Universidade do Minho
Escola de Engenharia
Departamento de Informática

Fábio André Araújo Gomes

Remote Management of Applications

Deployment of Applications and Configurations using a Rule system

October 2016

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Fábio André Araújo Gomes

Remote Management of Applications

Deployment of Applications and Configurations using a Rule system

Master Dissertation
Master Degree in Computer Science

Dissertation supervised by
Professor Victor Francisco Fonte, DI, UM
Marco Cunha, Creativesystems

October 2016

When you see a good move, look for a better one.

Emanuel Lasker

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Prof. Víctor Fonte of the Informatics Department at University of Minho. He was always available whenever I ran into a problem or had a question about my research or writing by steering me in the right direction whenever he thought I needed it.

I would also like to thank the experts at Creativesystems who were involved in the validation and development for this research project: my co-advisor and Project Leader Marco Cunha who was the one in charge of evaluating its applicability to the company's clients. Nuno Caleira, Project Manager, for the a constant support and for allowing me to join the company to create this dissertation. And to José Álvaro, my co-worker in the project and mentor. His expertise and advice during the planning and development phase were crucial for this project success and without his enthusiastic assistance and input, this dissertation would not end so well. Thank you all.

Finally, I must express my very profound gratitude to my parents and to my sister for providing me with unfailing support and continuous encouragement throughout my long years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

ABSTRACT

Users expect access to programs and business information anywhere in the simplest way possible using a device. With the diversification of devices, the standard is disappearing and we are going towards a more heterogeneous world of mobile devices. With this divergence increasing, it gets more difficult to update, support and control applications through all these new platforms. Therefore it is important to facilitate these tasks.

The solution to these problems lies on the *Mobile Device Management (MDM)* programs that can control what devices install and configure, providing remote tasks and access. This dissertation aims not to compete with the current products on the market, but to propose a different way to distribute content to the devices registered on the platform using a Rule system. This system will prioritize the newest rules by the device and its location characteristics. As so, providing a different way of grouping devices and distributing content to them.

RESUMO

Os utilizadores esperam acesso aos programas e informações corporativas em qualquer lugar da forma mais simples possível, utilizando um dispositivo. Com a diversificação de dispositivos, o *standard* está a desaparecer e estamos a ir em direção a um mundo mais heterogéneo de dispositivos móveis. Com esta crescente divergência, torna-se mais difícil de atualizar, dar suporte e controlar aplicações através de todas estas novas plataformas. É então importante que estas tarefas sejam facilitadas.

A solução para estes problemas reside nos programas de MDM que podem controlar o que os dispositivos instalam e configuraram, proporcionando acesso e tarefas remotas. Esta dissertação não pretende competir com os produtos existentes no mercado, mas para propor uma forma diferente de distribuir conteúdo para os dispositivos registados na plataforma através de um sistema de Regras. Este sistema vai priorizar as regras mais recentes por dispositivo e as características da sua localização. Proporcionando uma forma diferente de agrupar dispositivos e distribuição de conteúdo para eles.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1

INTRODUCTION

This thesis was proposed by *Creativesystems (CS)* to be part and incorporate the retail solutions on its clients. CS has in most of its customers hundreds of PDAs with versions of applications configured for a given feature such as to take care of store inventory or items receiving. The maintenance of these devices is done manually when there is a need to change any settings or carry out a software update. As there can be hundreds or thousands of devices, the time needed to do a manual task in each device is very high. Whether for updates or when providing support in case of failure, the normal procedure is to send some employee to the site (e.g.: Warehouse, Store or Distribution Center) of the specific device and do the work manually. There is also the initial process of all this, the devices must be installed/configured at the time of its first installation with all the required software to operate. In this case it is necessary that for each PDA someone is installing application by application individually resulting in a waste of precious time. So there are some drawbacks that can be improved.

These problems are common for companies that have to control a huge number of devices, like CS where this dissertation takes place. The company doesn't have a software or mechanism to remotely provide this kind of support to its clients. Software updates and configurations are common tasks and if there was a way to do them efficiently, it would be extremely helpful for both the client and CS.

This dissertation aims to research and develop a system capable of remotely manage applications and settings for each device, along with an error analysis platform that can proactively give information to the user that a problem is or has happened. These problems can be related to bad installations, configurations or hardware issues. The software must have a generic approach in order to be easily applied to all kinds of clients and its companies structures. This project will be applied to a client after its initial version is completed and used in future solutions of the company.

There are many software management platforms on the market and this dissertation won't try to compete with them, only suggest a new type of deployment mode, called Rule System, and devices' organization. The developed solution will be called MDM, as its characteristics are similar to other products that share this device administration features. The first step is to gather information about the processes and methodologies in solutions that are currently on the clients. Understand its limitations and find the important features that this project must have in order to be a valid approach. Then, a research on other MDM products has to be made and gather their important features and spot its flaws comparing with the requirements. This project will be focused on the Retail business, due to the CS market place, but it should not be strictly designed for this area only and be generic enough to be used on other environments.

1.1 MAIN CHALLENGES

As there will certainly be more than hundreds of devices registered, the system will have to handle that amount of traffic. This can be related to scalability. The diversification of mobile devices will lead to different *Operating Systems (OS's)* and architectures. The project will need to be agnostic to this kind of variations and proceed to the request handling as neutral as possible. As there will be applications for specific OS the devices can't receive updates related to other OS.

The File Versions and Configurations that each device has to install will be calculated based on the Rule System. As these rules can be assigned to sites and group values, it is required to find the devices that are on the sites that match the selected values. Besides the versions that are grouped on packages and have to be *expanded* in order to retrieve the full list of file versions. This computing will be the most demanding operation on the system. These topics are explained thoroughly on the Chapter 3.

Being *Windows CE* the main mobile OS the CS customers use, this is clearly the focus of the first implementation. Released in 1996, *Microsoft* licenses *Windows CE* to Original Equipment Manufacturers (OEMs) so they can modify and create their own user interfaces and functionalities like *RFID* and barcode scanners. This OS usage is decreasing and more modern systems are increasing like *Android* or *iOS*. But in order to do a complete shift, the customer have to remove all those old devices (usually RFID readers) to embrace the newer platforms and that costs a lot of money. So the support for this OS is important, because MDM will be used on the devices the customer already have and don't make them buy new ones. The main target platform on *Windows CE* is the *.NET Compact Framework* which is a cropped version of the *.NET Framework* which libraries are scaled down to use less space. Most methods were removed and the hardware limitations are challenging making the development for this devices a hard task. Because this has to be a future-aimed solution, the *Windows CE* can't be the only supported mobile OS and for that matter it has to be implemented with that focus and make the later development faster for those *os*, as the logic base will already be defined.

1.2 Document Structure

The chapter where this section is included introduced the reader to the problem but there are more chapters on this thesis, such as:

Chapter 2 Explains the concepts behind the MDM and in Section 2.1 some applications that are already available on the market will be analysed.

Chapter 3 On the research I've done on the retail business, the company's customers and the MDM solutions, this chapter has the general explanation of all the project's components and structure. With the definitions defined here, the development phase will take them in consideration.

Chapter 4 The implementation and some libraries used to help with the development are defined in this chapter, as well as a security analysis.

Chapter 5 I make the project's evaluation and propose some future work that can be done.

Appendix A Has screenshots to complement the thesis related to the *BackOffice (BO)*.

Appendix B Agent Screenshots are shown here.

2

MOBILE DEVICE MANAGEMENT (MDM)

MDM is a type of security software used by a company's IT department to ensure security, access control, install software remotely, monitor, manage and secure employees' mobile devices that are deployed across multiple mobile service providers and OS's. Optimizing functionalities and security of mobile devices within enterprise, while simultaneously protecting the corporate network. In order to facilitate this control, a program is required to make the connection between the device and the remote service that wants to control it, typically named Agent.

Bring Your Own Device (BYOD) is a theme that is growing recently due to the popularity of mobile devices and consequently leads to employees to use their devices on the job rather than the company's. It is already common in many businesses and in a 2012 Cisco survey that took place in the United States, 95 percent of the respondents said that "their organizations permit employee-owned devices in the workplace." (Cisco, 2012) This survey also estimated that the average "knowledge worker" uses 2.8 connected devices at work. Usually this is an issue for the IT department in organizations as they have to manage their employees devices, to ensure information security and access control. The greater the variety of devices and its quantity, the arduous it is to manage them. So the MDM is a service that is gaining impact and market presence as it appears to help these situations (Finneran, 2011). According to Hayes and Kotwica (2013) four in ten enterprise level organizations had a security breach related to BYOD.

The use of a management system as the MDM leads to a reduce of support costs and allows reports on the state of the device to be obtained, thus preventing problems by reducing the time in which the devices are unusable. During these situations of error, it is required that someone has to go to the device's location and proceed with re-installations or updates. If a program can do these requested operations remotely, the time to fix the problems would be greatly reduced and cheaper.

The focus of this dissertation project is not to secure company information as it is one of the topics of the MDM's implementations but it is focused on the applications installation and configuration and its deployment to the devices.

As Rhee et al. (2012) explains, there are five steps (which Step 4 and Step 5 are repeated regularly and as needed) in a device life cycle on a MDM system:

Step 1 - Enrollment The mobile device data and user data of the organization are registered in the MDM system and the policy to be applied to each mobile device is configured.

Step 2 - Distribution The agent is distributed and installed in the users' mobile devices. The agent can be distributed through the application store/market or in-house.

Step 3 - Authentication When an agent runs after the installation, the mobile device data (IMEI, IP/MAC address, phone number, etc.) are sent to the MDM server to verify if they match the data registered in the system.

Step 4 - Instruction The MDM server sends to an agent the mobile device control policy and commands like "remote wipe" according to the mobile device status data and user.

Step 5 - Control/Report The agent controls the functions of the mobile device according to the mobile device control policy/command and reports the results to the MDM server.

These steps were taken in consideration during the project development. As this is a growing theme, there are many MDM solutions available and the next section will analyse some of them to obtain information and collect procedures and the best methodologies that they have to include in this project.

2.1 State of the Art

The first thing to do before starting to think in a project's implementation or structure is to do a research on the subject. It will teach a lot about the research problem and by reading literature related to it I will learn from other researchers, becoming easier for me to understand and analyse the problem. It proves that this thesis problem has relevance and if many people are trying to solve the same problem as me, I hope to prove that this problem I am trying to solve is important. In this section I will present some of the most popular MDM solutions with their advantages and drawbacks by relating them to the thesis requirements to the software I am planning to make.

During the development, some difficulties will arise and before stepping into them in the later phase, some can be prepared and studied in the research stage. These are the cases of dependency resolution, distribution and cross-platform support.

2.1.1 *MDM Alternatives*

There are on the market several MDM solutions, on this section I present the ones that I consider the most important and popular in the enterprise world. Some requirements are set as essential so that a solution can be defined as potentially able to be used:

Windows CE support This OS is critical due to the handheld devices on the clients, e.g. *Motorola MC55A* or *Denso BHT-1281* running on *WinCE*.

***Android* or *iOS* support** Have at least one mobile OS supported besides the *WinCE*.

Easy setup Quick/simple installation of the MDM system configurations. The steps needed to install the MDM on a *clean device* have to be easier and shorter as possible to facilitate a new device registration on the system.

Device Grouping A method to group devices.

Devices and Locations Link devices to sites/stores so that a device becomes related to the location that it physically belongs.

App Versions Have multiple release versions per application and don't overwrite the old ones.

Deploy Apps to groups Distinct devices can receive different applications. Don't always deploy a new version to all the devices but have the possibility to deploy it to a limited list of them.

Organize Locations Define the company's internal sites/stores organization and relate devices and its locations to it.

Install and Configure Install apps and apply configurations on the device.

Log the actions Perform logging and report device's execution steps and status to a server.

2.1.1.1 *Microsoft Intune*

Microsoft has a paid management system, data, mobile devices and computers protection application called *Microsoft Intune* [1] included in the Enterprise Mobility Suite. It is managed through a web interface and allows you to control what devices can install and run and define some system settings through policies. It is focused on companies with multiple devices for their employees and the notion of BYOD.

[1] Web Page: `www.microsoft.com/pt-pt/server-cloud/products/microsoft-intune/`

It doesn't need an infrastructure to do the management because it is in the cloud and can also integrate with *System Center Configuration Manager* in this respect thus extending the existing policies. Without the need to buy some servers and bandwidth contracts, this cloud environment is getting popular as it is cheaper to extend the capacity or power than it is by doing it internally.

Intune resembles with what the company intends as it includes devices registration, application control, it groups devices, reporting, configures *Wi-Fi/VPN*, it supports *Windows/iOS/OSX/Android* and remote application delivery. It also contains the *Mobile Application Management (MAM)* with *Office* and multi-identity, allowing for example to block some user to copy enterprise information documents.

The control that you want to have with the devices must be passed to the customer and the company will have access only to support scenarios. Plus it has a limit of 7,000 users and 4,000 devices or 25,000 users and 50,000 devices only contemplating the *Intune*'s MDM solution. These values seem satisfactory at first glance, but several CS customers have a higher number of devices and can be a problem in the future as the solution grows. The partial support for *Windows CE* makes this a possible option but these usage limits are not enough in most existing customers.

2.1.1.2 *Amtel MDM*

Amtel is the only vendor that provides *Software as a Service (SaaS)* solutions for 3 mobile devices business management aspects - Security, MAM and Cost Control. A web page controls all the solution with the same system of rules, settings, dashboards and reporting simplifying mobile collaboration in companies, leading to better productivity.

The MAM[2] product restricts access and actions to mobile devices by keeping a barrier between the device's public/personal use and private enterprise applications. It also distributes applications with access to a private store where users can do installations. You can send application updates, order to remove applications, block applications access and usage, distribute recommended applications, configure the device and it is compatible with *App Store* for *iOS* and *Play Store* for *Android*.

This solution is only present in *iOS* and *Android*, so it is not a valid option to use in the company's projects that require *Windows CE*.

2.1.1.3 *IBM MaaS360*

Started by *Fiberlink* and now part of *IBM*, *MaaS360* [3] has great tools for MDM by supporting a great variety of mobile OS's like *Symbian* and *Blackberry* besides *Windows* and *OSX*. This

[2] Web Page: https://www.amtelnet.com/solutions/mobile-security/mobile-apps-management/

[3] Web Page: http://www.maas360.com/products/mobile-device-management/

solution includes a document with the BYOD's ten commandments (Tsang, 2016) where they explain its focus on MDM.

As expected, it allows the devices registration through their application, contains control and settings policies, has its infrastructure in the cloud, it allows remote actions on the device such as deleting data, lock the phone, get its location, hardware and status information, a catalog of applications and document sharing.

The distribution of single or packed applications for groups of users or devices is one required feature so that the MDM which the CS will use must contain. *MaaS360* has this important feature, which is lacking or not good enough on the previous solutions. It has a good list of features and it is very focused on solving the problems of BYOD which makes it a good choice for companies seeking this type of control. The lack of support for *Windows CE* makes this not a viable option for the company's requirements.

2.1.1.4 *AirWatch*

As an *Enterprise Mobility Management (EMM)* software, *AirWatch*[4] is a software and standalone management system for content, applications and e-mail. The goal of EMM is to determine if and how available mobile IT should be integrated with work processes and objectives, and how to support workers when they are using these devices in the workplace (Kietzmann et al., 2013). This software allows the access to work apps and information directly from a mobile device in a simple way. The access to those work apps is done without the need to worry about joining a VPN, manually setting up existing apps, or entering credentials for the *AirWatch* apps. The login is done using only the corporate e-mail. *AirWatch* has serious concerns about privacy by ensuring each client's personal information and data are kept separate from the work apps. With *AirWatch*, the IT department can securely provide business resources and apps to their devices while keeping their personal information private.

Regarding the Agent, it can be used to view the Device details, access up-to-date information and make sure you're following the rules set by the IT, access important messages sent by the IT department or contact the IT admins via phone or e-mail for additional support, configure telecom management to ensure the data usage doesn't exceed limits set by the IT department and completely separate corporate data from the personal stuff by keeping the work apps and data in separate containers.

The Catalog has the purpose to browse and install work applications from the company's own app store so that the users can access the applications recommended by the IT department across all of the enrolled devices. Contains the ability to browse the apps by category for quick and easy access and rate and review both public and internal apps by making comments visible to other users.

[4] Web Page: http://www.vmware.com/products/enterprise-mobility-management.html

The enrollment process in *AirWatch* uses profiles that have been pre-set by the administrator based on device type, ownership model or organization group and when a device is registered, the Agent automatically begins downloading the matching profile. Administrators create profiles from the *AirWatch* console that push enterprise applications, enable monitoring and enforce automated compliance through the *AirWatch* compliance engine. AirWatch (2016) discusses common enterprise use cases and define the advantages of MDM, containerization and the security benefits that may be realized when both are deployed together.

2.1.1.5 *MobileIron*

Just like the previous products, *MobileIron* [5] ensures data protection and applications control on the devices. Started in 2009, they proclaim themselves as the the EMM leaders. The MDM functionalities are focused on information security and mobile device configuration of different OS's supporting safe e-mail, automatic configurations, security via certificates and corporate data wipe. This is the purpose of their MDM, provide the companies secure and control insurances over mobile devices, applications and provided content, protecting the employee's privacy.

MAM is another set of functionalities which purpose is the deployment of applications to the employees devices. It includes an application catalogue, authentication access control and separation of personal and corporate applications.

The glsmdm and MAM solutions provide corporate data protection and applications availability but there is no support for *Windows CE*, the most important requirement.

2.1.1.6 *Review*

Any of the solutions presented in the previous section could not fully meet the essential requirements, but *Microsoft Intune* is the one that is closest to the desired features failing on the usage limits and the *AirWatch* is the market leader and has the customer success stories to prove it. Therefore, it will be developed a new MDM. The company is known for solutions tailored for the customers and it is intended that this solution is generic enough so that some changes required by the customers become minimal.

2.1.2 *Dependencies*

In software engineering, dependency is the degree to which each program module relies on each one of the other modules so when a class `A` uses another class or interface `B`, then `A` depends on `B`. `A` cannot complete it's work without `B`. For this reason the class `A` is called the

[5] Web Page: http://www.mobileiron.com/

dependant and the class or interface `B` is called the *dependency*. A dependant depends on its dependencies and this dependency is directional. Meaning that if `A` depends on `B`, it doesn't mean that `B` also depends on `A`. In the cases that there is a relation between two or more modules which either directly or indirectly depend on each other to function properly, it is called Circular Dependency.

