Microsoft Codename “Montego” – Data Import, Transformation, and Publication for Information Workers

Stephen J. Maine, Lorenz Prem, Clemens Szyperski, James F. Terwilliger,
and the Microsoft “Montego” Team
Microsoft Corporation
{smaine, lorenzp, clemens, jamest}@microsoft.com

1. INTRODUCTION

A fundamental problem in database systems is deriving useful information from untold quantities of data fragments that exist in the web’s data stores. Data is abundant, useful information is rare.

This problem space plays host to many successful and innovative solutions from industry (e.g., [2, 4, 5, 9]), and the open-source community (e.g., [11]). Each solution has its strengths and weaknesses based their balance of utility and usability. In this paper, we demonstrate the unique approach to data mashups that Microsoft Codename “Montego” brings to the space. The “Montego” tool allows non-technical users to create complex data queries in a familiar graphical environment, while making the full expressiveness of a query language available to professional users. “Montego” operates both as a standalone client, where a user can launch it from an application like Excel® to import and manipulate data into a spreadsheet, or as a cloud service, where a user can take the product of data transformation and publish its results into a database or to the web as an OData feed.

The “Montego” formula language – “M” for short – and the associated runtime provide the muscle necessary to create data mashups that consist of data from many sources and formats. “M” is a language similar in intent to the language used in the Excel formula bar. Like Excel, the “Montego” tool allows a user to construct data transformations piecemeal through composition; in Excel, one can build complicated expressions out of smaller ones assigned to spreadsheet cells. “Montego” uses that same paradigm to build larger expressions component-wise, either through explicit writing of the expression or through using gestures in the “Montego” UI.

Unlike Excel (and SSIS [9], and many other data integration or mashup tools), the entire instance of a “Montego” data integration session can be serialized as an instance of the “M” language. The rich duality between gestures and formulas enables both step-at-a-time convenience as well as the supportability, optimization, and reuse capabilities of a full-featured language.

The remainder of this paper is organized as follows: Section 2 covers the scenarios used to demonstrate “Montego” capabilities; Section 3 outlines some of the technical challenges addressed by the “Montego” tool’s implementation. The tool is the work of many whose contributions we wish to acknowledge.

2. WHAT IS DEMONSTRATED

We demonstrate the “Montego” tool in different settings drawing data from multiple data sources, transforming that data in non-trivial ways, and publishing that data by several means. In this paper, we show one such scenario, where we create a summarized data extract from three unrelated heterogeneous data sources. The tool uses a visual interaction paradigm approachable by business users familiar with Microsoft Office products. In our scenario, we combine Orders, Products, and Suppliers to create a summarized data set showing Sales by Supplier and Products.

After starting the “Montego” tool, the first step is to create references to the three data sources: Orders from a SQL Server database, Product data from an Excel spreadsheet, and Supplier information from the Internet in the form of an OData feed. This process is accomplished visually using the “Montego” user experience, which guides the user through the process of discovering and connecting to each data source. Each data source is introduced into the “Montego” environment as a named resource that can be referenced by subsequent tasks.

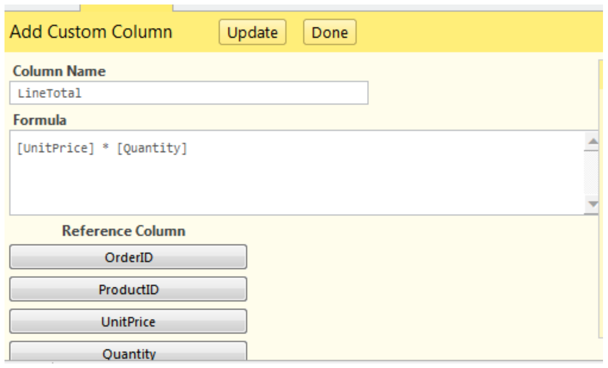
The next task is to reshape Orders into a summarized view of Sales by Product, a task with several steps. First is to add a computed column named LineTotal by calculating Unit Price times Quantity for each row in the Orders data set, accomplished visually via a formula builder, as shown in Figure 1. The builder constructs an equivalent formula in the "M" formula language:

```
Table.AddColumn(Orders,
   each [UnitPrice] * [Quantity], "LineTotal")
```

In “Montego”, user interactions can be expressed visually via formula builders or textually via “M” expressions, and the tool facilitates easy transitions between visual and textual work styles. The formula in the builder can be constructed manually, or the user can use the buttons shown in Figure 1 to add references to the available columns (for details on “M”, see Section 3).

