Profisee Group, Inc.

The MDS User Interface: Path to Optimal MDM Value

User-experience requirements for making the MDS UI build vs buy decision

Microsoft Master Data Services (MDS) is a powerful master data management platform, but it lacks good options for a user interface that is similarly "enterprise-grade". As a result, companies that want to use MDS as an enterprise MDM platform may be looking at a choice between building a custom UI on top of MDS (or getting a system integrator to build one for them), or buying an off-the-shelf solution, such as Master Data Maestro. As a follow-on to our previous paper, The MDS Desktop User Interface: Build or Buy?, this paper will help you evaluate the build-versus-buy decision. This paper looks at the decision from the detailed perspective of the specific UI requirements that must be met to fully and effectively support the work of your power data stewards, and presents comparative build vs buy cost estimates for meeting these requirements. This will give you a solid understanding of exactly what you would have to build to deliver the required UI functionality on top of MDS, as well as a detailed picture of what you gain from investing in a Master Data Maestro UI solution for MDS, so that you can make an informed and cost effective build vs buy decision.

Executive Summary

When you look at the breadth of the requirements outlined in this paper, you will see that it is a significant task to build a User Interface (UI) to optimize the Master Data Management (MDM) user experience and maximize the value of Microsoft Master Data Services (MDS). Unfortunately, many organizations embark on a build project because they have failed to understand and factor in those requirements. Instead, they are responding to a specific current challenge – for example, mastering and managing employee information – and end up focusing their development efforts on the specifics of that one domain and system, on the ways users access and use data in just that area. As a result, they may find it easy to make a build decision, and that may look like a financially advantageous decision – if they don't factor in future scalability, support for additional domains, or the full breadth of user requirements for handling complex MDM activities.

It's equally easy – and ill-advised – to embark on a project to build a UI without a thorough risk analysis. High failure rates (upwards of 70 percent) ascribed to in-house IT-related development projects derive from a number of common causes, including:

- Incomplete/changing requirements and lack of user involvement
- Lack of resources / employee, management turnover
- Unrealistic expectations / lack of management
- Lack of planning / estimation & scheduling issues
- Technical risks / skills deficit / functionality sacrifice
- Unavoidable risks (can't be controlled or estimated)

Cost comparison methodology

In determining the build costs represented in our build vs buy comparison, we have used a methodology that assumes a conservative average burdened cost-per-development-hour of $150. The number of estimated development hours required to build out a solution for each of the power user scenarios is based on Profisee's internal development experience as well as customer experience. Profisee has documented custom development costs from the customer perspective through interactions with organizations that have invested in building a solution, have subsequently determined that their custom solution is insufficient to meet their practical needs, and have engaged with Profisee to purchase a Maestro solution.

In terms of the cost-to-buy estimates, these assume a ground level investment starting point to address each user sub-scenario; however, in reality, you will need to address most or all of these scenarios, and would NOT need to purchase the Maestro base product for each. As you would also likely gain some reuse from any custom development effort undertaken to address more than one scenario, this from-the-ground-up approach for both cost estimates allows us to make more meaningful comparisons at the individual scenario level.

Even without factoring in the potential cost impact of these risk factors, you will see that there is already an overall cost factor of more than 6:1[1] in favor of the buy decision over building a solution in-house. However, through an assignment of likely percentage of impact for each risk area, based on patterns revealed through IT project failure analyses, we get a risk-adjusted perspective that changes the ratio to more than 8:1 in favor of a Maestro buy decision, as shown below:

Figure 1. Risk adjusted buy vs build cost estimates

Risk Area	Potential Cost Impact %	Total Build Cost Estimate in $1000s	Risk-adjusted Build Cost Estimate	Total Maestro Cost	Risk-adjusted Maestro Cost in $1000s
User Requirements	.15	$1,716	$1,973	$275	$275
Resources	.08	$1,716	$1,853	$275	$297
Management	.08	$1,716	$1,853	$275	$275
Estimation	.08	$1,716	$1,853	$275	$275
Technical	.08	$1,716	$1,853	$275	$275
Unavoidable	.03	$1,716	$1,670	$275	$283
Total (Adjusted for All Risk Incurred)		**$1,716 x .5 =**	**$2,574**	**$275 x .11 =**	**$305**