Some challenges with shared libraries can happen as it is common for applications to use external dependencies relying on dynamic library linking, instead of static linking. This dynamic linking allows the sharing of executable libraries of machine instructions across applications. In these scenarios, complex links between different applications that require distinct versions of libraries can result in a situation frequently known as *Dependency Hell*.

2.1.2.1 *Dependency hell*

Dependency Hell is a common term for the frustration of some software users who have installed software packages which have dependencies on specific versions of other software packages (Jang, 2006). A software usually uses libraries that are already available so that the programmer doesn't have to code something that is already made, promoting reusability. However, in the software world, where components evolve rapidly and depend significantly on one another, this problem becomes more pronounced (Donald, 2003).

This dependency issue appears around shared libraries on which other applications have dependencies but they depend on different (and sometimes incompatible) versions of the shared library. If the shared library can only be installed in a single version (only one installation in the system), the user may have to address the problem by obtaining newer or older version of the dependent application. Which may lead to other dependencies incorrect behaviour and push the problem to another set of applications.

2.1.2.2 *Conflicting dependencies*

Take the following example, `App A` depends on `libExt 2.5`, `App B` depends on `libExt 3.0` and different versions of `libExt` cannot be simultaneously installed on the system, therefore `App A` and `App B` cannot simultaneously be used because `libExt` can only have one version installed. A solution is the `libExt` to allow simultaneous installations, unlocking the usage of different versions.

2.1.2.3 *Private per application versions*

Private DLLs are a solution used on *Windows* OS to prevent dependency hell. There are copies of libraries per application in its directory (where it is installed). When the application requests the libraries, first it searches on the local installation path so that it is always prioritized and then it searches on the system directory with the system wide libraries.

Dealing with dependencies is not easy and can lead to problems like malfunctional programs. To address these problems, the Package Managers appeared.

2.1.2.4 *Package Manager*

A Package Manager is a collection of software tools that automates the process of installing, upgrading, configuring, and removing programs. A package manager deals with packages, distributions of software and data in archive files. Packages contain *metadata* with information about it and the most important is the list of its required dependencies so that it can run properly. Package Managers maintain a database of software dependencies and version information to prevent software version disparity and missing prerequisites.

The *RPM Package Manager* is one of the most popular package managers used in many Linux distributions.

2.1.3 *Cross-Platform Development*

A cross-platform software is a computer software that is implemented to execute on multiple computing platforms, typically OS's. Cross-platform software may be divided into two types: it requires individual compilation for each platform that it supports or it can be directly run on any supported platform without special preparation. For instance, a compiled Java code can run on all platforms that support Java without the need for recompilation.

As this MDM project first OS implementation is targeted to be *Windows CE*, Java can't be considered as it is not supported. *Windows CE* has *.NET Compact Framework* as the main framework for development, suggesting the usage of *C#* as the desired programming language. With the Mono Project software platform[6], it is possible to create a cross-platform C# application. Started in 2002 and currently maintained by Xamarin, It is an open source implementation of Microsoft's .NET Framework based on the *ECMA* standards for C# and the Common Language Runtime. It brings the *.NET* capability to other OS's like *Linux* and *macOS* providing a runtime environment to execute *.NET* programs, compilers for various source languages and implementations of the core class libraries specified in ISO/IEC 23271 (ISO, 2012). Several books discuss development with Mono, such as Schönig and Geschwinde (2004); Kaan (2007); Dumbill and Bornstein (2004); Mamone (2006). Other implementations can be found in King and Easton (2004). But Mono doesn't directly support *Android* and for that scenario *Xamarin* appeared.

[6] Web Page: `http://www.mono-project.com/`

2.1.3.1 *Xamarin*

With a C#-shared codebase, developers can use *Xamarin*[7] tools to write native *Android*, *iOS*, and *Windows* apps with native user interfaces and share code across multiple platforms. With the *Windows CE*'s *C#* requirement and the *Xamarin* mobile OS support, this platform is ideal.

The MDM logic will be the same independently of the OS and if it is written only one time, then the major concern when supporting a new OS will be the *Graphical User Interface (GUI)* that will be specific for each one.

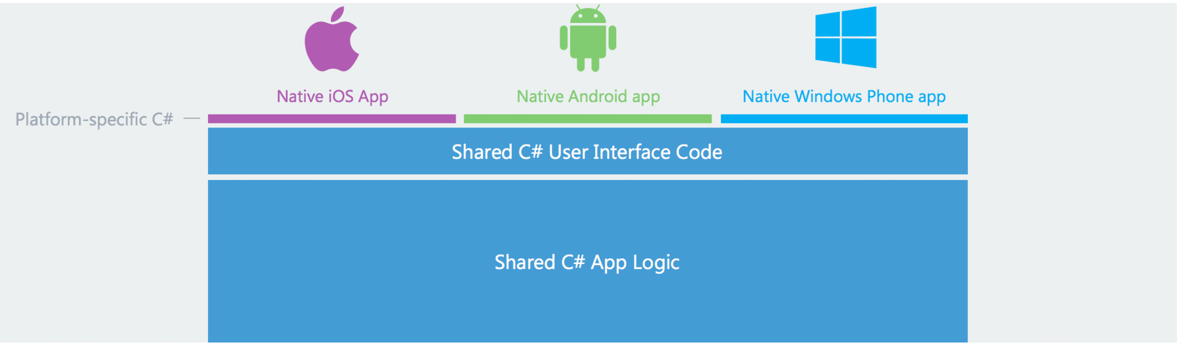


Figure 1.: *Xamarin* - Share code everywhere

[7] Web Page: https://www.xamarin.com/platform

3

ARCHITECTURE

As this proposal is to be used in the future CS's client releases, it is obvious that a working solution has to be implemented. This development will test the Rule System and Group Hierarchies concepts on real-life scenarios. The majority of devices used by the clients have *Windows CE* as their OS, making it the logical choice for the target of the first implementation. The following decisions will take this decision in consideration.

The installation, configuration and updating of applications on the device should be handled by an Agent (3.1) that is necessarily installed on each device. Its single point of communication with the server is done through a *Application Programming Interface (API)*. This remote service is responsible for informing the Agent which versions of applications and configurations it must install and provide all the necessary information about them, like name, version, size and file transferring details. The WS will have to calculate, by the rules created by the customer, what File Versions and Configurations have to be installed on each device. These Rules are assigned to a set of Groups Values, values which when combined create a Site filtering. These Rules may even be as specific as indicating what a particular device must install or configure. Groups and their Values will be organized by the client in a generic way thus giving freedom to shape its company's scheme as it sees fit, which is an important point as it allows a wider range of customers by reducing the need to create a solution of MDM specific to each new customer. This management should be done on a BO, a web application that will connect the user to the system.

Additionally, this solution should also be used to monitor the devices state, comparing what they have installed with what they should have installed or what errors occurred during installations. This information will be available in a special part of the BO called Dashboard. Its purpose is to display the status of the entire system to the user, allowing the customer to have a vision of what is working and what is failing.

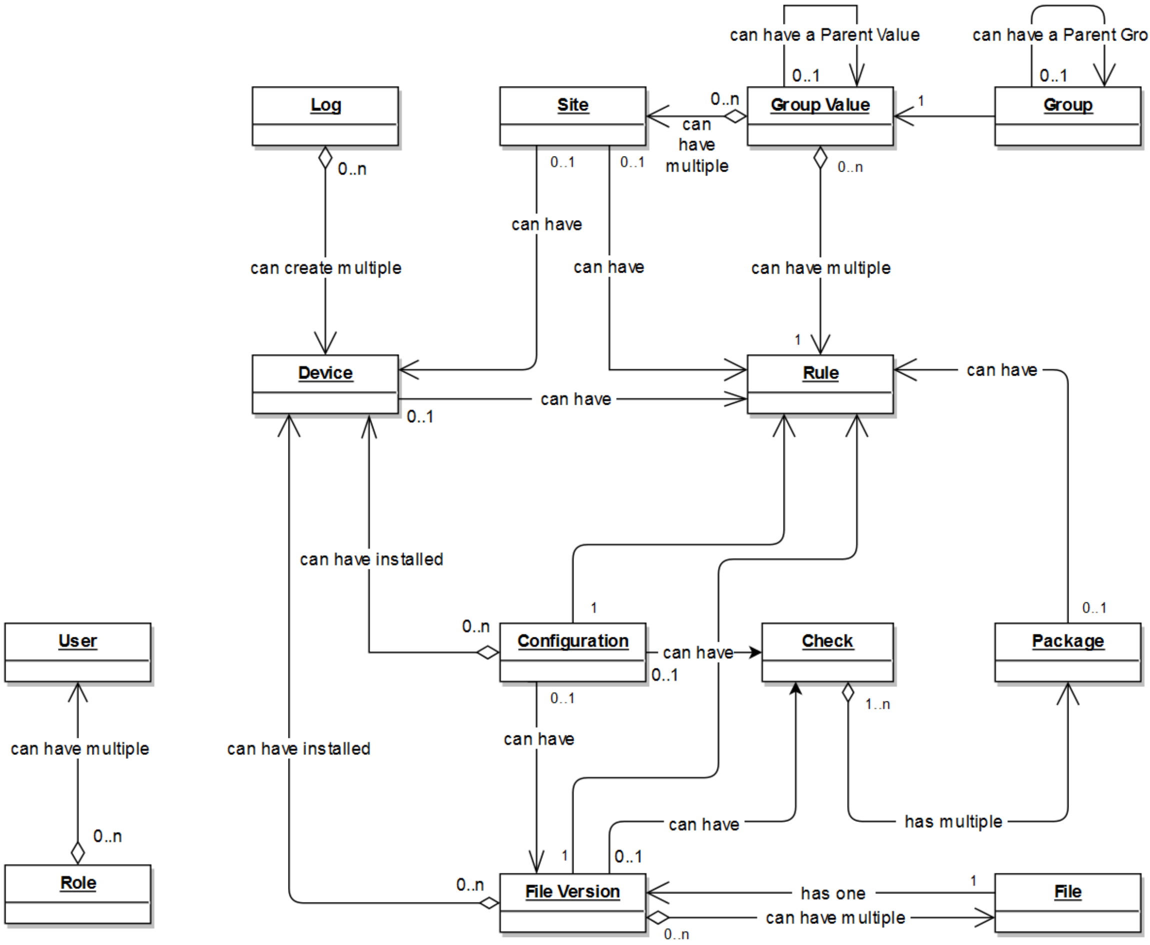


Figure 2.: Entities Diagram

Figure 2 is an UML Entity–Relationship model that links all the entities represented the system. Each entity has a section in this chapter that explains its function and importance.

A Device can have installed many File Versions, each File Version is related to one File and can have one Configuration. A Device creates many logs and can have a Site set. This is an example of some knowledge that can be extracted from this model and it can be used to implement the *Database (DB)*. The Users are not linked in the diagram because it would generate arrows connecting all the entities, as the User exists in the BO's context and it's responsible for handling the entities CRUD, I didn't include it in the main diagram.

3.1 AGENT

Monitoring application updates imply that there is an entity controlling the search, notification and installation of them. It is intended that this entity is configurable and is installed on multiple devices with different OS's to manage the versions of the applications and report the system status. In case of any updates installation, it should be able to safely install them so that if a problem happens, a roll-back is possible to the previous working state. In addition to applications, it is also necessary to be able to configure settings, such as changing system variables/registry and transfer files by placing them in specific folders. This is the Agent, the entity that will coordinate the updates on the devices.

The Agent is the application that the user will use to manage the device, it will connect with the WS through an API to synchronize and fetch the necessary data. It has various menus (Figure 18 on Section B.1) for the user to navigate and view the device information, check for updates, change the Agent settings, restore installations, change the site and force the synchronization.

There are some important procedures like Logs (Section 3.1.1), Registration (Section 3.1.2), Installation (Section 3.1.3), Updater (Section 3.1.3.2) and Synchronization (Section 3.1.4) that are explained on the respective sections.

3.1.1 *Logs*

Logging is the act of keeping a log that records events and information about a program's execution. A data collection method that automatically captures the type, content, or time of transactions made by a person from a terminal with that system, as Rice (1983) defined. This information is generally used by programmers for debugging purposes and by software monitoring tools to diagnose problems with software.

During any action, the Agent will produce logs reporting the progress and outcome of those operations. For instance, an installation log has the corresponding File Version or Configuration identifier for future easier reference and linking. I characterized 2 types of log storage:

Database The logs stored in the DB will use the identifiers (*id*'s) that came from the WS to link all the actions. With this direct information, the Dashboard (Section 3.8) can use it to perform better data analysis and the server to know for sure what File Version or Package it is related.

Files File-based logs contain some outputs and errors that the Agent produces during its execution and will be used for later support to find errors, dumps and stack-traces. This type of logs will be helpful for the developers to use information from the devices that are on the client. As there are log levels, the trace log can have more or less output information. This log level can be changed remotely to request more detailed logs from a device when necessary.

These logs are sent during synchronizations and erased from the device as soon as they are uploaded. Therefore, inform the WS what a device has installed and configured and what it has been doing as it is important for later support, to know what the Agent was performing and what was its state. The Database Logs are important for the server statistics and workflow, being the File Logs used only for later support reasons when necessary because the information on the Database Logs may not be enough to understand the situation.

Database Logs can be linked to form a tree structure in order to relate a log to a parent log. This connections are important to understand what originated that action, for instance, to know if a certain application was installed from a package. This linkage is more important on Packages (Section 3.2.2) because a package can be marked as not installed if one of its file versions failed during its installation and with this linking it is possible to understand the operations that the Agent made. Every File Log is associated with a Database Log and a File, with it the user can access the corresponding log. Check Section A.5 for an example of a log with a tree representation.

3.1.2 *Registration and Device State*

For a device to enroll in the system, it has to go through a registration process. During this registration, the device will inform the WS about its characteristics such as Serial Number, OS Version, Manufacturer and Brand. After this request, the agent will have to wait for it to be approved.

There are 2 boolean states that a device has, Approval and Enable. The Approval state will focus on the device's initial phase that will manage the Registration and when it requests a Site change, after it is approved the Enabled state manages the rest of its life until it is deleted. Figure 3 illustrates the 4 Device States Phases.

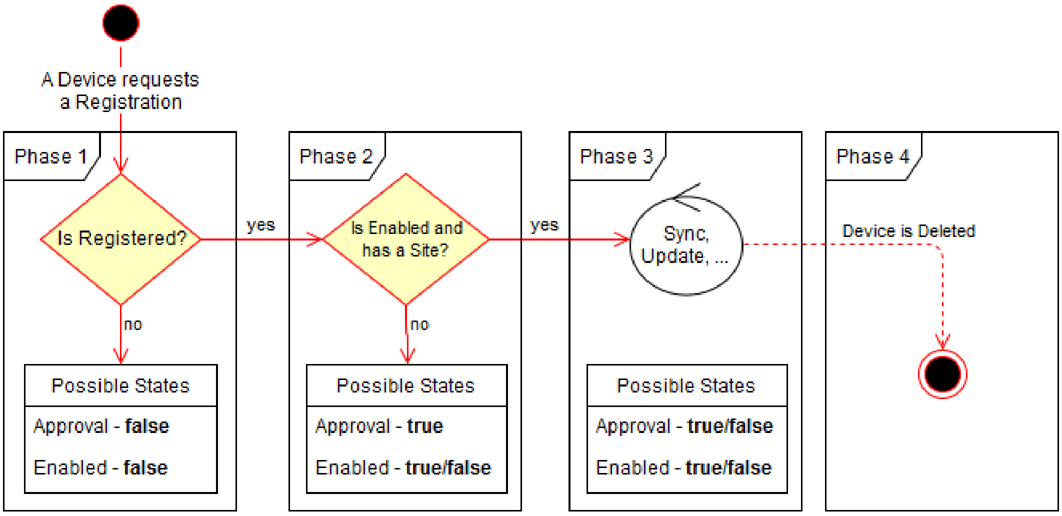


Figure 3.: Device States Diagram

Phase 1

The registration request is the first action the Agent will do to enrol itself. After this request, it will be marked as Disapproved and Disabled. In this state, a User on the BO will have to approve the device in order to proceed to the next phase, or reject it and the device will be deleted.

Phase 2

When the Device is approved the Agent will receive the WS endpoints, the Agent and Updater *id_file*'s, its *API-KEY* and the Site that it is in. Because the endpoints are sent from the WS when the device requests its profile data, the server can change them without the need of an Agent update as the result of an *Uniform Resource Identifier (URI)* modification.

In order to facilitate the registration of a Device and taking into consideration that only approved devices can use the MDM, it is required a quick start mode. A config file can be set in the MDM's installation folder that will contain the WS endpoint and some default configurations:

```
URL_DefaultHost=www.mdm.uminho.com
URL_DefaultPath=api/v1/mdm/
URL_DefaultProtocol=HTTPS
PORT=443
LANGUAGE=EN
```

Listing 3.1: `CS_Agent.conf` file

If this file is not present, the Agent will launch with default settings and will prompt the user to type the WS endpoint as seen on Section B.3. This startup file skips the initial system configuration that consumes time and can lead to errors or bad configurations. If a device was previously registered on the system, the Agent will boot up directly to the Main Menu. When *Denso* devices battery is drained out, sometimes they lose all their data because the devices do a hard reset and this file can be a way to ensure the Agent will install and start properly, recovering its state.

When using a new device, its addition to the system does not need to go through a manual registration in BO, so whenever a device tries to connect to the WS, the WS must check whether it is already registered on the system via Device Number (hardware unique code). If it is not registered, the server makes an entry in the DB with its information. The User on the BO can complete its information or change it. With this method it is expected that the deployment on new Devices becomes quicker but the manual installation of the Agent is still required, which is the only operation required for the system to load properly.

Once a Device is approved and if a User haven't assigned a Site to it, the Agent will request the person using the Device to select one from the available list. It is possible to filter the list using Group Values (explained on Section 3.4 and represented on Section B.5). There is a particular situation in this case, once the Agent informs the WS the selected Site it becomes assigned to it without the need of approval. But in the future, every time the Agent requests to change the site for its device, the device becomes disapproved until a BO User approves the change by re-approving the device. In order to proceed to the next Phase, the Device must have a Site defined and be enabled and approved. This auto-approval case is to ensure the quick-setup mode to skip a BO action.

So, 2 situations can occur in this phase when a user is going to approve a device registration request:

User also sets a Site The user approves the device and sets a site to it, then the Agent doesn't have to request one and continues.