To complete the shaping of the Orders data set, we hide unnecessary columns from the data set and sum the results of the LineTotal computation by ProductID. This step is again accomplished visually by adding a task to the linear task stream representing the overall shaping operation. The task stream is represented visually in the tool, as shown in Figure 2.

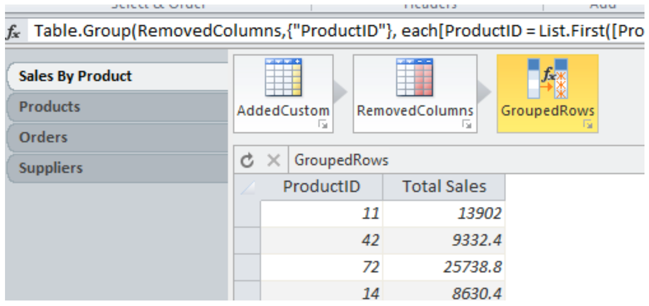
At the left side of Figure 2 is a vertical list of items called *resources*. Each resource represents a value that can be referenced by name by any formula in the tool. Clicking on a resource reveals a preview of the value of that resource, in addition to the task



Column Name: LineTotal

Formula: [UnitPrice] * [Quantity]

Figure 1. The builder for a lookup column addition.



```
Table.Group(RemovedColumns,{"ProductID"}, each[ProductID = List.First([Pro
```

ProductID	Total Sales
11	13902
42	9332.4
72	25738.8
14	8630.4

Figure 2. Grouping rows; note the result preview.

stream (shown as a horizontal set of steps) that generates the resource's value.

The expression represented by the task stream shown in Figure 2 for resource "Sales By Product", in the "M" formula language:

```
#"Sales By Product" = let

  AddedCustom =
            Table.AddColumn(Orders,
                each [UnitPrice]*[Quantity],
                "LineTotal"),

  RemovedColumns =
            Table.RemoveColumns(
                AddedCustom,
                {"OrderID", "UnitPrice", "Quantity"}),

  GroupedRows =
            Table.Group(
                RemovedColumns,
                {"ProductID"},
                {{"Total Sales", List.Sum}})

in

  GroupedRows;
```

Although the user is building expressions in "M" exclusively, the entire expression above is delegated to SQL Server for execution. The "Montego" tool is capable of determining that the "Sales By Product" formula is dependent only on the Orders data set, which is stored in a SQL Server database. Thus, the bulk of the computation needed to produce the Sales By Product view is done on the database server and not the "Montego" runtime. In general, "Montego" translates "M" formulas to equivalent T-SQL syntax for efficient execution by the database server instead of the "Montego" runtime whenever such translation makes sense.

To create the final summarized view of Sales By Product With Supplier, we use "Montego" to join each Product (stored locally in an Excel spreadsheet) with a list of Suppliers exposed on the Internet as an OData feed. Next, we incorporate the computed value for Total Sales from the previous resource by adding a lookup column keyed by ProductID. Finally, we limit the data to only the relevant information by excluding all but the Supplier, ProductName, and Total Sales columns.

Figure 2 shows the "Montego" tool's preview of the current result set. The user can go backward and forward along a task stream to see the data changing at each step and to make changes to each task's definition, as appropriate. Once a satisfactory result set has been formed, the user has the option to share the computed data set in a number of ways, including publishing to a SQL Database table, filling an Excel spreadsheet with the output of the computation, or publishing the data set broadly as an OData feed.

Figure 3 shows the "Montego" integration with Excel, offering to add the resulting data to an Excel range; Figure 4 shows the filled Excel sheet.