Although we've broken down costs by user scenario, in order to make a valid build vs buy comparison, you need to look at supporting all of your user requirements, data management strategies, and domains, across business units, across the enterprise, for the immediate future and beyond. This paper looks at the full breadth of support required to have a truly enterprise-grade MDM solution, and reveals the true costs of providing that support. It goes into great detail about what is required to support each user scenario, and reveals the true complexity and inherent challenges of meeting these requirements. If simply reading and understanding them seems daunting, you have a good indication of how difficult it is to spec, design, build and maintain a system that will meet them – a challenge with which the Profisee development team has great experience, having done just that with Master Data Maestro. With a Maestro buy-decision, you get the benefit of that experience and knowledge, with an out-of-the-box enterprise-grade solution that gives you 100 percent scalability to readily handle escalating numbers of records, and single-solution support for any number of domains, any level of complexity. With Maestro's built in support for other applications, models, user types and so on, you get very rapid time-to-value and an escalating return on investment (ROI) as additional applications are integrated and rolled out across the enterprise.

[1] This ratio is based on the cost to purchase the minimum Maestro configuration required to address all requirements, versus the cost to build the same, with a reduction of 50% in build cost applied to each scenario following the first, based on the assumption that each subsequent build-out would benefit from the core development work accomplished in the initial solution build. It should be noted that both the Maestro buy cost and the estimated build cost associated with each separate scenario assumes a blank slate cost; i.e., if you were to buy or build a solution that only met the requirements of that scenario, you would incur all costs from the ground up.

Introduction

Microsoft Master Data Services (MDS) is a powerful master data management (MDM) platform, but it lacks good options for a user interface that is similarly "enterprise-grade." As a result, companies that want to use MDS as an enterprise MDM platform may be looking at a choice between building a custom UI on top of MDS (or getting a system integrator to build one for them), or buying an off-the-shelf solution, such as Master Data Maestro. In our previous paper, "The MDS Desktop User Interface: Build or Buy?" we discussed what is involved in making a build-versus-buy decision when it comes to the MDS UI. At a high level, that paper laid out important user experience considerations, defined the two primary MDM user scenarios: power data stewards, and workflow or (casual) data stewards, and presented some key UI development challenges for supporting each type of user.

Note: User requirements for initiating, orchestrating and managing the contribution and approval tasks of the workflow data steward will be explored in detail in the next paper in the MDS Buy vs Build series, along with requirements for the development and maintenance of the underlying workflows, forms and management tools.

In this paper, we take a deeper dive into the work of the power data steward, listing the detailed UI requirements that must be met to fully and effectively support this primary user type, and providing comparative build vs buy estimates of costs and time-to-value for meeting these requirements.

This document is intended to reveal the totality of the buy vs build decision – including the 90 percent of the iceberg below the surface – and provide you with important insights for navigating the decision-making process. With this information in hand, you will have a solid understanding of exactly what you would have to build to deliver the required UI functionality on top of MDS. You will also have a detailed picture of what you gain from investing in a Master Data Maestro UI solution for MDS.

Empowering the MDM Power Steward

As defined in the previous paper, the power data steward's role is one of frequent, high-volume interaction with the data, typically performing tasks that cover the entire spectrum of the MDM process. The power steward spends a lot of time with the UI, working on a lot of records, handling frequent updates, entering and managing data on a large scale. These users need personalized, reusable views of the data. They need the ability to do mass updates, imports, and edits. For the power steward, it's all about getting a lot of work done accurately and efficiently, so they need a more powerful, more productive general user experience.

The power steward's work is fairly represented by the scenarios described below. Detailed user requirements are enumerated for each scenario, and we also present comparative estimated costs to build vs buy for each, based on meeting the requirements for each scenario in their entirety.

1. Manage and use custom views

Cost to Build	$$$$$$
Maestro Cost	$

Power data stewards will tend to need many specialized custom views to perform specific tasks. For example, a product pricing specialist may need a specific view of products and pricing related columns for common mass updates that she performs while a production manager wants a completely different view of selected product rows and columns. A certain finance manager in accounts receivable wants a specific view of credit line per customer account. These specialized view requirements are in addition to the many common shared views that will likely be needed across the company. Therefore, self-service for user view creation and customization is a key scenario. Examples of some related requirements include:

1.1. Requirement: View definition

Users define the columns, filters, formats of the view and reuse common attribute groups. Users should be able to adjust existing standard or custom views quickly and incrementally without starting from an empty definition or losing other formats already applied to the view.