User doesn't set a Site It is possible that the User is just accepting new registrations and not assigning sites to them. In this case, the Agent will understand that and request the User to choose one from the list.

Phase 3

This Phase contains all the operations a device does, like Synchronizations and Updates, until it is deleted and goes to Phase 4. In Phase 3 a Device can be disabled and enabled. A disabled device can't operate until it is enabled. Once the Agent starts up or do an operation, it will check the device state and inform the user about it if it is not allowed to do it. The device can be Disapproved if it requests a Site change, being in that mode until a User on the BO approves the change or sets a new Site.

Phase 4

A Device can be deleted from the system and won't show up in the BO and it is the last Phase a device will be. But it can come back and recover its definitions. If a device that was deleted registers itself again, it will go to the same process beginning on Phase 1 and the WS will recover its data because its the same Serial Number and won't make a new registry.

3.1.3 *Installations*

In a device point of view, it needs to know what files it has to install and what configurations it needs to apply and in order to get that information, it has to request the WS about that. The Agent just needs to get a list of file versions and configurations and proceed with its installation. The response looks like the content of Listing 3.2:

```
[
   {
      "id":103,
      "id_InstallationPackage":null,
      "id_FileVersion":12,
      "id_ConfigurationType":1,
      "id_Configuration":10,
      "InstallationOrder":null,
      "DestinationPath":null
   },
   {
      "id":102,
      "id_InstallationPackage":100,
      "id_FileVersion":14,
      "id_ConfigurationType":null,
      "id_Configuration":null,
      "InstallationOrder":1,
      "DestinationPath":null
   },
   {
      "id":102,
      "id_InstallationPackage":100,
      "id_FileVersion":112,
      "id_ConfigurationType":null,
      "id_Configuration":null,
      "InstallationOrder":2,
      "DestinationPath":"C:/path/to/copy/"
   }
]
```

Listing 3.2: Installation Checks JSON response

The majority of this content are *id*'s because it is only required to get the full information about a File Version or a Configuration if the Agent has to install it or is not in its internal DB. This feature is intended to reduce traffic to the WS leading to less requests and information on payloads. The *id*'s will be used on Logs and on server requests. From this example, the Agent understands that it has to install File Version *12* with Configuration 10 and then the Package *100* with two File Versions, first the *14*th and then the 112 that will be copied to a specific path.

3.1.3.1 *Installation and Restore*

Installing a File is quite simple and three installation methods were defined:

Copy Copy a file to a respective folder

Unzip Extract a compressed file to a respective folder (e.g.: .zip)

Install Install a file (e.g.: .cab)

The Agent before proceeding with an installation ensures that it has the installer from the currently installed version, if it does not then the Agent downloads it. This option is important if the new installation results in an error, the Agent can re-install the previously installed version. But if the installation is successful, then the Agent can delete the installer from the now previous version and retain only the current one. An installation can occur in an error by various reasons, like when extracting or copying a file to protected folders, the file is corrupted or by hardware reasons. So the Agent has to be prepared for these situations and whatever the outcome, a log file is uploaded that can be useful in error situations.

The Restore is an option that the User can use to re-install the current version of an application. If there is an online connection and there is an update to that application, the Agent only shows the update option forcing the application to be updated.

If a new application has been installed or a configuration applied, the Agent informs the WS about it during the synchronization. Like this, the WS has information regarding what each Device has installed.

What if the application to install is an update for the Agent? The Agent would have to shut itself down and call the installing process. If any error occurred there would not be any logs. The solution is to create other entity, the Updater.

3.1.3.2 *Updater*

The Updater has only a single purpose, to update the Agent. This is an usual approach in software engineering, to create a separated program that will handle the update of the main application.

When the Agent detects the installation is an update for itself, by checking if the File id is equal to the Agent id that it gathered from the server, it calls the Updater informing where is the installer and it closes itself. This information is stored on the Registry and the Updater will read from there. The Updater installs the Agent, logs the activity, marks the installation as completed on the Registry and launches the Agent which will collect those logs and synchronize.

After its update, the Agent will continue with the rest of the installations if that is the case. With this coordination, the User will update several applications, including the Agent, and just have to wait for all to finish because he won't be prompted to any confirmation. This cycle is explained on the next section.

3.1.3.3 *Batch Installations and Packages*

The user can choose what he wants to install, it is not required to install all the available updates but if there is an update for the Agent or the Updater, the Agent marks those updates as mandatory. This situation is important if an installation can lead to an error and the newer Agent version fix that problem. It's a way to ensure that the latest Agent and Updater are installed. Given a list of updates to install (like Listing 3.2), the Agent executes some steps to proceed with the installations:

1. Check the list if it has an Agent or Updater
    1.1. If it has the Updater then install it
    1.2. If it has the Agent then install it
    1.3. If a problem has occurred, abort the installation
2. Check the list if it has Packages
    2.1. If it does, order the File Versions from each package by Installation Order
3. For each File Version on the list
    3.1. Check if the Agent has that Version already installed using the File Version id
    3.2. If it does, then skip that version's installation unless it has a Configuration attached and it has to be applied
    3.3. If the Version's Min and Max OS doesn't match the current system, ignore it
    3.4. For each Version that is not installed, add it to the *to install list*
4. For each File Version on the *to install list*
    4.1. Download the installers for the current version if they are currently installed and the Agent doesn't have it on the backup directory
    4.2. If the Agent doesn't have the installers for the Version, then download them
5. Begin the installation of those and apply the Configuration after each one if the File Version has a Configuration with it
    5.1. If a problem has occurred, abort the installation

Note that aborting the installation won't revert to the old installation by default, it marks that installation as failed. Example, if the package's installation from Listing 3.2 installs File Version with *id 14* and fails while installing File Version with *id 112*, the Package installation has failed but the File Version *id 14* has been installed successfully. To restore an application the user has to access the specific menu to do that as it is not an automatic procedure.

3.1.4 *Synchronization*

Synchronization is the process of establishing consistency among the Agent's DB and the server's data. It will keep the WS to be up-to-date with the state for each device. The sync is done whenever possible by the Agent and before important tasks that require the device to be synced, like the check for updates. The Agent sends the Database Logs and the File Logs to the WS and when it receives a positive response, he deletes them from its DB and log folder. The Database Logs contains 2 tables besides the logs themselves, the installed versions and the installed configurations.

3.2 FILES

A File is a representation of a computer file and has its basic characteristics, such as type (*.cab* or *.apk* to install, *.zip* to extract or a regular file to copy), OS, name, if it is Enabled or a Main Application. Multiple versions will be uploaded to the system regarding the same File as it is the way updates works, so it makes sense that only the changes are submitted as the information about the File remains the same. Dividing into File and File Version, it is possible to know that a Version *X* and Version *Y* belong to the File *F*. It will be useful when launching Rules (Section 3.6).

A Main Application is taken into consideration as an important File and has a special behavior when creating Packages and treating dependencies. It is considered that a Main Application may have dependencies on other no main files such as *SQLite* or *.NET CF*.

3.2.1 *File Versions*

When a File is created on the BO, the upload of its File Versions is unlocked and the User can upload its versions assigning them to that File. The User has to indicate its Version that has to be unique regarding all the versions for a file. This version identifier follows the Microsoft's Major, Minor, Build, and Revision scheme[1]. This scheme is also used to compare the maximum/minimum versions of the OS's that this version will run and should be set on the Version's details if it is desired to use this filter. For instance, if a Version is not

[1] https://msdn.microsoft.com/en-us/library/system.version.aspx

intended to work on later versions of *Windows CE 5.0*, then it should not be released to the Devices that doesn't meet this criteria. Other attributes can be set, such as the size required (useful for extraction or installations), the file path to copy if that is the case and a Configuration to be applied after its installation. The rest of the attributes are calculated by the WS after the file upload, such as size, hash and file name.

There is no order or sequence on the File Versions upload, meaning that if a File has two versions uploaded, the third one resulting from the next upload don't exactly mean that it is the File's most recent version just because it was the last one. So the order of upload/creation is irrelevant.

It is possible to edit a File or a File Version's details and even upload a replacing file for that version, but the Agent won't be notified of that change if it has already installed that File Version. Meaning that the Agent won't download that replaced file until the user requests a restore because the Agent just only for more information (including its Hash) during the check for updates phase and only gets the full details when it doesn't have that version installed (Section 3.1.3.3).

Because the File Versions are used on Rules and they are installed on Devices, it means that when the User wants to delete one from the system, a search for its usage has to be made. So, it is only possible to delete a File Version if it is not used in any Rules and no Device has that version installed. Regarding the database, the entry can't actually be deleted because the File Version identifier is also used in Logs and obviously on the History tables, so it is marked as deleted and won't appear anywhere on the BO. The same strategy applies to the File, it can only be deleted if all the File Versions related to it are removed. This is a usual behaviour when requesting a delete on any entity. There are history tables to free the tables from the deleted entries, every update or delete on a monitored table will insert the old row on the corresponding history table. This is done with triggers on the monitored tables.

The Agent has two methods to ask the WS about a File Version: get its meta info and request a download. The first one is used to get its information when the Agent gets its id from an updates check. The second one is to download and internally store the file for its installation. This download request has an extra step that the Agent must perform after its download, to check that the Hash from the downloaded file matches the meta info from the server in order to verify the integrity of the file. If the two hashes are the same, then the Agent can proceed with the installation.

3.2.2 *Packages and Dependencies*

Some applications may require other programs or libraries to work properly, called dependencies. These are some requisites that the Main Application must have to work properly and it may not work at all if any of those is missing. Meaning that when a BO user creates a rule demanding the installation of `App X`, he would have to create another one for each of its dependencies. It would not be a helpful way of doing this because he may forget to add one dependency and it won't be reusable to future situations. Because of this need, the notion of Package is introduced.

A Package is an ordered set of Files Versions which purpose is to indicate that a Main Application needs other programs or configurations to work properly, called dependencies. The installation order indicates the Agent the course to take in the installation of all files because the order may be important in resolving these dependencies. A Package can also include with each File Version a Configuration (Section 3.3) that will be applied at the end of the file installation in addition to the configuration that can be attached to the Version general definition.

There are some guidelines in the Package creation that must be met:

One Main Application There has to be exactly one Main Application in a Package

Ordered The File Versions are ordered

Same OS The Files on the Package must have the same OS as the Package's Main Application or have an Undefined OS (e.g. text files)

The Agent will get a package and its content already "expanded" and won't have to query the WS about it because when he requests it for what it has to install, the list will already contain the files with the respective package and its order. The Agent will have to detect it and order those to proceed with the installations.

Take in consideration the following example:

Main Applications	Regular Files
$\underline{\mathrm{F}_A}$	F_B
$\underline{\mathrm{F}_E}$	F_C
	$\underline{F_D}$

Package	#1	#2	#3
P1	$\underline{F_A V_1}$	$F_B V_1$	$F_C V_2$
P2	$F_D V_1$	$\underline{F_E V_1}$	
P3	$\underline{F_A V_2}$	$F_C V_2$	
P4	$\underline{F_A V_2}$		

Table 1.: Example of Packages and Dependencies

Following the example of Table 1, there are 2 underlined Files marked as Main Applications: F_A and F_E and 3 Regular Files, F_B, F_C and F_D. In Package *P1*, we have Version *V1* of the File F_A, Version *V1* of the File F_B and *Version 2* of the File F_C. This means that File F_A needs those Versions of Files F_B and F_C. Later was released another Package for the File F_A, Package *P3*, which indicates that the new Version *V2* of the File F_A only needs File F_C's Version *V2*. This loss of the need of File F_B indicates that the F_A no longer needs F_B to work. By just creating a new package, it was possible to create new dependencies without messing up with the older ones.

Another situation has to be analysed, there is also the Package $P4$ that addresses the same $F_A V2$ as $P3$ also does. This could mean that some devices may already have $F_C V2$ installed or it is not necessary for them, like drivers. So, there can be packages regarding the same Main Application.

The creation of Packages don't directly affect any device updates or installations unless it is assigned to them by a Rule. This topic is addressed on Section 3.6 as the Packages have influence in the Rules assignment and File Versions to install.

3.3 CONFIGURATIONS

The main focus of any MAM is to be responsible for installing applications on devices, however some applications may require additional settings or even the devices themselves. A Configuration is linked to an OS and have a Configuration Values list. This link with the OS is important because some configurations may not be valid on other OS's like Windows Registry values and *Android* Shared Preferences.

A Configuration Value contains a Type (string, number, boolean, ...) and a Key-Value pair. The Value type is linked to a Configuration Type so that the Agent will know how to set that parameter. These Configuration Types can be Registry Values (*Windows*) or Shared Preferences (*Android*). Assuming *Windows CE* as the target environment, some Configuration Paths and Configuration Value Types are defined as:

Configuration Value Types for *Windows CE*:

- String
- Binary
- DWord
- ...

Configuration Paths for *Windows CE*:

- Classes Root
- Current User
- Current Config
- Local Machine
- Users

With these definitions it is possible to simulate a Configuration:

Name	App1 Config
Description	Configure Language and Server Port for App1
Type	Registry
OS	*Windows CE*

Table 2.: Example of a Configuration

	Configuration Value 1	**Configuration Value 2**
Configuration	App1 Config	App1 Config
Description	Language for the App	Port for the server host
Key	`SOFTWARE/CS/APP1/Language`	`SOFTWARE/CS/APP1/APIPort`
Value	`PT`	`9876`
Value Type	String	DWord
Path	Local Machine	Local Machine

Table 3.: Example of 2 Configuration Values regarding the Configuration on Table 2

The configuration called "App1 Config" will create on the Registry two entries defining the Language and Server Port for `App1`. Note that only Key, Value and Description are stored as values on the DB and Configuration, Value Type and Path are stored as *foreign key*'s (*id*'s). The same occurs on Configuration's Type and OS as these are only keys and Name and Description as values. As a Configuration can have many values, this is a way to save storage space and reuse data taking advantage of the DB.

3.4 GROUPS

One of the objectives set to this dissertation was to propose a new approach respecting the organization of devices. The research done on Section 2.1 was important to collect some data on how other MDM solutions addressed this topic.

The general method that most of those MDM solutions used to tackle the organization and grouping is to request the User to choose the devices that he wants to gather and create a new group. So, every new Device that would come to the system would have to be assigned to a group. This may work well for a relative small number of devices and offices, but this could lead to more problematic situations when the Updates become involved and the number of devices increases. Because the project will be applied to retail customers, let's take them as an example in the following generic scheme defining a sample hierarchy to illustrate how typically they are organized in Figure 4.

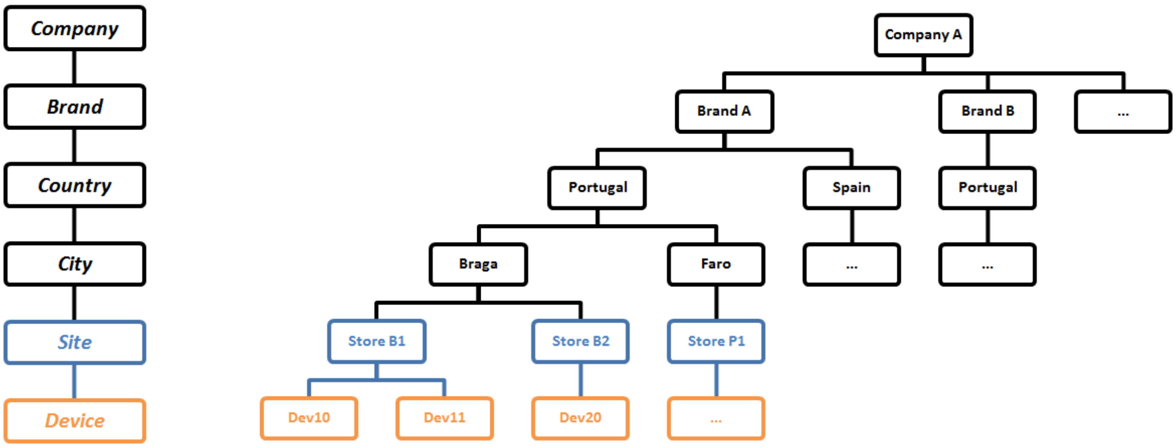


Figure 4.: Example of a Retail Customer's Hierarchy

Company A has two Brands in their possession that are linked to Countries, the *Brand A*, which exists in *Portugal* and *Spain* and the *Brand B* that is located only in *Portugal*. Following the *Brand A* in *Portugal* the next segment is City and there are two of them, *Braga* and *Faro*. The City is the lowest segment because that is where you assign the Sites (Section 3.5) and place the Devices. Sites *Store B1* and *Store B2* in *Braga* and *Store P1* in *Faro* are highlighted in blue and have some Devices on them, highlighted in orange, ending the hierarchy.

Regarding this example, whenever you, as an administrator, add a new Brand you will have to create a hierarchy similar to *Brand A* replicating that "branch". And as it can be seen, replication of data is one of many problems about this tree style approach. If we want to describe the segments that the *Store B1* belongs, we will have to start in *Braga* and pass by *Portugal*, *Brand A* and *Company A*. So, to define one Site it is required to set a value for each segment that structures the organization's tree. Because different Brands are on different sides of the tree, the same Country, *Portugal*, appears on both of them and can't be assumed as to be the same. These are the main problems about this organization scheme. But there is a simpler solution as an alternative, to break up the tree in order to make a more generic hierarchy and re-use already created values like *Portugal*.

I propose that by splitting the tree from Figure 4, it will make the solution more generic and a more relaxed form of hierarchy. Allowing this implementation to be adaptable to other contexts beside retail. It is crucial that there is a dynamism in the creation of the hierarchy so that it is not necessary to create versions of MDM for each CS client because the structure of their organization is different, while also preventing the client to use a structure defined by the CS. It is up to each client to define the best hierarchy that reflects the organization of its company and so that this dissertation can be used outside of retail environments.