3. TECHNICAL DETAILS

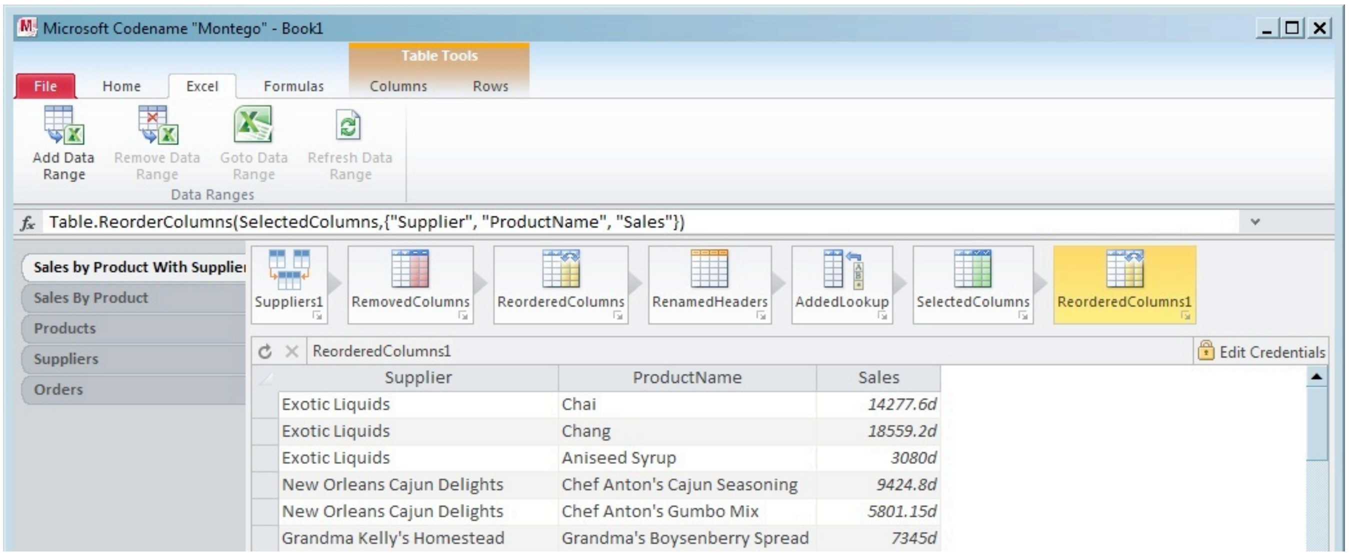
"Montego" encourages the interactive building of complex expressions in the presence of dynamic result previews. To facilitate this interactive functionality, the "Montego" runtime uses special optimizations to quickly get partial preview results.

The interactive model focuses on linear chains of tasks cascaded on top of each other. Each such chain can be used to define the starting point of one or more further chains. A chain can be started by merging (joining) multiple starting points. In essence, users of "Montego" create complex expressions that can be understood as fork/join dataflow graphs – but without ever encountering such terminology or bewildering graph-based topologies in the UI.

The user can switch between UI and textual "M" views at any time – all information is captured in "M" with no additional state. Editing either view maintains the other (since the UI view is always derived from the "M" text). All common "M" expressions are covered by the UI. However, one can write more advanced expressions, such as custom function definitions, that are not covered by the UI. Such expressions will appear verbatim as "M" text in the UI.

The "Montego" system takes an expression authored in "M" and evaluates it by drawing on the query capabilities of external stores and a local runtime. The local runtime is used both to decide how to federate work and to back-fill work locally that cannot be federated. External data sources have query capabilities of varying degrees, from trivial get-all mechanisms like reading a text file to full-blown query processors like SQL. The computational cost of copying data versus sending sub-queries for remote evaluation varies, including dynamic variations caused by effects such as current network performance. Finally, the real cost of evaluation versus bulk copying varies. "Montego" aims to cover all these variations to a degree enabling a good user experience.