1.2. Requirement: Save, recall and share views

Users must be able to save their view definitions for later instant recall. The view definitions should be portable across environments and sharable / distributable to other users. Users should be able to save a complex workspace definition involving multiple saved views saved in a particular arrangement or relationship.

1.3. Requirement: Integrated paging, filtering and sorting

With large data domains, the paging, filtering and sorting functions are required and must intelligently integrate with backend server services. For example, to retrieve the top 50 products, the UI cannot rely on retrieving all (perhaps millions) of the records and then sorting locally. You also can't expect users to define very restricted filter criteria before retrieving and browsing any records.

1.4. Requirement: Response to change

Are the view definitions resilient to change? Definitions of business entities, attributes and hierarchies will undoubtedly change over time. What happens when an attribute name change is required? The view definition mechanism must respond to changes, additions, deletions without breaking and without losing the work of end users.

1.5. Requirement: Version flag selection and version migration

When defining views attached to a certain data version, the users must be able to select by version flag when version flags are moved from version to version to communicate and redirect users to the correct version of data. Furthermore, the user should be able to easily redirect their views from one selected version to another.

2. Mass change

Cost to Build	$$$$$$$$
Maestro Cost	$

The UI for power stewards must support mass change capabilities. For the power steward, mass change can mean a dozen, dozens, or even hundreds of records – numbers that the user definitely won't want to change one record at a time. Power stewards should have the power to change these large numbers of records in a single operation, whenever needed.

Power stewards frequently need to make the same change to a large number of records – for example, changing the reorder point for all of the products in a particular category to accommodate a longer lead time for shipping; or making changes to customer records in the wake of a territory realignment or salesperson change, where all customers assigned to salesperson Fred need to change to John. In most user interfaces, the user must edit the records one at a time; bring up a list of all products in the category, for example, click one product in the list to open an edit form, change the reorder point for that product, save it, and then go back to the product list to select the next product, and repeat the process. This is a completely impractical and time-consuming process for the power steward, who is typically working with very large numbers of records.

The Maestro interface enables the user to highlight the entire range of records to be updated – for example, all products for which the reorder point is being changed – then type the new value one time, and hit enter to change all records at once. Alternatively, the user can copy and paste the selection across rows. These are some mass change usability features that are crucial to the power steward's productivity; features that make building a good UI very difficult, and which are often overlooked or simply not attempted in a build scenario – but they are features that can ultimately make the difference in the effectiveness of an MDM implementation.

2.1. Requirement: Integrated paging, filtering and sorting

In a typical power steward scenario where the user needs to change about 50 records from a dataset of millions, the UI will determine how quickly and easily those specific records are displayed on the screen. Or, where even more records are returned – if, as may well be the case with the power stewards, thousands meet the criteria – the user should not have to download the whole list. The UI should provide an option to download, for example, the first 100 records, and then the next 100, and so on, so that the user can see what needs to be changed and act quickly, without having to wait for the system to retrieve the entire list again each time. The user should have the option to sort specific records to the top of the list, and handle changes in multiple different page sets of records.

2.2. Requirement: Accumulate and publish changes in bulk

In order to accommodate mass changes, it is necessary to allow the user to accumulate a set of changes, and then have a view that shows staged changes locally in the UI, so that the user can see the impact of the changes before committing them. The user must then be able to publish the changes in bulk.

With mass change capabilities, the user has a very “sharp knife”, and can change large numbers of records just like that; therefore, it should be done very carefully and cautiously. In the Maestro interface, the changes are accumulated and highlighted, capturing the fact that the user wants to change these records, but it doesn’t actually make the changes until the user can see what is being changed, visually confirm that it is right, commit and publish them all at once; or spot any mistakes, backup and undo, without publishing, without having affected the underlying model.

2.3. Requirement: Mass edit

The user should be able to perform mass edits, for example, when there are a number of similar products or customers all requiring the same types of changes. This could be the case where a new sales representative takes over responsibility for all of the customers in a particular territory, or when product categories are realigned, and a large number of products need to be assigned to a different category. In these types of situations, the power steward will want to have the power to change, for example, 100 records at a time. In order to support this, the UI should allow the user to select a range of cells, and select a value to apply across multiple rows within the selection, in order to change the same property across multiple records.

2.4. Requirement: Mass copy / paste

An important element of mass edit capabilities will be to allow the user to copy one value and easily apply it to multiple rows.