Using the example in Figure 4, it is simpler if the hierarchy is defined in a dispersed form:

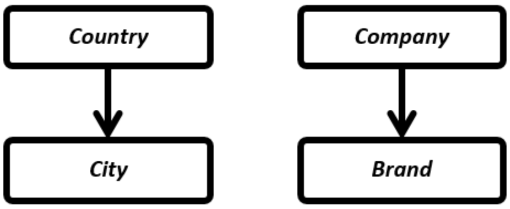


Figure 5.: Representation of Figure 4's dispersed Group Hierarchy

There are only two main "trees" but more can be added without interfering with the ones that already exist providing more detail on the definitions. In this Figure 5 an extra segment named Mall can be added after the City to demonstrate the flexibility with this scheme $Country \rightarrow City \rightarrow Mall$. These segments are called Groups and they will visually help to define the structure. The top-level groups are called Root Groups and each group can have one and only one link to another group. This link will make a subgroup have a parent group, e. g., Country is the parent of the City subgroup. The values like *Portugal*, *Brand A* or *Faro* presented on the example of Figure 4 are missing on this figure because they are defined inside each group:

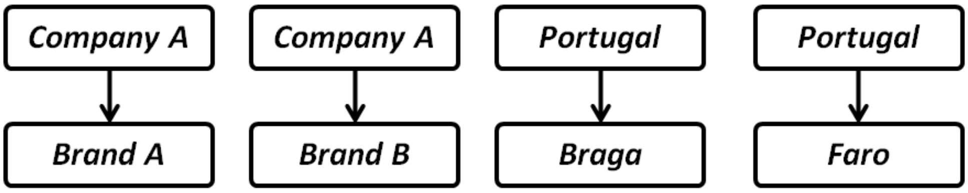


Figure 6.: Representation of Figure 5's Values

Figure 6 contains the Group Values definition of Figure 4 and just like the Groups, these Group Values also have a parent value to link them to an upper group value. Each Group can have multiple Group Values, as Country has *Portugal* and *Spain* and City has *Braga* and *Faro*. Because there is a parent relative to each Group Value, the values from the Root Groups have to be inserted first and as they are in the top, they don't have a parent value. If the group called Mall was inserted after the City (being City the parent group of Mall), the Group Values that already exists don't have to be changed in any way. To insert a Mall value it would be *Deluxe Shopping* with *Braga* as its parent value, representing $Portugal \rightarrow Braga \rightarrow Deluxe\ Shopping$. Because *Braga* has *Portugal* as its parent, this representation is possible.

3.5 SITES

A Site is the place where the Devices are located, such as a store, distribution center or a warehouse and has special features because it is the anchor between a Device and the MDM system. Each Site will have only one value per group branch, for instance, according to the example in Figure 5 a Site will have a City and/or a Brand because you can only choose the lowest Group Values of each branch. As stated before, each value has a reference to the Group's value above him and by pointing only to the lowest Group Value it is possible to recreate the values list to the root Group Value like *Company* or *Country*. It is not necessarily required to have a value of each Group, only the ones necessary to characterize the Site because the more Groups an organization has, the more detailed a Site can be.

Depending on the values assigned to it, the devices will receive updates based on those values. Thus if a device changes its Site, the Versions that it has to install may change as the new values on the new Site can be different causing different versions to install. This situation is explained in detail on Rules Section 3.6.

The dynamism created on the Groups will be reflected primarily in Sites because it is simple to define a Site, comparing to the previous example of Figure 4, it is only needed to assign a City and a Brand at most and values that can be changed later.

3.6 RULE SYSTEM

The purpose of the MDM is to assign applications to devices remotely and the Rule System was conceived for that. A rule can be the release of a version of an application to the devices in *Braga*'s sites or a language setting for all the devices that are in *Portugal* regardless of the brand in question. It states what it is to install on which devices.

I defined a Rule as a pair of a Content and a Target. A Content is something to install, such as File Version, Package or a Configuration that a number of devices must have installed. These devices are covered by the area of action from the Target definition such as a Site or Group Values. The previous definition of Group Values and its relation to Sites will make the creation of rules an easy task.

The Content has three possible combinations:

- A File Version with or without an extra Configuration
- A Package with or without an extra Configuration
- Only a Configuration

Note that a File Version can have a Configuration and if it is associated to a Package, it can have a second Configuration and if that package is on a rule with an optional configuration, there will be three levels of configurations. With these options, the client can use it to alter configuration values without the need to create newer configurations as the the order that the Agent will apply these configurations is from File Version, then the Package and then from the Rule.

The Target also has 3 possible combinations:

- A Site with or without a Device
- Group Values with or without a Device
- Only a Device

If it is intended that the rule is to be applied to a specific device wherever it may be allocated, then the user selects it as the only target. To deploy a rule to more than one device there are three options as stated before: select a site and all the devices that are in there will receive the update, select a set of Group Values to filter more than one site or create different rules with the same content targeting each device.

For the rules that target group values, the MDM system will have to find the sites that match those values. This situation is explained on 3.6.1. Targeting a site or group values, both of these options can have a Device specified along with it and only when the device is on that site or on a site with those values, it will have that rule available. This is important because a device can change its site and have different file versions when it is located on a specific site.

The launch of a new Rule implies that other rules and respective versions of files may be no longer considered if they are related to the same File. This is also the purpose of the Rules, to release newer versions and their dependencies to replace the older ones. The concept of dependencies on Packages is applied here again because version conflicts must be avoided. The calculation of versions and configurations to be installed by a device will be the most computational demanding part, it will have to go through several tables if the DB is concerned. The latest rule by File will prevail, if it is associated with a package the Agent must also install its dependencies.

3.6.1 *Processing Rules*

As explained on the previous section, a rule can have three types of Content and Target, and depending on each, the effect of the rule is different. In order to calculate what a device must install, it is required to iterate the rules and process them, filtering out the ones that doesn't matter for the specific device, like the ones to a Site that doesn't belong to that device or from a different OS.

The rules are prioritized by the creation date in a descending order, so the most recent rule is chosen over an older one regarding the same file. Because creating a rule for devices on sites in *Braga* can be a common creation, there should be a reuse in terms of the DB about the group values when used in rules. If another rule for *Braga* is created, it can use the same *row id* as the one created before without the need to create another DB entry. This decision is useful to not replicate data and promote reusability.

The following procedure finds what a specific device must install:

1. If the device has a site, then go to 1.1.
   - 1.1. Get the Site where the device is located
   - 1.2. Get the list of rules that are assigned to that site
   - 1.3. Get the list of group values from that site and then:
     - 1.3.1. Get the group values assigned to the site and add it to the list
     - 1.3.2. For each one of them, get its parent group value
     - 1.3.3. If that group value also has a parent add it to the list and go to 1.3.2.
   - 1.4. From the rules that target group values, get all those values
   - 1.5. Intersect the values from 1.4. that match the values from the site list of 1.3.
   - 1.6. Exclude the rules that have more values than the ones assigned to the Site
2. Get the rules assigned specifically to that device and add the ones from 1.6.
3. With the rules gathered so far (Target), proceed to find the file versions and configurations to install (Content):
   - 3.1. Filter the rules related to the device's OS or that have an undefined OS
   - 3.2. Get the Main Application for the ones that have a package
   - 3.3. Sort the rules in order to find the rule for the most recent file version by file, including the main applications from step 3.2.
   - 3.4. If any file version from the previous filter is related to a package, then expand the packages by listing all the file versions and linking them to the same rule
4. Get the configurations for the rules that only have it as content and add it to the list
5. Prepare the listing for the device

The step 1.6. will remove the rules that have more group values than the ones matching the site. For instance, if a *Site A* has only *Brand A* as its Group Value and a Rule is created to *Portugal* and *Brand A* this Rule will be accepted on step 1.5. but it is not valid because it requires the Brand *Brand A* and that site doesn't have it, being rejected on step 1.6.

Observe the following tables 4, 5, 6 and 7 that have the entities that will be used on the Table 8 to illustrate how the assigned rules dictate what a device has to install.

File	File Version	Configuration
F_A	V_1	
F_A	V_2	$Conf_1$
F_A	V_3	
F_B	V_1	
F_B	V_2	
F_C	V_1	

Configuration
$Conf_1$
$Conf_2$
$Conf_3$

Table 4.: Example of File Versions and Configurations

There are three Files, F_A, F_B and F_C with the respective Versions and three Configurations $Conf_1$, $Conf_2$ and $Conf_3$. Note that $F_A V_2$ has $Conf_1$ so when the Agent install this version it will apply this configuration. The contents of the configurations are not important for this exemplification and therefore they were not detailed.

Group	Parent Group
Company	
Brand	*Company*
Country	
City	*Country*

Group Value	Group	Parent Group Value
Company Inc.	*Company*	
Brand A	*Brand*	*Company Inc.*
Brand B	*Brand*	*Company Inc.*
Portugal	*Country*	
Spain	*Country*	
Braga	*City*	*Portugal*
Madrid	*City*	*Spain*

Table 5.: Example of Groups and Group Values

The Groups and Group Values are reused from the previous sections.

Site	Group Values
$Site_1$	*Braga, Brand A*
$Site_2$	*Brand A*
$Site_3$	*Porto*

Device	Site
$Device_A$	$Site_1$
$Device_B$	$Site_2$

Table 6.: Example of Devices and Sites

Table 6 contains the Sites (with the Group Values for each one) and Devices (with the assigned site) that will be used on Table 8.

Package	#1	#2
P_1	$\underline{F_A V_1}$	$F_B V_2$
P_2	$\underline{F_A V_2} + C_2$	$F_B V_2$
P_3	$\underline{F_C V_1}$	$F_B V_1$

Table 7.: Example of Packages

The Packages are listed on Table 7, indicating the File Versions and the respective installation order, 1 and 2. Package P_2 has an extra configuration, C_2, that will be applied by the Agent after the installation of $F_A V_2$.

The following table includes rules with a time index, Content and Target. The greater the time index the most recent the rule is, meaning that the rule **1** is older than **3**.

	Content			Target		
#	**File Version**	**Configuration**	**Package**	**Site**	**Device**	**Group Values**
1	$F_A V_1$			$Site_2$		
2	$F_B V_1$					*Spain*
3	$F_C V_1$			$Site_1$	$Device_B$	
4	$F_C V_1$	$Conf_2$		$Site_2$	$Device_A$	
5	$F_A V_1$			$Site_1$		
6			P_1			*Portugal*
7		$Conf_3$	P_2	$Site_1$		
8	$F_A V_2$				$Device_A$	*Braga*
9	$F_A V_3$			$Site_1$		
10			P_3			*Brand A, Portugal*

Table 8.: Example of Rules and File Versions for a Device

$Device_A$ is the device used as the specimen in order to analyse the Table 8. This device as seen from Table 6 is located on $Site_1$ which has *Braga* and *Brand A* as its Group Values (Table 5). Let's assume that the rules are launched one by one and we will evaluate each one and consider the ones that came before. Consider also that the Files and Configurations from Table 4 have the same OS as the device from this example.

Rule 1 is set for $Site_2$ which is not where the device is located, so it is ignored.

Rule 2 is set for all the sites in *Spain*, as $Site_1$ is in Portugal, this rule is also ignored.

Rule 3 is set for $Site_1$ so it may be valid for our target device, but it is specific for $Device_B$ and then it is ignored.

Rule **4** besides being targeted for our $Device_A$ it is only when the device is on $Site_2$. As it is on $Site_1$ right now, it is discarded until it is moved to that location.

Rule **5** is aimed for $Site_1$ only, meaning that this is a valid target. The content is the F_AV_1 and the only application the agent has to install.

Rule **6** was launched for sites in *Portugal* and as the site is in *Braga*, this is a valid rule as we can see from the group values from Table 5. It requires to install the Package P_1 which contains F_AV_1 and F_BV_2. As it has F_AV_1, it will replace the last rule as it has a version of F_A with a new dependency F_BV_2. This rule can be interpreted as the devices in *Portugal*, another version is required along with F_AV_1.

Rule **7** is like Rule 5, it is targeted for $Site_1$ and therefore it is a valid rule for this device. It contains the Package P_2 with an extra Configuration $Conf_3$. This package has a special case because it contains the file F_AV_2 with an extra configuration, C_2, and other file version, F_BV_2. The Table 4 shows that F_AV_2 has configuration C_2 and as the package also has an configuration, the Agent will apply C_1 first an then the C_2 from the package. But there is another configuration that was added on the rule definition, so the Agent will apply the C_3 in the end completing the installation. With this rule, the agent will install this package because this version of F_A prevails over the previous rules because they only issued versions of F_A.

Rule **8** is destined for $Device_A$ when it is in *Braga* and therefore it will be taken in consideration. This rule demands that this device installs F_AV_2 only, without extra configurations or dependencies as the previous rules demanded. Two situations can occur, if the device is clean (or it was cleaned) of installations, the Agent will install this version, being that the only version installed. But if the device applied all the rules one after another, as those were timelined, the configurations C_2 and C_3 and the F_BV_2 may already be installed, and the F_AV_2 behavior may be influenced by that. This can occur because the Agent don't uninstall applications.

Rule **9** is a valid rule for the device as it is targeted for $Site_1$. It demands the installation of F_AV_3, replacing the previous rule.

Rule **10** is the last rule and it is intended for the sites in *Portugal* and that are a member of *Brand A*. As the site belongs to both values, this rule is valid. The package that this rule issues has F_CV_1 and F_BV_1, as F_C was never delivered before, it will be installed as well as the F_AV_3 from the previous rule.

3.7 BACKOFFICE

The client will have a Web portal called BO to perform all system management features. It's purpose is to facilitate the administration routines and its interface is not seen by the store costumers. It must be possible to perform all system administration tasks on the BO, such as approve and register Devices, create Rules, manage Groups and Values, create new Versions, etc... Like these actions, all the entities described on this chapter are managed in the BO as this is the only way to do system management.

3.7.1 *Users*

To access the BO, user accounts are used via login with username and password. The accounts are typically attributed to the administration but it is possible by using a permissions scheme to create accounts with view-only permissions to other employees which blocks the insertion and modification of data. Each Permission will be linked to an action in BO and only users who have this permission can access the respective menus or perform those actions. This permission scheme is explained on Section 3.7.3. Modifications on the data/DB are stored with the *timestamp* and the User who did it for history/monitoring purposes. The accounts are created by a User with user creation permissions.

3.7.2 *Authentication and Communication*

There are two entities that communicate with the WS: Devices and Users through the API. The server will have to interact with them in the most generic way possible and independently of OS that the requests come from. The Users authentication with *username/password* is related to this topic as its the way to link a person to a system account and all the following requests have to be linked to that account. The same with the Devices that won't have a *username/password* combination as that would be inconvenient and against the quick setup procedure that is required and they will authenticate themselves with its Device Code as it is a unique feature that can be used to differentiate devices.

3.7.3 *Roles and Permissions*

The API requests contain the identification of who is requesting the service and the server has to know if the one requesting is allowed to use that feature. As mentioned on section 3.7.1, each operation on the BO will be linked with a Permission. To create a Device, view a File details, update a User or delete a Configuration requires different permissions. Instead of creating a Permission for every possible operation, a Unix-like permission mode

(Srirengan, 1998) was chosen to help with these scenarios. On these systems, basically there are three permissions *Read/Write/Execute* (ignoring the classes *User/Group/Others*), on MDM it was changed to a CRUD style (*Create/Read/Update/Delete*). So, if a User has Read and Update on the Permission *Devices*, he can only view and update its info, but not delete it or create new devices.

A Role is a collection of Permissions. These Permissions have some operations blocked or allowed. A User can have multiple Roles and the merge of all its Permissions results in the operations that it can do. If two (or more) Roles have the same Permission, the result of this is the union of the state of its operations. For instance, if *Role A* has Read of Devices and *Role B* doesn't allow it, the result will be Read of Devices because if an operation is allowed in at least one Permission in a Role it will be possible to execute. As a User can have multiple Roles and those can be summed up, the Roles can be easily divided in smaller functionalities like Users Roles or Devices Roles. If an administrator grants the Devices Delete permission to an already existing role that had that operation blocked, and that permission isn't granted by any Devices' permission related roles, all the Users that have that role will now be allowed to delete devices. By removing and setting roles to users, the administration can easily change menus accesses without the need to change each user individually. The API endpoints are protected by setting the required permission that the user requesting must have in order to be able to do that WS action.

The endpoints that the Device connects to are also protected by a similar scheme focusing on the Device's State (Registered/Not Registered, Approved/Not Approved and Enabled/Disabled). The Registered state is always required except for the first registration step when the device sends its information to be enrolled. All the following requests demand Registered state and therefore it is omitted on the API endpoint permission definition. Take the following scenarios as example: to allow a device to check for updates, it has to be Approved and Enabled, but to set the Site after the device's registration, the device can be Not Approved but it has to be Enabled. If a device is Disabled, it will be blocked on any WS requests. The same occurs when a Device or a User is disabled or deleted.

3.8 DASHBOARD

The Dashboard is an easy to read module with a real-time user interface showing a graphical presentation of the current MDM system status. It is integrated on the BO serving the purpose to report some statistics gathered from the Agent's logs and the data on the system in order to help the administrators to take informed decisions. If it has a good information design, the dashboard will clearly communicate key information to users and makes supporting information easily accessible (Barr, 2010). With this feature the administrators can quickly spot negative situations by saving time, compared to go through all the reports one by one.

There are four reports and each one will have a moving tile on the BO's main menu showing a statistic for the last week:

Devices Distribution Lists the devices and the version that each one has installed. Includes the date of installation. It is possible to filter the devices by site/values. It has a moving tile on the main menu that will display the Top 3 Sites with more devices by OS.

Wrong Versions Matches the required versions demanded by the rules with the File Versions installed on the devices and shows only the versions that are missing. The purpose of this Dashboard is to report the devices with wrong versions. The tile will show by OS the devices that have at least one wrong version against the total of devices for that OS.

Installation Issues If the Agent couldn't perform an installation or Configuration it reports, in the logs, the cause. This dashboard helps to figure out problems and complete the previous dashboard with possible causes for a device not have a File Versions installed. The tile shows the Top 3 Files that have more installation errors by OS.

Log Errors All the logs sent by the Agents are stored and can be accessed in this dashboard. For more detailed logs the admin should check the log files that include the full trace. The Installation Issues dashboard is a filtered version of this one in order to provide a more specific filtering. The tile displays the Top 3 types of errors collected by the Agent grouped by OS.

4

IMPLEMENTATION

The development of a software application involves the use of technology according to the context and objectives of the project. Therefore, it is sometimes necessary to make some choices concerning the technologies to be used. In this section the main technologies are explained and the reasons why certain choices were made. All technological choices were made to ensure the interconnection between the entities.

Regarding the project characteristics, various alternatives and were taking into consideration about the technologies to be used. Comparative analyses on certain technologies were made in order to find the best solutions for certain aspects for the development. The decisions and conclusions are presented after.

