The important key to remember here is that deployment will not be as smooth as in the web option, and that we also loose the option to deploy to mobile clients, such as tablets.

## 1.3. Web Stack with Device Helpers

A hybrid approach that can keep both test engineers and test executives happy could be a mix of web and desktop client technologies. The bulk of the application is implemented using a Web stack (this is the part where test engineers will use most of the time); the operations that require direct device communication (the ones required by a test executive) can be implemented using small desktop clients, or command/daemon based applications, that use web-friendly options (such as exporting data in JSON format, running a local web server, etc) in order for the web application to read the data coming from the devices. Deployment and rolling updates for said command based applications still represent an obstacle that must be overcome; the same deploy strategy as a full desktop client stack can be applied here.

## 1.4. Shared Web and Desktop Stack

This is an option unique to Eclipse RCP, as the same application can be accessed from the web using

Eclipse's RAP technology. However the fact that a web based application cannot access local devices while the RCP one can means that RCP applications must be deployed to benchmark stations while web based applications can be deployed everywhere else. Deployment of RCP applications remains as a hurdle to be overcome. Frankly there are not many advantages for this option as if you have to come with a solution for the deployment problem for a subset of users you can apply it for all of them.

## 1.5. Separate Web and Desktop Stack

Finally we come to the last option where Web and Client might share some common codebase but they are inherently different from one another. They might implement the same functionality but they do so in totally different ways. In a sense it's writing the same application twice, using two non-homogeneous technology stacks. This gives the best flexibility overall in terms of richness and behavior required by each user however it's perhaps the one that requires more thought about its architecture, as it will have much more moving parts that then other options.

# Chapter 2. Client Features

The following features define the behavior and essence of each client stack. We believe them to be the main drivers for picking a technological stack.

## 2.1. Security

They are inherently insecure given that they rely on an execution environment (the browser) that can't be fully controlled. If needed, opening access to the filesystem or short time persistence storage can cause further security risks, at the very least they require an additional step during deployment and configuration. There may be some environment (such as mobile) where direct access to the hosting environment is not possible.

Security is stronger as the running environment can be closed down to every possible outcome (using a very strict SecurityManager for example).

## 2.2. Ubiquity

These applications can be accessed virtually from anywhere. Given responsive design techniques and good styling practices these applications can appear and behave like their native counterparts when accessed from a mobile device.

These applications can only run on the computer on which they are installed. This also ties the user to explicit location unless the application is installed on a laptop computer that can be freely moved to anywhere the user needs to be.

## 2.3. Deployment

Deploying a web application is a transparent step as users only need to be concerned by accessing a given URL on their browsers. Mobile applications can use their native application store (iTunes, Play, etc), meaning that installing a new application is basically one click away from the end user's perspective. Developers can use a combination of scheduled releases, rolling updates, even continues delivery to push new application versions to a web server. Mobile applications still require an authorization process in order to push new releases to the authorized application stores.

The first deployment can be performed either manually or using automated solutions as file copying or application provisioning tools. This is perhaps the topic that hinders most this type of applications as IT must be involved directly in order to assess how and when applications can be installed into a particular machine.

## 2.4. Updates

Like deployment, updates can be delivered transparently or with a single click (in the case of

application stores) to the end user. A point to consider here is that some users may need/want t access an older version (if possible), in which case versioning the communicating APIs is a good practice.

Rolling new updates is the second pain point that these application suffer. Either the same deployment techniques are used (updating the application as a whole, like a black box), or a customized update solution is built and put in place (for updating just the changed deltas). Such customized solution may be provided by the chosen technology stack (Eclipse auto-update feature) or built from scratch.

## 2.5. Device Communication

Devices can be separated in two camps depending on their communication strategy with the external world. - passive: they output data to a know location using files or sockets. They may also expose a REST interface or a webservice. - active: they can push data to a registered endpoint (webservice, REST, other) when data is available.

Active devices should pose a smaller integration risk than passive devices, as the former define a strict API that consumers must follow.

By design, web applications can’t break out of the browser. Consuming data directly from an external passive device can be very tricky depending of the communication interface that the device exposes. Mobile application will face the same problem.

Desktop applications can talk with both types of devices without much problem as they run on hardware that’s physically linked to the device or reachable by other means without compromising security.

# Chapter 3. User Feedback

## 3.1. From BMW

The test engineers (Versuchsingenieure) at BMW work mostly at their desktop in their office. Test executives (Prüfstandstechniker) run tests at workbenches in industrial buildings. One interview partner stated that the test executives in his department do not use the OpenMDM application. Instead they use their own application to gather the specification of the tests as well as to forward the measurement results to the ODS system. Another interview partner told that some test executives use the OpenMDM application to search for the full test specification by entering the key information from the event generated for the testbench's Outlook calendar.

Currently the client is distributed as a zipped file by superiors and colleagues. In order to fulfill special business needs of certain users/ user groups the client has been extended by plugins. Thus many different versions are in use now among the departments. Some departments do not benefit of plugins because they do not know of their existence.

Explicitly assigned roles and rights assure that only authorized users can manipulate test specifications (prior to the test order) and add measurement results and other associated data.

One interview partner mentioned that the OpenMDM application has to easily connect to sophisticated analysis tools i.e. to be able to open external desktop applications while providing an ASAM ODS path to the data of interest. Simplified analyses should be possible from within the application (in order to get an overview of the testsuite, find the test of interest). Another interview partner uses the OpenMDM system just for revision-safe archiving.

### 3.1.1. From Others

Security is regarded as a core requirement. The OpenMDM application has to provide storage and transport encryption and has to support data backup and recovery. Central infrastructure services like databases, file shares, corporate directories and authorization/authentification services which are already in place must be easily integrable.

The logged-in user should only see test data he is authorized for. In order to get access to data from others the user has to file a permission request which is either granted or denied by a designated administrator.

# Chapter 4. Evaluation

The previous categories have been summarized in the following table along with a valuation of their impact given the proposed set of technological stacks.

A - indicates less (or negative) impact where as a + indicates a higher (or positive) impact.

| Category | Sub-Category | Full Web Stack | Full Desktop Stack | Web Stack with Device Helpers | Shared Web & Desktop Stack | Separate Web & Desktop Stack |
|---|---|---|---|---|---|---|
| **Security** | Can close down environment | - - - | + + + | - | + + | + + |
| | Requires local setup | + + + | - - - | - | - - | - - |
| **Ubiquity** | Runs on desktop | + + + | + + + | + + + | + + + | + + + |
| | Runs no mobile/tablet | + + + | - - - | - | + + | + + |
| **Deployment** | Requires local install | - - | + + | + | + | + |
| | Requires downtime | + + | - - | + + | + | + |
| **Updates** | Requires local install | - - | + + | + | + | + |
| | Requires downtime | + + | - - | + + | + | + |
| **Device Communication** | Direct | - - - | + + + | + + | + + | + + |
| | REST / Webservice | + | + + | + + | + + | + + |
| **Time to Market** | Development Speed | + + | + + | + | + | - - |
| **Evaluation** | | **13** | **17** | **14** | **16** | **15** |