2.5. Requirement: Undo / redo

The user making mass changes must be able to review those changes on screen prior to publishing, and have the ability to easily undo and redo changes.

2.6. Requirement: Bulk import and merge

In making mass changes, the power steward may not be keying the changes directly in the UI; instead, for example, they may be importing the changes from a spreadsheet. For this, they need bulk import and merge capabilities that allow them to map the new information to the data model, and then apply changes all at once. The merging aspect of this requirement is key. The UI must be able to find and update existing records, versus creating new ones, and be able to set up a relationship; for example, when importing new products that will be assigned to a new category, as well as assigning existing products to the new category.

3. Navigate efficiently

Cost to Build	$$$$$
Maestro Cost	$

The power steward spends a lot of time navigating the data model, working with many business entities, moving among various views of entities. It is critical to productivity that this navigation be accomplished without making the user wait for lists to be populated each time. If you don't build in awareness of model, and cache certain information, then every time the user needs to navigate within the data model, it will be unacceptably slow. For example, if the user wants to modify a view, add a column, the UI would have to get the full list of columns every time. If it is unaware of permissions, then each time a user attempts to navigate to a new view, the UI will have to find out for that user what they are allowed to see; navigate to another entity, click on menu, UI has to stop and query all entities in the model. This results in a very slow application, with unhappy, unproductive users. The following requirements are essential to the power steward's ability to navigate efficiently.

3.1. Requirement: Cache context for performance

The UI should cache the user context – who is this user? What does the model look like to this user? What entities, columns, and attribute groups does this user have access to? What is the set of information this user needs to know about a given column? Is it sourced from a particular entity? For example, if the user clicks on a DBA, the UI will need to present a drop-down list of applicable values. To ensure a fast response on user-click, the UI will need to cache all of this user context data, rather than waiting until the user attempts to perform an action, and then re-establishing the user context each time an action is performed.

3.2. Requirement: Linking and drill down

The requirement for linking and drill down capabilities refers to the user's ability to start on a given record, and jump, for example, from account to contacts (customer account is related to multiple customer contacts), or from category to products (product category is related to multiple products). If you are building the UI, you will have to either build in the functionality to accommodate linking and drilling down contextually, on the fly, or you will have to hard code the links and relationships. If hard coded, you will need to determine how you will handle ongoing model changes that affect those relationships, and look at how this will affect performance.

3.3. Requirement: Hierarchical filtering and drill down

The users need to be able to navigate through the hierarchy tree, editing as they go. For example, start with a set of regions, click on a specific region, and be able to see the customer list for that region, and make changes as needed, from within the hierarchy view. Or for products, be able to navigate through a product category hierarchy, click on a category, view and perform operations on the products within the category. Any UI development work will need to provide that level of hierarchical navigation and editing capability for power stewards.

3.4. Requirement: Visual cues for read-only columns and records

Where there are attributes that certain users don't have access to, the UI will need to present the user with visual cues. If you build a UI, you will need to determine how you will identify which columns and records are read-only for a specific user (look up user's permissions), prevent the user from performing operations outside of their permissions, and develop a mechanism for the UI to represent this information to the user – "grey out" or otherwise disable illegal operations? Error message on submit? All of this functionality must be specified and developed – including performance considerations – to design the right user experience for your power stewards.

3.5. Requirement: Multiple related views open

The power steward will often work with multiple related views open, and will need to keep the views in sync as changes are made. For example, with the product list open, the user may need to access and manage the category list, but doesn't want to have to jump out and completely lose all context of his work within the product list to do so. He wants to be able to return to the product list right where he left off, have the view still open, scrolled down and over to the exact position he was in, with all his changes, even unpublished, still there. How will the UI accommodate this requirement? Can the user keep both lists open, and toggle between them, jumping back and forth easily, without losing their place, or work in progress?

Meeting this requirement presents significant challenges if you're working with a web UI, where you're displaying only one thing at a time. If the user clicks to navigate to category, he is leaving product, going to a new page. If he goes back to product, it's going to reset everything – previously selected filters are gone, scroll position is gone, record selection is gone. While building on the MDS Web UI may seem to present a shortcut, it won't account for these kinds of key usability requirements.

Additionally, with multiple views open, how will the content of one view be refreshed to reflect any changes made in another open view? If the user adds a subcategory in the category view, then flips back to products, does the UI have to reload the whole products view, discarding the work in progress, or is there a way to refresh only the changed content of the product view to make the new category visible?