The MDM project will manage all applications and their respective versions installed on all devices that use CS applications. Ideally, all future developed solutions will integrate with MDM, which allow them to be installed, configured and updated with newer versions as needed. It should have support for *Windows CE* due to the type of existing devices in customers, group devices by local or stores and through these groups assign versions of applications or special settings (such as language settings or application if a store or a warehouse). Group applications to be distributed in sets, typically due to dependencies. In terms of scalability, it must withstand tens of thousands of devices also facilitate registration of new devices in the system. These are the minimum requirements that a product MDM must have to be able to use.

4.1 SECURITY

The security is one of the most important part of a system that will be released to commercial use. As there are many attacks in the internet communications such as Man-in-the-middle. In this mater Cryptography takes a huge part in order to increase the security.

Regarding confidentiality on the communications, a certificate will be used on the WS to allow the usage of the *HTTPS/TLS* protocols.

Secure Sockets Layer (SSL) is a cryptographic protocol that enables secure communications over the Internet. SSL was developed by Netscape and released as SSL 2.0 in 1995 (Elgamal, 1995) and an improved SSL 3.0 was released in 1996 (Freier et al., 2011).

Transport Layer Security (TLS), defined in Dierks and Allen (1999), is the successor to SSL. The differences between TLS 1.0 and SSL 3.0 were significant enough that they did not work with each other. TLS 1.1 (Rescorla, 2006) and TLS 1.2 (Dierks, 2008) were later proposed. Modern browsers support TLS 1.2 [1]

Hypertext Transfer Protocol Secure (HTTPS) is an application-specific implementation that is a combination of the *Hypertext Transfer Protocol (HTTP)* with the SSL/TLS and is used to provide encrypted communication with a server. It is also important to ensure the server identification, a secured and bidirectional tunnel for data exchange between two hosts.

4.1.1 *API-Keys*

API-keys are a popular way of authenticating requests between web services. These are simple persistent access tokens generated once and appended to each HTTP request headers, which uniquely authenticates and authorises a user for that request. Their implementation can be insecure if the requests made over standard HTTP (and not SSL-encrypted HTTPS) are sent as clear text and the token is susceptible to snooping (Farrell, 2009). But it is a simple way to promote authentication between the WS and the devices/users and as the connection is secured by using HTTPS, this implementation can be used.

The Users and Devices are associated with an *API-KEY* when they are registered, that will represent them on all future WS operations. This key is generated on the User/Device creation and is used on the *Authentication* field on the HTTP request's header in a technique called *Basic Access Authentication*.

4.2 DATABASE

There are 2 Databases (DB) on the system, the Server's and the Device's. The Server DB is not a major concern in terms of choice, because there are not hardware reasons to choose one over another. But in terms of the device, that is a different scenario. *Windows CE* is a very limited OS and the simpler the DB engine is, the better. The choice has to be the SQlite, a file-based DB that supports multiple OS's.

[1] Webpage: `https://en.wikipedia.org/wiki/Template:TLS/SSL_support_history_of_web_browsers`

4.2.1 *PostgreSQL*

PostgreSQL is a powerful, open source object-relational database system. It has more than fifteen years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major OS's, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It has several data types, like INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including images or video. It has native programming interfaces for C/C++, Java, .Net, among others [2].

Like the majority of the Databases (DB), PostgreSQL has Transactions, Indexes, Triggers and Replication. One important feature is the Schema. In PostgreSQL, a schema act like a namespace, allowing objects of the same name to co-exist in the same database. Which means that there is no need to create a new DB just to have separated tables, a schema can do it.

4.2.2 *SQLite*

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is an embedded SQL database engine and unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. [3]

As this is also a cross-platform it will be used on the devices' DB implementation which is an important feature besides the nonexistent impact on the system that a usual DB service have.

4.3 BACKOFFICE

The BO will be a Web Page for administration purposes. It will also have a Dashboard to display information about the current system status. In order to help doing this module, some technologies had to be used and are explained in the following sections.

[2] Web Page: https://www.postgresql.org/about/

[3] Web Page: http://www.sqlite.org/about.html

It is a *Single-Page Application (SPA)* so it can be accessed on computers, cellphones or tablet browsers, being in that way easily accessible. All the necessary code (HTML, JavaScript, CSS and images) is fetched with a single page load and then the user can use it without the need of further downloads as the site page does not reload at any point in the process. Even the URI anchors change while navigating through the page, it is only for navigability purposes. The dynamic part is the data which is inquired to the WS which requires an internet connection. Therefore, new pages are capable of being generated without any interaction with a server. To create this SPA, I used *AngularJS*.

4.3.1 *AngularJS*

AngularJS[4] is a dynamic Web App Framework written in JavaScript focused on creating a single-page application and compatible with both desktop and mobile browsers. It is open-source and created by Google. It helps with the development of such applications by providing a framework for client-side Model-View-Controller (MVC) and Model-View-ViewModel (MVVM) architectures. It is distributed as a JavaScript file and can be added to a web page with a script tag. This framework works by first reading the HTML page which includes custom tag attributes and interpret them as directives to bind input or output parts of the page to a model that is represented by JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources. The data binding and dependency injection eliminate much of the code that would have to be written.

AngularJS extends HTML with *ng-directives*. For instance, the *ng-app* defines an AngularJS application and is obligatory. The *ng-model* directive binds the value of HTML controls like an input to the application data and the *ng-bind* directive binds the application data to the HTML view allowing that when the user types on the input field the value is written on the data without the need of extra code. it's called Data-binding, an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes.

HTML was invented as a declarative language for static documents, it is not focused on creating applications and these kind of frameworks are used as a workaround. To help with the design and the UI, Angular Material was used. Angular Material implements Google's Material Design Guidelines[5]. It provides a collection of UI components based on Material Design.

[4] Web Page: https://angularjs.org/

[5] Web Page: https://www.google.com/design/spec/material-design/introduction.html

4.3.2 *Dashboard Graphs - D3.js*

D3.js[6] is a JavaScript library to manipulate Data-Driven Documents and turning them into charts by using HTML, SVG, and CSS. It will be used on the Dashboard to generate graphs and charts based on the values returned from the WS.

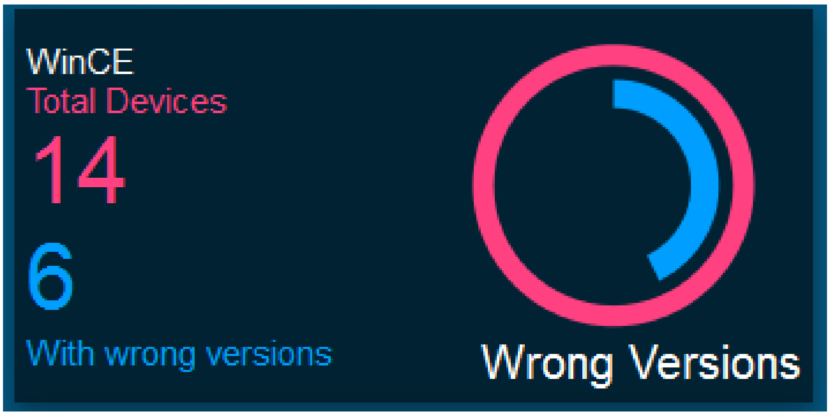


Figure 7.: D3.js usage on Section 3.8's Wrong Versions Dashboard

The final version can be seen on Figure 10 at Section A.2.

4.3.3 *Translations*

Writing multilingual programs is important when dealing with users that don't share the same language as the developers and it provide a better support for globalization. The *angular-gettext*[7] helped with the translation support as it has a tiny *footprint* on the integration with *Angular.JS*.

```
<h1 translate>Hello {{name}}, it's nice to meet you!</h1>
```

Listing 4.1: *angular-gettext* example in *Angular.JS*

The general programs that provide gettext functionalities follow some guidelines, *Poedit*[8] is a *.po* file editor that I used to help with the translation of texts provided from the texts extracted from the *angular-gettext*.

[6] Web Page: https://d3js.org/
[7] Web Page: https://angular-gettext.rocketeer.be/
[8] Web Page: http://poedit.net/

4.4 WEB SERVICE

A WS is a method of communication between devices via the World Wide Web. Through this technology, two devices with Internet access can communicate and exchange information between them. This solution allows the interaction between different applications with different languages to communicate via a universal format such as JSON or XML. Were considered two methods of communication for the architecture, REST and SOAP, and analysed their advantages and disadvantages in this context, in order to sustain the decision.

4.4.1 *REST vs SOAP*

REST and SOAP are two protocols of exchanging information between two entities. This chapter will focus on explaining both of them and then a comparison is made to choose one for the data communication.

4.4.1.1 *REST*

In 2000, Roy Fielding defined *Representational State Transfer (REST)* in his doctoral dissertation Fielding (2000) as an architectural style for designing distributed systems focusing on a system's resources rather than implementation details. REST is not strictly related to HTTP, but it is most commonly associated with it (Booth et al., 2004). If a system conforms to the constraints of REST, it can be called RESTful. The communication over HTTP with the same HTTP request methods (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers (Rodriguez, 2008). REST has this principles:

- Use HTTP methods.
- Be stateless.
- *eXtensible Markup Language (XML)* or *JavaScript Object Notation (JSON)* as data representation.
- Expose an URI structure as a uniform interface.

REST systems interface with external systems as web resources identified by URIs, for example `/devices/1`, which has the information regarding device number 1 can be fetched with `GET /devices/1`. The response would be a XML message:

```
<code>BB8F8CA112</code>
```

Listing 4.2: Response as XML

Or a JSON message:

```
{ code : "BB8F8CA112" }
```

Listing 4.3: Response as JSON

4.4.1.2 *SOAP*

Simple Object Access Protocol (SOAP) is an XML-based messaging protocol for exchanging structured information that can be used for simple one-way messaging but is particularly useful for the implementation of WS. Every operation the service provides is explicitly defined, along with the XML structure of the request and response for that operation. Each parameter is defined and bound to a type (e.g. integer or string). All of this is codified in the *Web Service Definition Language (WSDL)* and can is like a contract between the provider and the consumer of the service, a method signature for the WS.

It doesn't have a fixed transport protocol, HTTP is usually used but SMTP or FTP are also an alternative. Nor is it tied to any particular OS or programming language so the clients and servers in these dialogues can be running on any platform and written in any language as long as they can create and read SOAP messages.

Example of a method signature:

```
string getDevice ( int identifier );
```

Listing 4.4: SOAP method signature

The request that can be sent:

```
<?xml version="1.0"
      encoding="UTF-8"
      standalone="no" ?>
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <ns1:getDevice
   xmlns:ns1="urn:MDMService">
    <param1 xsi:type="xsd:int">1</param1>
  </ns1:getDevice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.5: SOAP request

And a response from the WS:

```
<?xml version="1.0"
      encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <xsd:getDeviceResponse
   xmlns:ns1="urn:MDMService"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <return xsi:type="xsd:string">BB8F8CA112</return>
  </xsd:getDeviceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.6: SOAP response

4.4.1.3 *Conclusion*

The focus of this decision centres on which WS best meets the project needs, rather than which protocol to use. The decision has to focus on the multi-OS support and compatibility with *Windows CE*. REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL -based interface design because it is easier to use. SOAP is definitely the heavyweight choice for Web service access, it provides the following advantages when compared to REST:

- Language, platform, and transport independent (REST requires use of HTTP)
- Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
- Standardized
- Provides significant pre-build extensibility in the form of the WS* standards
- Built-in error handling
- Automation when used with certain language products
- Security is defined by using WS-Security extensions, RESTful WS inherits security measures from the underlying transport layer used

REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

- No expensive tools require to interact with the Web service
- Efficient (SOAP uses XML for all messages, REST can use smaller message formats like JSON)
- Fast (no extensive processing required), SOAP defines much standards that need to strictly be followed for communication.
- Closer to other Web technologies in design philosophy
- JSON is more lightweight compared to XML, resulting on lesser bandwidth and resource usage

In the principles of RESTful interface design, JSON over HTTP is a powerful combination that allows different applications to easily connect, address, and consume resources. Exposing a system's resources through a RESTful API is a flexible way to provide different kinds of applications with data formatted in a standard way. As the project will deal with different OS's, it seems a good choice. It helps to meet integration requirements that are critical to building systems where data can be easily combined and to extend or build on a set of base, RESTful services into something much bigger.

As in this architecture is highly valued the efficiency due to high demand and the interoperability with the BO, REST is the most suitable solution for this project. The server will be a RESTful WS.

4.4.2 *ASP.NET Web API*

To build the RESTful WS I used the *ASP.NET Web API* framework using the .NET Framework [9]. It will ease the build of the HTTP services that will reach the mobile devices and the BO.

4.4.3 *Newtonsoft.Json*

As defined on the previous section, the communication data between the WS and the other entities will be formatted in JSON. To parse and create JSON data, the extension *Json.NET* [10] was used. It is a popular high-performance JSON framework for .NET and will parse the incoming requests body data and produce the responses directly from the objects.

[9] Web Page: http://www.asp.net/web-api

[10] Web Page: http://www.newtonsoft.com/json

4.4.4 *Npgsql*

Npgsql is the .NET data provider for *PostgreSQL* and will do the connections to the DB [11].

4.4.5 *PetaPoco*

With *Npgsql* the connection to the DB is made but to transform de data that came from there to usable objects with defined classes is a priority. This is called Object-Relational Mapping (ORM) and it was used the *PetaPoco* project [12].

4.4.6 *AutoMapper*

As the objects come from the DB and are easily mapped to the POCOs, it is required their mapping to other classes. For instance, following a request and response path, first a request comes through which has some properties and it should be mapped to a middle class. That middle class will be between the classes with the outside interactions (requests and results) and the ones regarding the DB.

As the logic layer deals with the request and is time to do some DB operations, it is mapped to a POCO class to use with the data provider. When the operation is done, the result can be fetched (if that's the case) and the POCO class is mapped to the middle class and if some data is required in the response, it is mapped to a result class. This process will facilitate with the DB tables because the POCO classes will have properties/fields with the same name as the columns, the request/response classes will have the properties with the desired fields. Like this, if a table's columns vary it is only required to change the POCO classes. The same for the requests and responses. It can limit the alterations on the logic layer so that changes on a layer affects only types in that layer.

Because mapping code is dull, the chosen library is the *AutoMapper* [13]. An object-object mapper that works by transforming an input object of one type into an output object of a different type.

```
Mapper.CreateMap<Site, SiteResult>().ForMember(m => m.id,   r => r.Id)
                                    .ForMember(m => m.Name, r => r.SiteName);
Site s = SiteLogic.GetSite(site_id);
SiteResult result = s.MapTo<SiteResult>();
```

Listing 4.7: AutoMapper example

[11] Web Page: http://www.npgsql.org/

[12] Web Page: http://www.toptensoftware.com/petapoco/

[13] Web Page: http://automapper.org/

4.4.7 *Apache log4net*

On Section 3.1.1 I explained why logging is important for later support and the *Apache log4net* [14] is a library that fits this job. It is a tool to output log statements to a variety of output targets that will focus on exporting errors and the history of requests.

4.4.8 *Obfuscation*

In software development, manual obfuscation is the deliberate act of creating obfuscated code so that is difficult for humans to understand. A defense against reverse engineering is obfuscation, a process that renders software unintelligible but still functional (Collberg and Thomborson, 2002). Obfuscate code is used to conceal its purpose as security through obscurity or its logic, in order to prevent tampering and prevent reverse engineering. I used *.NET Reactor*[15].

4.5 AGENT AND UPDATER

The advantages of Cross-Platform development were explained on Section 2.1.3 and *Xamarin* was chosen for the mobile implementation due to its characteristics. The agent and updater logics were all coded in *C#*.

4.5.1 *Preparing Cross-Platform*

The development didn't need the *Xamarin* platform for the *Windows CE* version, it was chosen as the desired platform for the future *Android* or *iOS* releases. The interfaces and the *Inversion of Control (IOC)* helped to ignore the OS specifics that the Agent will run because the logic should not behave differently depending on which OS it is currently on.

First, the containers are registered like this during the app start:

```
container.Register<IInstaller>(new CabInstaller(), Scope.Singleton);
```

Listing 4.8: Container Registry example on *Windows CE*

And on Android the `IInstaller` class will be different:

```
container.Register<IInstaller>(new ApkInstaller(), Scope.Singleton);
```

Listing 4.9: Container Registry example on *Andoid*

[14] Web Page: https://logging.apache.org/log4net/

[15] WebPage: http://www.eziriz.com/dotnet_reactor.htm

The logic just uses the interfaces to ignore the definitions and the *Windows CE* project will register the containers for each module (Listing 4.8). File manipulation, Navigator, Installers and PopupDialogs are examples of interfaces that have to be implemented accordingly to each OS. `IInstaller` interface is like the following listing:

```
public interface IInstaller {
    InstallerType TypeInstaller { get; }
    Task<ResultErrorPair> Install(string filePath);
    Task<ResultErrorPair> Uninstall(string filePath);
    Task<ResultErrorPair> Repair(string filePath);
    Task<ResultErrorPair> Execute(string filePath,
                                  string arguments,
                                  bool waitForExit);
}

public enum InstallerType {
    ApkInstaller,
    CabInstaller
}
```

Listing 4.10: IInstaller interface example

The `CabInstaller` implementation of `Install` is shortly represented in Listing 4.11:

```
public class CabInstaller : ICabInstaller {
 public Task<ResultErrorPair> Install(string filePath) {
  string errorMessage = null;

  var process = new Process();
  process.StartInfo.FileName = "wceload.exe";
  process.StartInfo.Arguments = System.String.Format(
                                 "/noaskdest /delete 0 /noui \"{0}\"",
                                 filePath.Trim());
  process.Start();
  process.WaitForExit();

  bool result = process.ExitCode == 0;
  if (!result)
    errorMessage = string.Format("Installation {0} error! ExitCode :{1}",
                                   filePath, process.ExitCode);

  return new ResultErrorPair() { result = result,
                                 errorMessage = errorMessage
                               }.Task();
 }
}
```

Listing 4.11: Implementation of Install method on CabInstaller

The missing link of all this is how the logic uses this modularity, take the following example:

```
IInstaller installer = Container.Resolve<IInstaller>();
IInstaller popup = Container.Resolve<IPopupDialog>();
IInstaller log = Container.Resolve<ILogger>();

foreach(var fv in fileversionlist){
  if(!installer.Install(fv.FilePath)){
    popup.ShowAlert("Installation failed");
    log.Error(fv.toErrorString());
    error = true;
    break;
  }
}
```

Listing 4.12: Installation Logic example

The three containers are resolved and if they were properly registered on the application launch they are ready to be called. For each file version to install in the list, call the `Install` method from the unknown installer (the logic does not need to know its definition) and if it returned true then proceed to the next item on the list. If an error ocurred, use a Popup Dialog message to alert the user via *GUI* and log the error.