4. Perform complex data edits

Cost to Build	$$$
Maestro Cost	$

The power steward must be able to efficiently perform complex data edits, with domain-based pick lists, on-the-fly attribute creation, sortable/filterable large domain lists, support for file and link data type editing, data type enforcement, pop-up calendar for date selection, and long text field editing capabilities.

4.1. Requirement: Domain pick lists and automatic additions

The UI must provide domain pick lists; simple drop-down lists based on the contents of the domain or domain-based attributes. The user must have the choice to display and sort domain lists either by business key or by descriptive name. In addition, if the user goes to select, for example, a product category and a product, and the desired category has yet to be created, the UI should make it easy for a user with the appropriate permissions to simply enter the new value and have it created on the fly. This should be an option that is available from within the pick list, without requiring the user to leave where they are, go add it somewhere else, and then come back to complete their work.

4.2. Requirement: Large domain pop-ups that are sortable and filterable

In some cases, your domain pick lists may be too long to be handled in a simple drop-down list. When you get more than 30 to 50 choices in a drop-down list, it becomes unusable. For example, you can't pick from 10,000 products in a drop-down list. In such cases, the UI will need to provide a full list of products with all the different attributes, and allow the user to sort and filter to find the right one and select it. The Maestro UI addresses this requirement by presenting a pop-up dialog with a full grid list of choices in the domain, allowing users to sort and filter it in any way they need to in order to find the item they want to link to.

4.3. Requirement: File and link data type edits

If your model uses specialized kinds of attributes like file attributes or URL links, you will need to provide an appropriate user experience; one where the user can open and view the contents of a file, and then select another file to update that attribute. Some thought will need to be given to how to build this capability into the UI. This can be especially difficult in a browser application, which typically requires the user to download something first, and then open it separately, as opposed to just clicking and opening and viewing in-line.

4.4. Requirement: Data type enforcement for numbers, dates, text

Data type enforcement ensures consistency and accuracy. When you design the UI for the create/edit experience, are you going to dynamically figure out and enforce the data type for numbers, dates and text, based on the model meta data? Or will you try to hard-code it?

4.5. Requirement: Calendar date selector

The UI should provide a pop-up calendar for selecting dates. Some thought should be given to how your will incorporate that capability, and what that will add to the scope of building the UI.

4.6. Requirement: Long text editor

If the user starts entering text in a small cell in a grid, but needs the ability to enter multi-line text, as frequently happens with the work of the power steward, the UI will need the capability to accommodate the required long text entry. For example, instead of a fixed-size cell, you might need to provide the ability for the user to pop out into a sizable, scrollable window for that field, so it can expand to be as large as it needs to be for that user in that situation.

5. Matching and survivorship review and editing

Cost to Build	$$$$$$
Maestro Cost	$$$

A key activity of the power steward is reviewing and editing matching/survivorship results. This will require support for sorting and filtering, and creating page lists, drill-down into matching details, and approve, reject or find other options from within the detail display.

5.1. Requirement: Paging, sorting and filtering of match groups and match group members

Typically, with matching, the power steward is going to want to sort, filter, and create page lists of records to find the things that require review. The user will want to find and list all proposed matched members, and then have the ability to select an item, and see the group in context, alongside the matched members list. This requirement will not be met by presenting the user with a single, hierarchical list. The UI will need to provide the ability to sort and filter, for example, to create a list of just masters, and then within the one group the user wants to review, the ability to sort and filter that, or even page it, differently, in order to work effectively. This requires the user to be able to control separately how the rows within the group are presented.

5.2. Requirement: Match group detailed display and drill-down

In order for the power steward to perform an effective review, the UI will need to provide functions to drill down into the matching details, so the user can see why something was matched in a specific way. The details of the match score should show not only which specific record was matched, but also the breakdown on each matching attribute, and within those matching attribute values, show any corrections with synonyms and normalization rules. These details are useful in determining why, even though two values don't look quite the same, after they were normalized, they were closer, and once they were closer, the similarity calculation scored them in this way. Availability of this type of detailed information is absolutely critical to the work of the power steward.

5.3. Requirement: Approve, reject and find match actions

From within the match group detailed display, the user must be able to approve, reject or select a different match and join it to that member, or move it to that group. The user should also have the option to see all other potential matches, in order to determine whether the proposed match is the best fit.