4.5.2 *Android*

The *Windows CE* development phase, including the server and BO, finished with a working version that fulfilled the requirements and was ready for a CS client PoC. But the project went with so much detail for the future support of other mobile OS's that with the current state, somehow I felt that it was uncompleted. And what better way to prove that the planing of that multi-platform solution than actually implement it in other OS. The *Android* version was completed in a shorter amount of time, comparing to the *Windows CE*, as it was expected.

There was not much to change, the major implementations were: create the `ApkInstaller`, use the *Java* IO File handling, create the new views for all the controls and forms and implement the new Navigator. The Navigator was difficult to port, because the way *Android* manages the views involves using *Context* and *Activities* which is different from the `form.Show()`. The other tough section was the installer because the interface of Listing 4.10 demands it to be synchronous. The way *Android* does installations is with *Intents* that are launched and the *Activity* that requested it has to wait for the result of it. But the controller/logic is not an *Activity*, so the *ApkInstaller* has to be one and then it can launch the *Intent* for installation like Listing 4.13.

```
Intent intent = new Intent(Intent.ActionInstallPackage);
intent.SetDataAndType(Uri.FromFile(apkfilepath),
                      "application/vnd.android.package-archive");
intent.SetFlags(ActivityFlags.NewTask);
intent.PutExtra(Intent.ExtraReturnResult, true); // Return result code
_Activity.StartActivityForResult(intent, INSTALL_REQUEST_CODE);
```

Listing 4.13: Apk Install example

But the problem is back again, it is required to wait for the *Activity* to close and for that you have to be an *Activity* too. The solution for this was to use C#'s *TaskCompletionSource* and *delegate* to catch the `OnActivityResult` and wait for that to return the result of the installation. Other situation with the *Android* is that to request an installation, the user has to accept it unlike the *Windows CE* installations of *.cab*. It means that the flow of updating will have to be accepted by the device's user and the auto-installations are not possible, unless the device has root and that should not be a valid restriction to request the client to have.

Besides these issues with the port, the result was a stable version for *Android* but with a major feature lacking, Configurations. As there are no Registry values it wasn't implemented. Some similarities can be found with *Shared Preferences* that work like the Registry and can be private or public for each application. No logic was changed to support this new version, proving that the work done before to support this new OS was a success.

4.6 SYSTEM REVIEW

In this chapter some libraries and frameworks for this architecture were presented as how I used them for the implementation. By the set of listed characteristics, it's possible to conclude that this architecture corresponds to a *Service-Oriented Architecture (SOA)*. This type of architectures consists in providing features in the form of services to its users.

Relating to the WS and the BO, this *cloud* part of the system will be hosted on the *Amazon Web Services (AWS)*:

EC2 Elastic Compute Cloud hosts the WS as it is AWS offering of the most fundamental piece of cloud computing: A virtual private server. These 'instances' and can run most Linux, BSD, and Windows operating systems. A *Windows Server 2012 R2* instance is the host for the server with a load balancer. The load balancer will divide the server requests for different instances to split the workload providing scalability.

S3 Simple Storage Service is the AWS' standard cloud storage service, offering file storage of arbitrary numbers of files of almost any size, from 0 to 5 TB. Items, or objects, are placed into named *buckets* stored with names which are usually called keys. The main content is the value. This storage service will be used to save the devices logs, file versions and the BO.

Cloudfront is a Content Delivery Network (CDN). It gives web application developers an easy way to distribute content with low latency and high data transfer speeds. In this service, the BO will be hosted as it is a SPA and it doesn't need a server to run, this is a viable solution as it is only downloaded.

RDS Relational Dabase Service is a managed relational database service to deploy and scale databases more easily. It supports *Amazon Aurora*, *Oracle*, *Microsoft SQL Server*, *PostgreSQL*, *MySQL* and *MariaDB*. The *PostgreSQL* DB runs on a RDS instance.

Amazon VPC Amazon Virtual Private Cloud provides an isolated section of the AWS cloud where is possible to launch services in a defined virtual network with a complete control over the virtual networking environment, including a selection of *IP* address range. By creating this security groups, the RDS DB is only accessible by the server's EC2 instance (Windows Server 2012 R2, Intel Xeon E5-2670 v2 @ 2.50 GHz, 1GB RAM; 29.6 GB); the BO and the WS can access the S3 file storage. The file transfer is made through the server and not directly to the S3.

4.6.1 *Cloud Proposal*

Based on the previous definitions of Agent/Updater, BO and WS, the system's architecture will be cloud-based as the following figure illustrates:

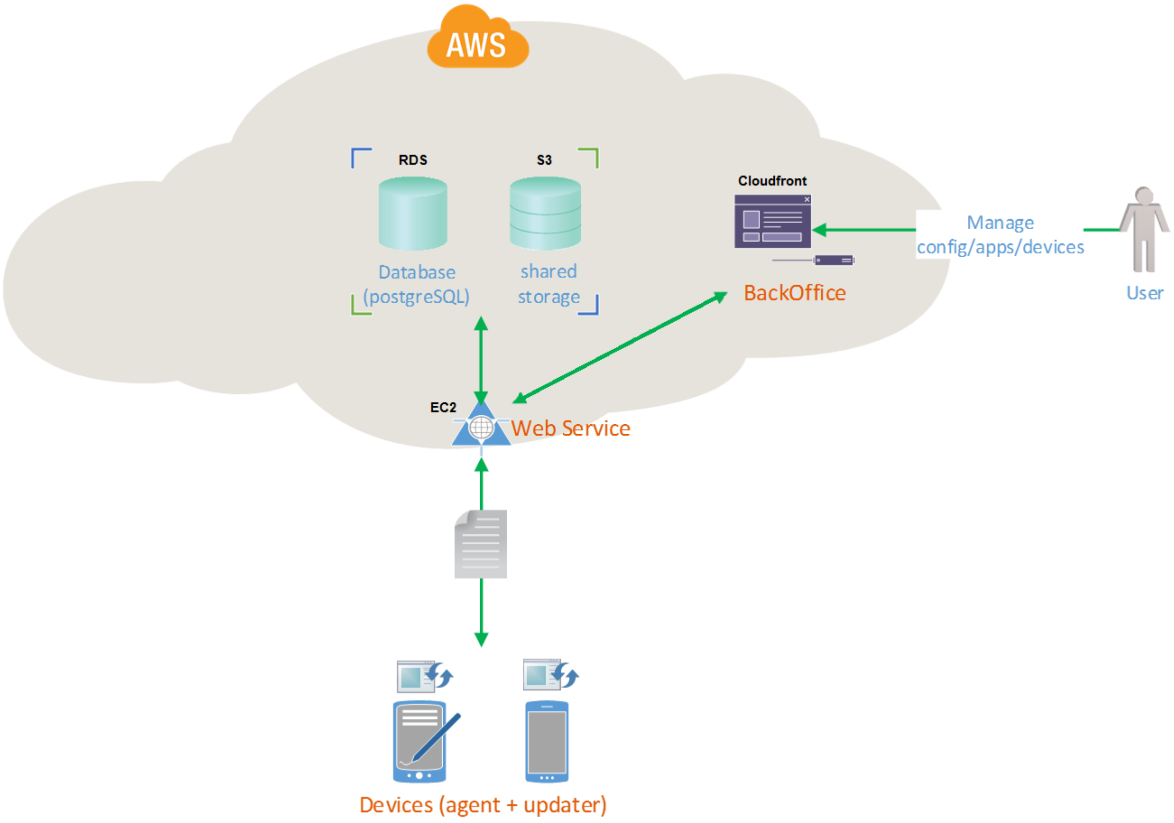


Figure 8.: Cloud Architecture

4.6.2 *Security Analysis*

Software security is an idea implemented to protect software against malicious attack and other hacker risks so that the software continues to function correctly under such potential dangers (McGraw, 2004). Security is necessary to provide authentication, integrity and availability. Any compromise to it makes a software insecure, susceptible to be attacked to steal information, monitor content, introduce vulnerabilities and damage the behaviour of software.

In the scope of the Device, the connection to the WS is the main concern. The server uses HTTPS/TLS and the device connects to it to ensure privacy and server authentication. To help to identify the device that made the request, an *API-KEY* is given to each Device during its registration (Section 3.1.2). As the *API-KEY* is unique by device code, if a malicious device somehow can change its device code, it can get other *API-KEY* and make itself logged in as that device. The same can happen if that key is stolen. For this last issue, the key should have a short expiration date to ensure a new device registration/login. So it is possible to disguise as other device in terms of communication but that may not be major risk at all. The device only receives updates and sends its state to the *ws*, the only changes it can make are not a security risk unless receiving an update that shouldn't be available to a certain set of devices.

The BO interacts with the API the same way as the devices but its *API-KEY* is retrieved using the login credentials for the user. Each module has its permissions and only the users with the allowed roles can access them.

On server-side there are more variables to take in consideration. The DB can only connect to the *EC2* instances and therefore is not publicly available. The *S3* bucket where the BO is hosted is made public read-only by the *Cloudfront* and the bucket for the device logs and file versions is like the DB, it's private to the server. The access to the server instance, where the server program is obfuscated, is only allowed for certain ranges of *IPs* restricted to internal computers as the access to it would allow connection to the other entities. These restrictions are configured with the AWS *VPC* security groups. The server instance is publicly available through the balancer to the server port. With coordination with the CS client, the *IP* distribution for the devices would be important to increase security by configuring the *IP* range in AWS. But this configurations are external to the MDM product itself and should be discussed with the network responsible on each customer. The WS components (*.dll*'s) are obfuscated to help protect the program content contained within software by making reverse-engineering difficult.

5

CONCLUSION

The problem that CS had with remote application management was not a new one, it is a global thing and there are a lot of software solutions that tackle it but they didn't fit the company's requirements. An internal MDM project was proposed for this dissertation and it consisted mainly on providing the capability to remotely update applications on the client's PDAs as for this tasks, some employee had to do it on the site. Because it has to execute on the client's PDAs, this software was developed for them, namely *Windows CE* but with support for the later implementation on other OS.

The requirements and planing phase were the first and took more time than I expected, but it was worth it, because the specification became more robust. The logic layer was complicated because it had to be developed with a focus on the generic approach and leave the OS specifics to other modules. After the *Windows CE* release, there was time for the *Android* version to start and apply all the effort made before to ensure a rapid development, which turned out to be true. This *Android* version is visually similar to the first release and behaves equal, but it was developed in a shorter amount of time. The cross-platform proved essential as well for this objective that was not set as a priority but as an extra feature for the duration of this dissertation.

This dissertation concluded with a stable version and currently on a client helping mainly the CS retail solution to be updated for now.

5.1 ONGOING WORK

By the date this dissertation concluded there was some work in progress, specifically monitoring the performance as the MDM is being used in an increasing number of devices when new stores are being added to the CS RFID solutions. Data is being recorded in terms of usage, like the peak hours of updates, maximum and average load the system reaches and study when new nodes are required to be added to the balancer.

5.2 FUTURE WORK

This project can be expanded with more features as it is not a complete MDM system by comparing it with any product on Section 2.1. Besides it is not intended to compete with any of those, it should be more robust to face the future problems that haven't occurred today. I will detail some improvements that would make it a more complete MDM solution.

The updating process of an application requires that it must be terminated before starting that process. As the Agent will update other applications that are not in its control, there has to be coordination between the Agent and the application to be updated so that it is not closed when it should not. For instance, to terminate an inventory or a disk defragmentation application when it is running, because there is an update and the Agent has decided to install it. Only the application knows when it is the best time to do so, when it is in idle mode for example. So it should be created a module to be included by the applications that want to be updated, for them to communicate with the Agent indicating that they can be updated if there is the need (update) for it. This integration would also be used for another purpose, who have this module knows how to communicate with the Agent and if it does not find this application in its installation logs, the Agent can conclude that it was a manual installation. Right now it can only detect if the Agent was a manual installation if it was initiated with clean DB and registries. Not all programs should include this module, only the Main Applications (Section 3.2). On *Android* version, *Content Providers* could be used as this is the main purpose of them.

Another functionality that should be implemented is the possibility to uninstall applications. As the usual operations a device does are installations of newer versions and as these ones (installers) take care of the update phase, this MDM version did not have the need to contemplate this feature. Because the notion of File Version is not just about installers, the system will perform deletion of files as well. Regarding a *cab* or *msi*, it is possible to uninstall them if the *cab*/*msi* is still present. On *Android* the uninstall process is way easier for instance. A new file type or similar, should come up so that after the installation, the Agent could copy the installer to a designated path and execute a task list.

Operations like uninstallation can be added and grouped in Remote Tasks, a set of tasks that a device must execute. Besides uninstallation, a task list can have file deletion and manipulation (move, copy), zip/unzip files that are not on the MDM scope, request an Agent DB dump to be sent to the WS, remote wipe and app disable (*Android*). This task list could even execute periodically or just once. Push Notifications on mobile Agent implementations that can trigger the auto-update or inform the device status has changed. They could be used to trigger devices tasks without the need to wait for the next sync request.

The Groups hierarchy is being locked until all the groups that are on a node are deleted and that node can only be deleted if it doesn't have a sub-node. Only then it is possible to remove a node. This can be a problem if there is an already created structure and it is required to move a node to other position because this is not supported. Some consideration has to be made to allow this kind of operation because if a node changes its position on the graph, the values would need to be changed provoking alterations on the already created rules and sites. Data may become inconsistent and wrong. Besides that, it could have a "migration" mode to proceed with these changes and ask the user what the system should do in cases of conflicts, like decide what parent value should a group value now have.

Configurations can be improved as the only support they have are *Windows* Registry Values. *VPN* or *Wi-Fi* networks setup would be extremely helpful as the stores usually have private networks and the devices have to connect to them. These networks may need to use certificates and the Agent could provide it to them. During the initial Agent setup process where it reads from the config file and tries to connect to the WS, the Agent would configure the network if its definitions were defined there.

The Dashboard should be able to export the displayed data to a *.csv* file, so that the administrators can use it for other purposes like integrations and history. More reports are expected to be added and related to server performances or agents installations duration. The logs contain so much data the Dashboard can easily be expanded. The number of days that the moving tiles from the Dashboard collects to display, the OS available, enable or disable the auto-approval of devices when they set the first site in the registration phase, allow or disallow the re-registration of devices that were deleted back to the system, file types or BO visual settings like colors, those settings should be configurable on the BO and without the need to do that directly on the DB or in the source code. So that the administrator has more configurations at his control and leave the platform more personalizable.

An intensive stress test has to be performed to analyse the system performance, both on the server and on the DB. With the results, some changes would occur like adding some indexes, remove or add new relations on the tables and study replication just on the DB. About the server-side, provide an asynchronous module to free some computation on the webserver and facilitate scalability. For this matter, it could be used the AWS *SQS* for message queues and *Lambda* functions for chrono events to trigger jobs that would be executed by the workers. These workers execute independently and the more are added to the system, the system capacity to handle asynchronous jobs may increase until a certain point. Regarding performance, a cache is an optional choice to increase data retrieval performance on requests. *Redis* or *ElastiCache* are examples of these services. This stress test should have been executed during the development phase but it was not an obligatory requirement and the MDM usage on the client that has currently the solution allows for it to be done later because the performance is not an issue right now and the system is comfortable with the current status.

In this section I explained some features that this solution is missing or lacks improvement, resulting in work that can be done to make it more complete.

BIBLIOGRAPHY

AirWatch. *Enterprise Mobility Best Practices: MDM, Containerization or Both?* AirWatch, 2016. URL http://triangle.ie/wp-content/uploads/2016/09/airwatch-whitepaper-mdm-containerization-or-both.pdf.

Stacey Barr. *7 Small Business Dashboard Design Dos and Donťs*. Stacey Barr Pty Ltd, 2010.

David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. 2004.

Cisco. *Cisco Study: IT Saying Yes to BYOD*. PhD thesis, 2012. URL http://newsroom.cisco.com/release/854754/Cisco-Study-IT-Saying-YesTo-BYOD.

Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Transactions on software engineering*, 28 (8):735–746, 2002.

Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

Tim Dierks and Christopher Allen. The tls protocol version 1.0. 1999.

James Donald. Improved portability of shared libraries, 2003.

Edd Dumbill and Niel M Bornstein. *MONO: A developer's notebook*. " O'Reilly Media, Inc.", 2004.

Taher Elgamal. The secure sockets layer protocol (ssl). In *agenda for the Danvers IETF meeting*, pages 1–5, 1995.

Stephen Farrell. Api keys to the kingdom. *IEEE Internet Computing*, 13(undefined):91–93, 2009. ISSN 1089-7801. doi: doi.ieeecomputersociety.org/10.1109/MIC.2009.100.

Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. URL http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Michael Finneran. *BYOD Requires Mobile Device Management - To keep "BYOD" from translating to "bring your own disaster," IT needs MDM*. 2011. URL http://www.informationweek.com/mobile/byod-requires-mobile-device-management/d/d-id/1097576.

Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0. 2011.

Bob Hayes and Kathleen Kotwica. *Bring your own device (BYOD) to work: Trend report*. Newnes, 2013.

ISO. Iec 23271: 2012: Information technology–common language infrastructure (cli). *International Organization for Standardization, Geneva, Switzerland*, 3, 2012.

Michael Jang. *Linux annoyances for geeks*. O'Reilly, Beijing Sebastopol, CA, 2006. ISBN 9780596552244.

Candar Kaan. Mono. net goes linux. 2007.

Jan Kietzmann, Kirk Plangger, Ben Eaton, Kerstin Heilgenberg, Leyland Pitt, and Pierre Berthon. Mobility at work: A typology of mobile communities of practice and contextual ambidexterity. *The Journal of Strategic Information Systems*, 22(4):282 – 297, 2013. ISSN 0963-8687. doi: http://dx.doi.org/10.1016/j.jsis.2013.03.003. URL http://www.sciencedirect.com/science/article/pii/S0963868713000395.

Jason King and Mark Easton. *Cross-platform. NET Development: Using Mono, Portable. NET, and Microsoft. NET*. Apress, 2004.

Mark Mamone. *Practical Mono*. Apress, 2006.

Gary McGraw. Software security. *IEEE Security & Privacy*, 2(2):80–83, 2004.

Eric Rescorla. The transport layer security (tls) protocol version 1.1. *Transport*, 2006.

Keunwoo Rhee, Woongryul Jeon, and Dongho Won. Security requirements of a mobile device management system. *International Journal of Security and Its Applications*, 6(2):353–358, 2012.

Borgman & Rice. *The use of computer-monitored data in information science*. Journal of the American Society for Information Science, 1983. ISBN 0-201-85469-4.

Alex Rodriguez. Restful web services: The basics. *IBM developerWorks*, 2008.

Hans-Jürgen Schönig and Ewald Geschwinde. *Mono kick start*. Sams Publishing, 2004.

K Srirengan. Understanding unix. page 58, 1998.

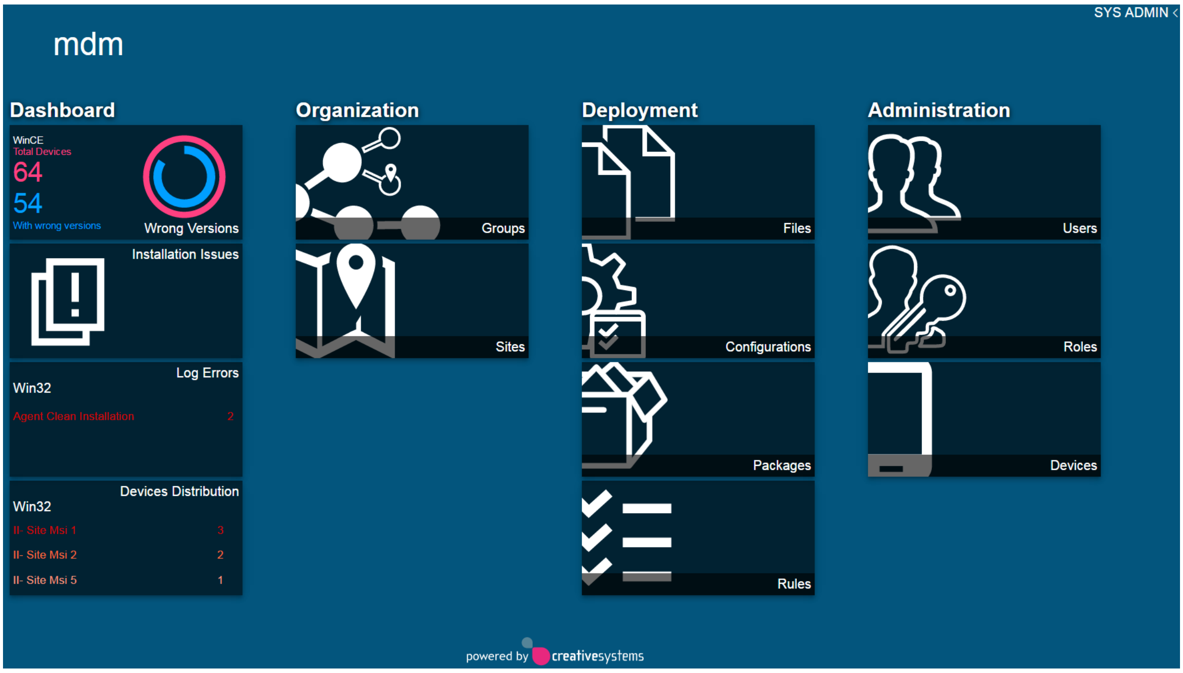
J. Tsang. *The Ten Commandments of Bring Your Own Device (BYOD)*. IBM Corporation, 2016. URL https://www.maas360.com/lp/eb-ten-commandments/.

A

BACKOFFICE SCREENSHOTS

A.1 MAIN MENU

The Main Menu is divided in 4 sections: Dashboard, Organization, Deployment and Administration. Each one contains tiles that will access the menus. The Dashboard tiles are dynamic, cycling the OS's every 6 seconds and updating its info every minute.



WinCE
Total Devices
64
54
With wrong versions
Wrong Versions

Installation Issues

Log Errors
Win32
Agent Clean Installation 2

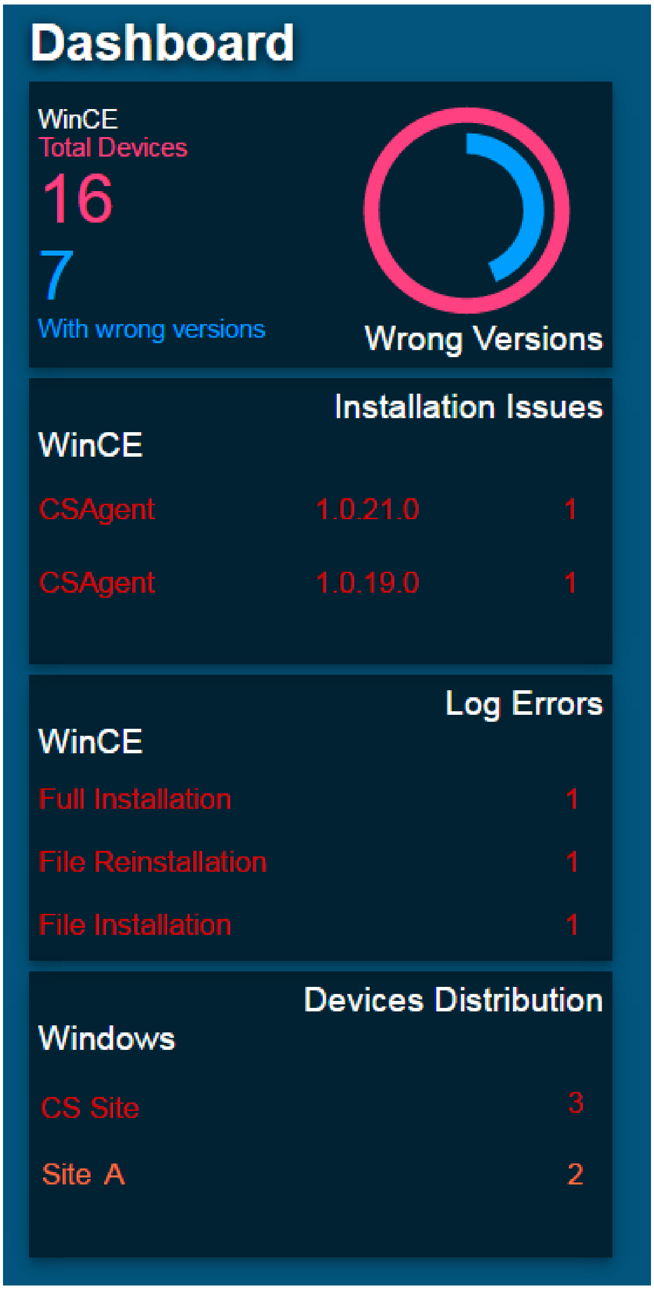
Devices Distribution
Win32

II- Site Msi 1	3
II- Site Msi 2	2
II- Site Msi 5	1

Figure 9.: Main Menu

A.2 Dashboard

The four Dashboard tiles.



Dashboard

WinCE
Total Devices
16
7
With wrong versions

Wrong Versions

Installation Issues

WinCE

CSAgent	1.0.21.0	1
CSAgent	1.0.19.0	1

Log Errors

WinCE

Full Installation	1
File Reinstallation	1
File Installation	1

Devices Distribution

Windows

CS Site	3
Site A	2

Figure 10.: Dashboard Tiles

A.3 GROUPS GRAPH REPRESENTATION

In Figure 11 there is the representation on the BO of the example of Groups used in Section 3.4. The *root* node will link all root groups, the purpose of this is to visually have a better perception of the graph and to facilitate the addition of new root groups. The grey content and yellow borders are the root groups and the white with blue borders are the sub-groups that will continue to the right.

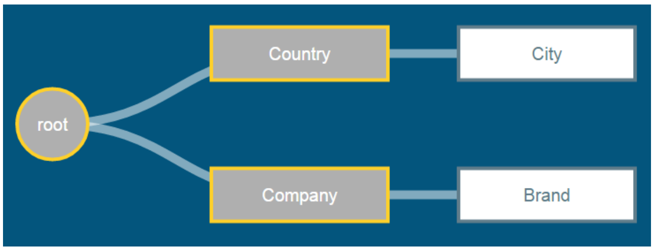
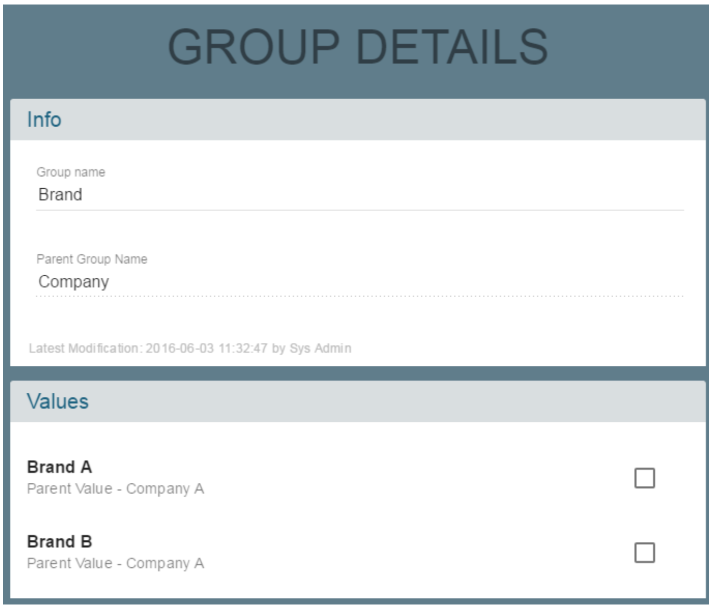


Figure 11.: Groups graph representation

A.4 GROUP DETAILS

If the *Brand* node was selected on the previous image, its details page would be like Figure 12. You can see its name and parent group, a timestamp of the last modification and who did it and the group values that this group has. The values also indicate its parent group value. It's possible to modify its values and name but not change the parent group as changing the hierarchy is not possible.



GROUP DETAILS

Info

Group name
Brand

Parent Group Name
Company

Latest Modification: 2016-06-03 11:32:47 by Sys Admin

Values

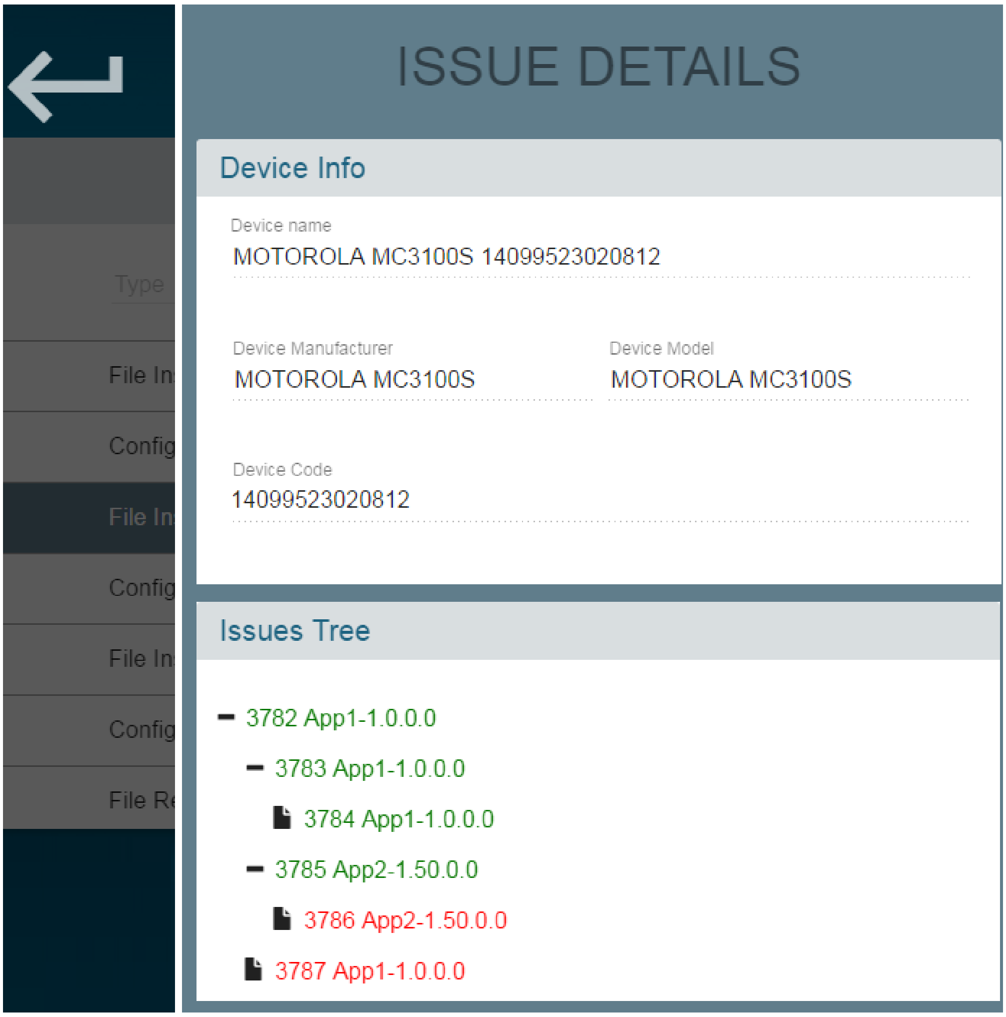
Brand A
Parent Value - Company A □

Brand B
Parent Value - Company A □

Figure 12.: Group Details of Brand

A.5 ISSUES EXAMPLE

Figure 13 is an issue tree example with an error occurring during a package installation. On the top there is the information about the device that produced this log and below there is the issue tree with the relation of logs (explained on Section 3.1.1). The logs with a successful operation are in green and in red the ones with errors. The name on the tree is composed by its *log_id*, file and file version. The first item says *App1* because that is the package's main application. The icon is to show that it is about a file, the other case is a configuration icon. If clicked, the log's full details are shown.



ISSUE DETAILS

Device Info

Device name
MOTOROLA MC3100S 14099523020812

Device Manufacturer
MOTOROLA MC3100S

Device Model
MOTOROLA MC3100S

Device Code
14099523020812

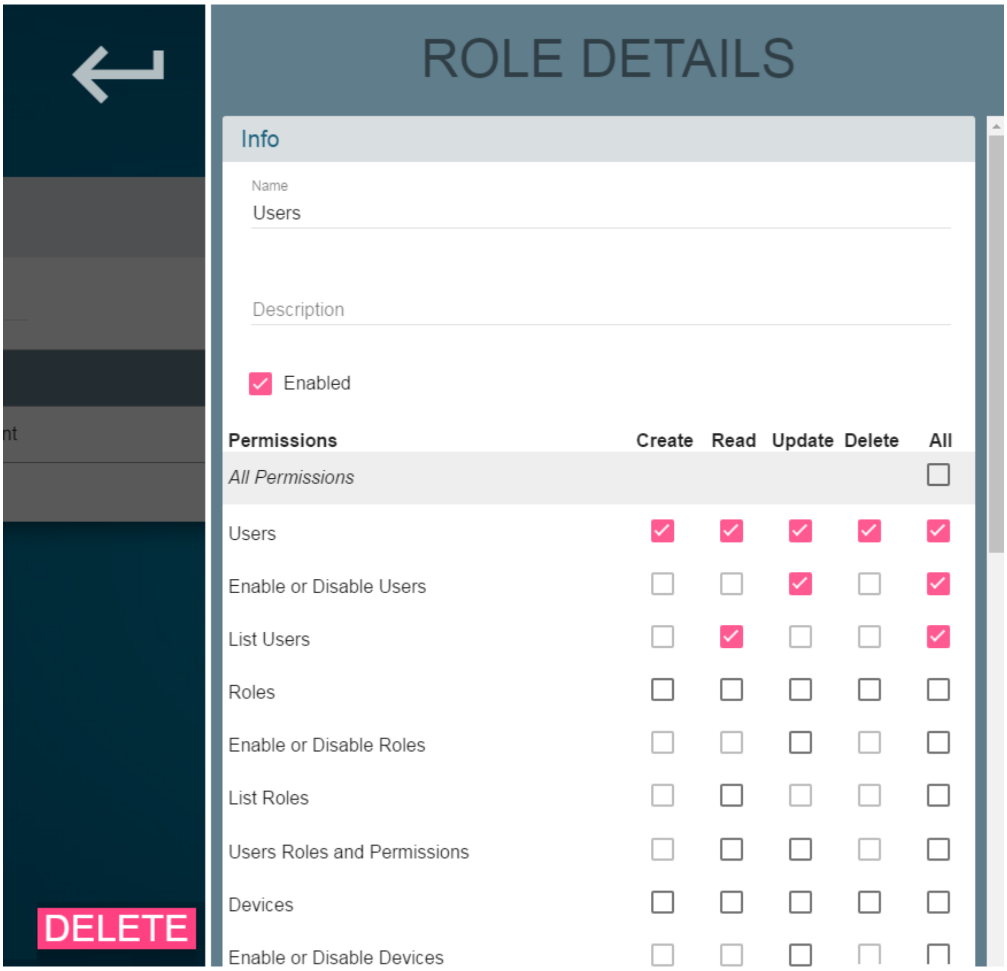
Issues Tree

- 3782 App1-1.0.0.0
  - 3783 App1-1.0.0.0
    - 3784 App1-1.0.0.0
  - 3785 App2-1.50.0.0
    - 3786 App2-1.50.0.0
  - 3787 App1-1.0.0.0

Figure 13.: Issues example

A.6 ROLE DETAILS

This is a role details page of a role called '*Users*' that only has permissions regarding user's accounts actions. It has a name, a description and an enabled state. Below there is a list of all the available permissions using the rules described on Section 3.7.3. Note that the permission *Users Roles and Permissions* isn't selected, only with that permission it is possible to edit other users permissions to override the roles.



ROLE DETAILS

Info

Name: Users

Description

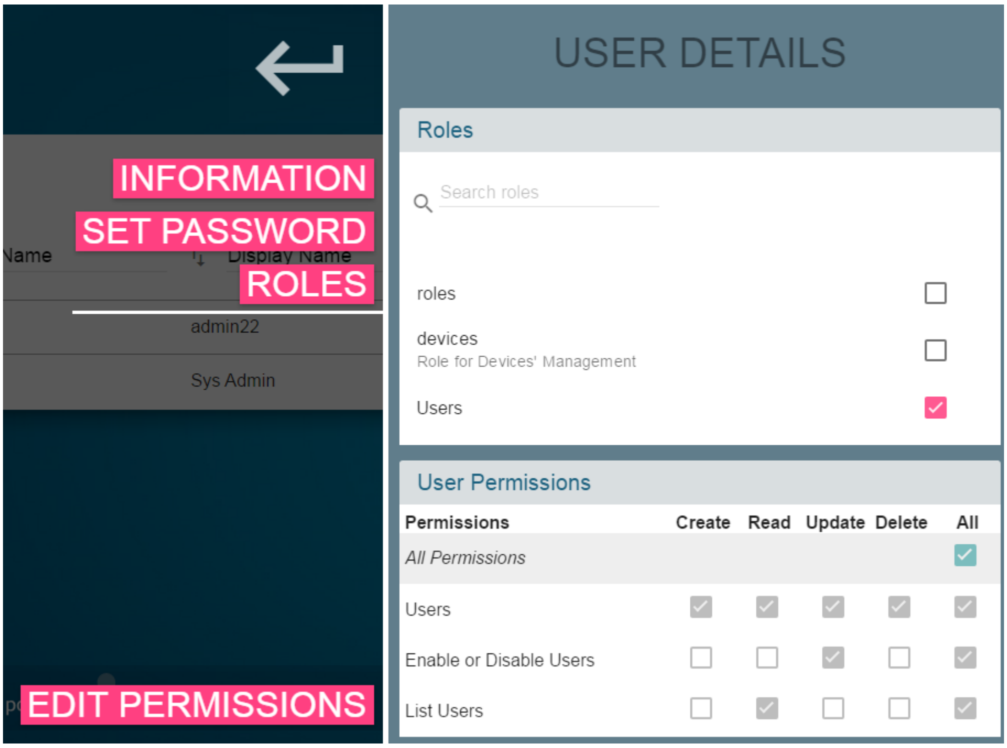
☑ Enabled

Permissions	Create	Read	Update	Delete	All
All Permissions					□
Users	☑	☑	☑	☑	☑
Enable or Disable Users	□	□	☑	□	☑
List Users	□	☑	□	□	☑
Roles	□	□	□	□	□
Enable or Disable Roles	□	□	□	□	□
List Roles	□	□	□	□	□
Users Roles and Permissions	□	□	□	□	□
Devices	□	□	□	□	□
Enable or Disable Devices	□	□	□	□	□

Figure 14.: Users only role example

A.7 USER ROLE DETAILS

The permissions that a user has with the example role from the previous section A.6. He only has that role selected and therefore only its permissions are available. If the administrator adds the *devices* role, this user would be granted with those too and the available permissions below would include them. By default the user permission edition is locked, to unlock it and proceed with changes the user has to click on *Edit Permissions* button on the bottom of the options side menu.



Roles

roles ☐

devices
Role for Devices' Management ☐

Users ☑

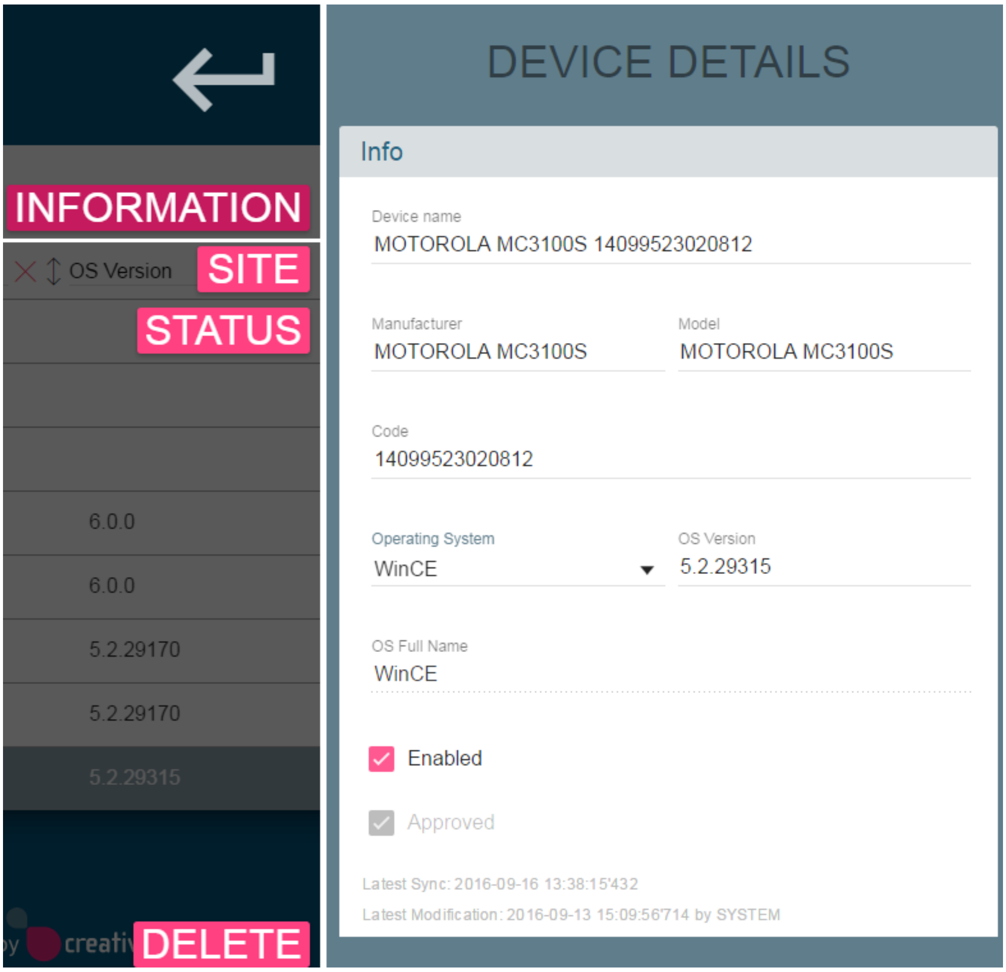
User Permissions

Permissions	Create	Read	Update	Delete	All
All Permissions					☑
Users	☑	☑	☑	☑	☑
Enable or Disable Users	☐	☐	☑	☐	☑
List Users	☐	☑	☐	☐	☑

Figure 15.: User account displaying the available permissions

A.8 DEVICE DETAILS

A details page of a device with *WindowsCE* OS. It includes its name, manufacturer, model, device code, OS and version, OS full name, enabled and approved states, timestamps about its last synchronization and modification. It is only possible to disable this device, besides the deletion because the only way for it to become disapproved is to request a site change on the Agent.



DEVICE DETAILS

Info

Device name
MOTOROLA MC3100S 14099523020812

Manufacturer
MOTOROLA MC3100S

Model
MOTOROLA MC3100S

Code
14099523020812

Operating System
WinCE

OS Version
5.2.29315

OS Full Name
WinCE

☑ Enabled

☑ Approved

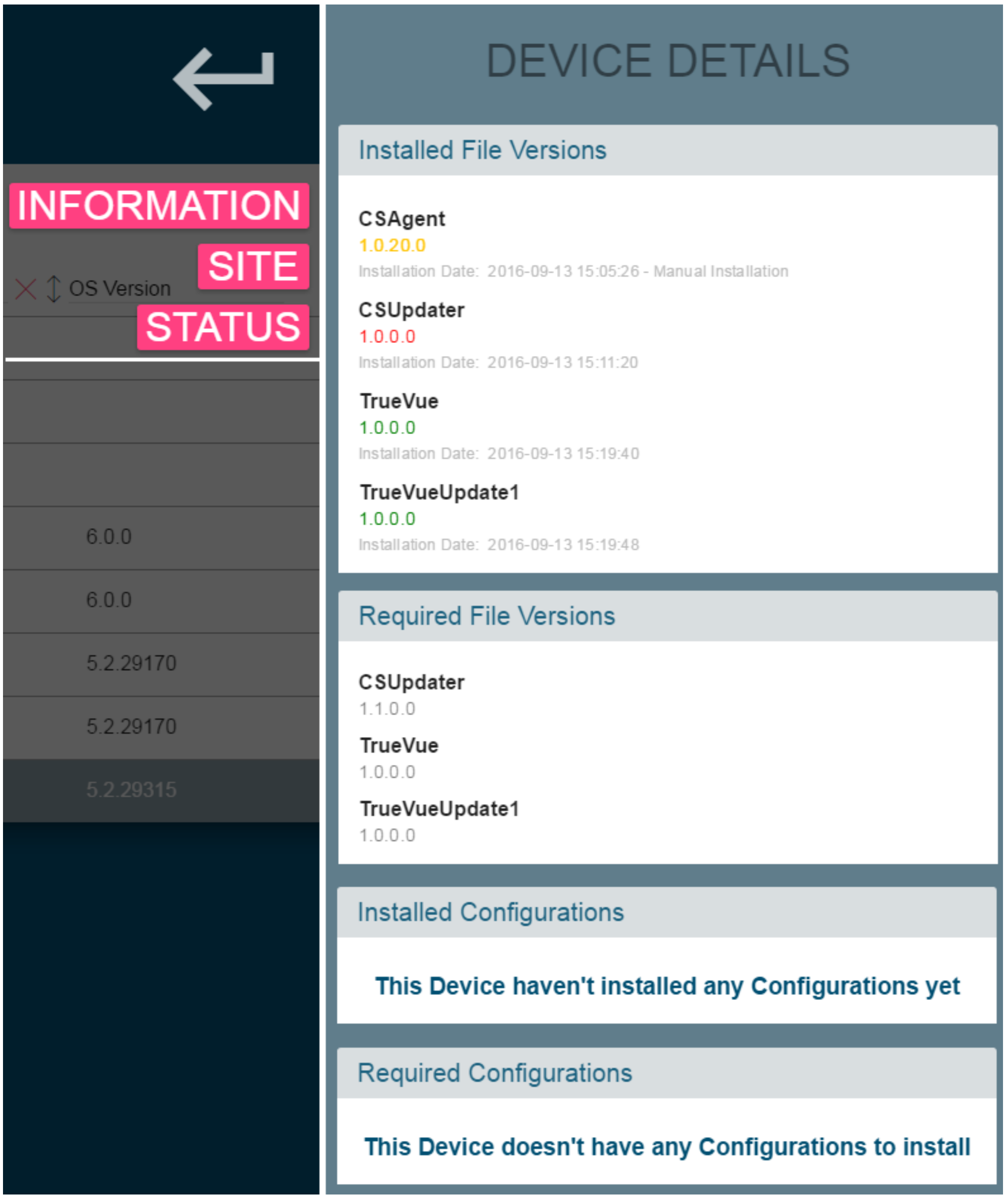
Latest Sync: 2016-09-16 13:38:15'432

Latest Modification: 2016-09-13 15:09:56'714 by SYSTEM

Figure 16.: Device Details

A.9 DEVICE STATUS

The following image shows that the device has correctly installed the *TrueVue* and *TrueVue-Update1* applications, *CSAgent* is not demanded by any rule but it was installed manually and the MDM detected it as so, and the *CSUpdater* is on version *1.0.0.0* and should have the version *1.1.0.0*. Along with the versions installed, it is displayed the installation date. This device haven't configured anything as it was not demanded to do any.



DEVICE DETAILS

Installed File Versions

CSAgent
1.0.20.0
Installation Date: 2016-09-13 15:05:26 - Manual Installation

CSUpdater
1.0.0.0
Installation Date: 2016-09-13 15:11:20

TrueVue
1.0.0.0
Installation Date: 2016-09-13 15:19:40

TrueVueUpdate1
1.0.0.0
Installation Date: 2016-09-13 15:19:48

Required File Versions

CSUpdater
1.1.0.0

TrueVue
1.0.0.0

TrueVueUpdate1
1.0.0.0

Installed Configurations

This Device haven't installed any Configurations yet

Required Configurations

This Device doesn't have any Configurations to install

Figure 17.: Device Status showing what it has installed

B

AGENT SCREENSHOTS

This chapter contains screenshots of the Agent *Windows CE* version.

B.1 AGENT MAIN MENU

The Main Menu contains 5 menus, Device Information, Configurations, Update Applications, Restore Applications and Synchronize. It has on the top left an exit button to close the program.

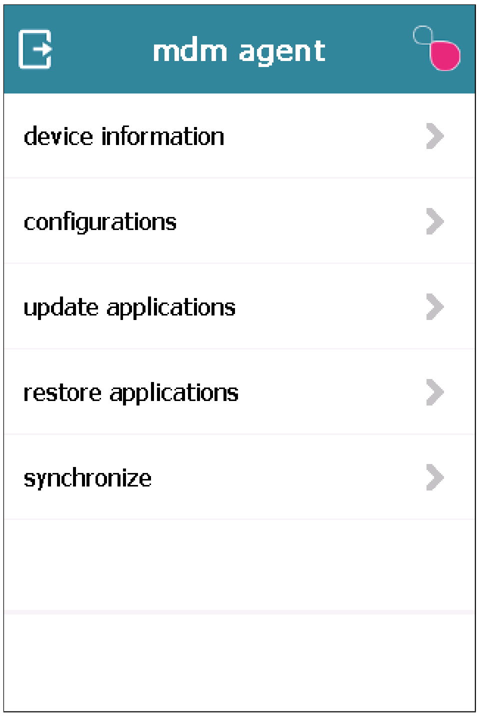
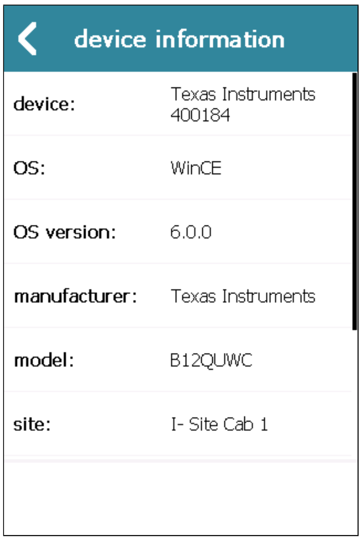


Figure 18.: Agent Main Menu

B.2 DEVICE INFORMATION

Is a scrollable menu showing the device's information similar to the details page on BO (Figure 16).



device: Texas Instruments 400184

OS: WinCE

OS version: 6.0.0

manufacturer: Texas Instruments

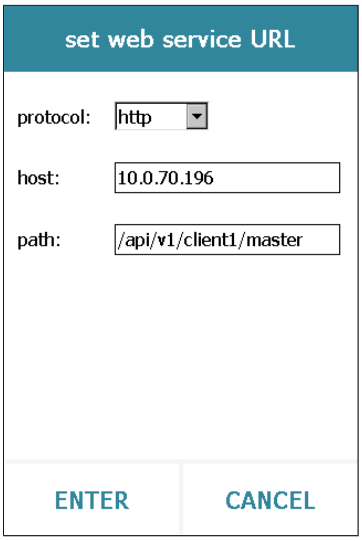
model: B12QUWC

site: I- Site Cab 1

Figure 19.: Device Information Menu

B.3 SET WS URL

This menu appears if the settings on the default config file lead to an connection error or if that file is not present. This menu can also be later accessed in the Configuration menu.



protocol: http

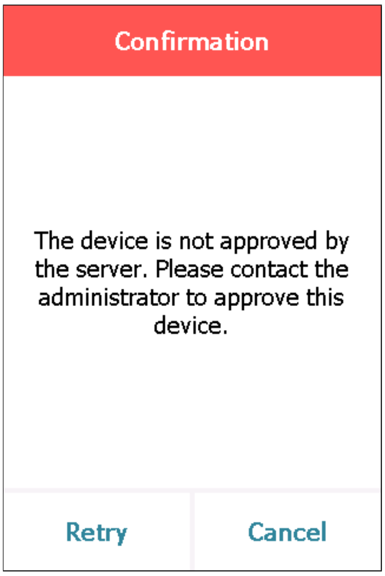
host: 10.0.70.196

path: /api/v1/client1/master

Figure 20.: Set WS connection

B.4 NOT APPROVED ERROR POPUP

When the device is not approved and is requesting an API action, this popup message is displayed. The user can retry or cancel the action.



Confirmation

The device is not approved by the server. Please contact the administrator to approve this device.

Figure 21.: Not Approved error popup message

B.5 SET SITE

The user can select the site for the device in this menu and use the filters by clicking the button on the top right corner to filter the sites by its values.

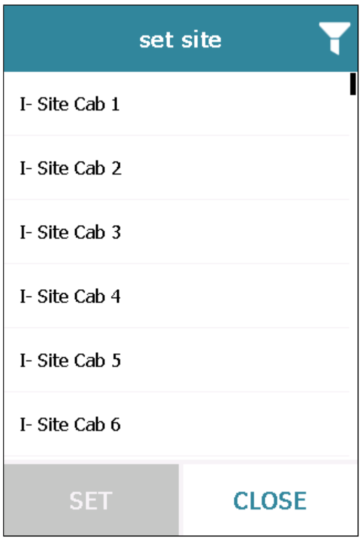


Figure 22.: Set Site

B.6 CHECKING FOR UPDATES

This is the loading screen when the user requests the Agent to check for updates.

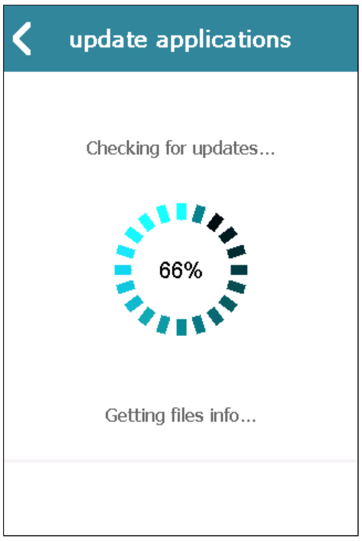


Figure 23.: Requesting the WS for Updates

B.7 DOWNLOADING FILE VERSIONS

When downloading, this form shows the percentage of download completed for the corresponding file next to its name and the overall percentage on the center of the loading animation.

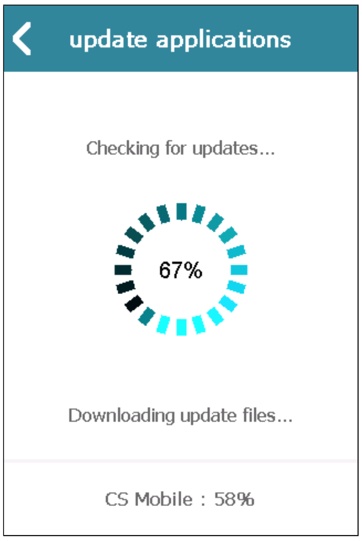


Figure 24.: Downloading *CS Mobile* file version to be installed