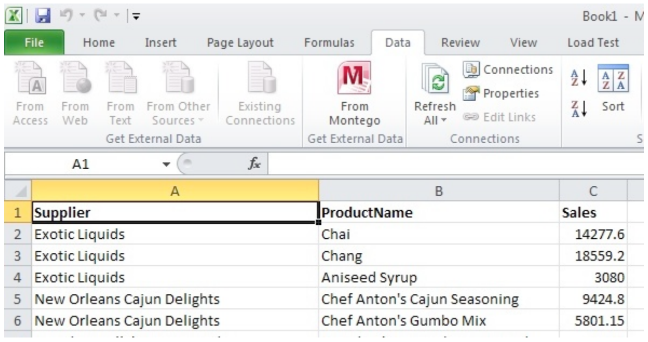
The "M" language is a dynamic, higher-order functional language with lazy record, list, and table constructors and a simple uniform data model. "M" has types as first-class values and uses dynamic, lazy type checking to assert type constraints. For instance, an assertion that a streamed list contains only values of a certain record type can be stated but the assertion will only be checked as values are accessed, thus not undermining data-streaming. More complex structures are built from composing the primitive types.



```
Table.ReorderColumns(SelectedColumns,{"Supplier", "ProductName", "Sales"})
```

Supplier	ProductName	Sales
Exotic Liquids	Chai	14277.6d
Exotic Liquids	Chang	18559.2d
Exotic Liquids	Aniseed Syrup	3080d
New Orleans Cajun Delights	Chef Anton's Cajun Seasoning	9424.8d
New Orleans Cajun Delights	Chef Anton's Gumbo Mix	5801.15d
Grandma Kelly's Homestead	Grandma's Boysenberry Spread	7345d

Figure 3. The Montego user interface with the preview of an expression, and the Excel context menu to import the result into Excel.



	A	B	C
1	Supplier	ProductName	Sales
2	Exotic Liquids	Chai	14277.6
3	Exotic Liquids	Chang	18559.2
4	Exotic Liquids	Aniseed Syrup	3080
5	New Orleans Cajun Delights	Chef Anton's Cajun Seasoning	9424.8
6	New Orleans Cajun Delights	Chef Anton's Gumbo Mix	5801.15

Figure 4. Excel filled with results.

The "Montego" runtime uses two core technologies to make federation decisions: typeflow and cost estimation. Typeflow is used to predict tight type bounds over sub-expressions to determine whether federation to typed systems such as SQL or OData (EDM) is feasible. It is also used to drive the UI; suitable controllers and views are chosen based on the type of the current context. For instance, if the current preview is table-shaped, then a tabular preview is used and table tools are offered while record or hierarchy-based tools are hidden. The UI shown in Figure 3 exhibits both cases: the preview shows tabular data and the "ribbon" at the top has highlighted "table tools".

The typeflow system uses advanced bi-directional type inference [10] based on abstract interpretation over partially evaluated contexts. For example, a select-like query at list level might take the following form in "M":

```
List.Transform(
  {[ID=1, Name="Alice"], [ID=2, Name="Bob"]},
  each [Name])
```

Here, `each [Name]` is "M" shorthand for a unary function `(item) => item[Name]`, where `item` is an item (a record) in the list and where `item[Name]` selects the `Name` field of that record.

Based on the schema of the list (the first argument to `List.Transform`), typeflow infers that the untyped anonymous function passed as the second argument is a function from a record of type

```
type [ ID = Number.Type, Name = Text.Type ]
```

to a value of type `Text.Type`. With that inference established, typeflow determines that `List.Transform`'s return type is

```
type { Text.Type }
```

That is, a list of simple text values. For typeflow to work even in this simple example, inference must flow both top-down and bottom-up. Note that cases like the following (from the demonstration) require inference based on both values and types:

```
Table.AddColumn(Orders,
   each [UnitPrice] * [Quantity], "LineTotal")
```

Specifically, the value "LineTotal" (a text value) is interpreted by `Table.AddColumn` as the name of a new column thus affecting the type of the return value of `Table.AddColumn`. Typeflow draws on values where available to accomplish such inferences.

Cost estimation is used to predict the multi-dimensional cost facets if federation of a feasible sub-expression were performed. The cost-prediction heuristics map costs into real units (such as time, space, and money) to enable effective federation decisions in highly heterogeneous systems.

Query federation is ultimately performed by "folding" nested queries over queryable sources. The folded sub-queries are then translated to external query languages such as SQL, OData, or XPath or to instructions for internal data processors that crack files in formats such as CSV or Excel xlsx. Our approach is similar to other federated systems where portions of queries are delegated to remote servers (e.g., [1, 3]). Unlike at least some of these systems, the user is never required to know the syntax or peculiarities of the remote system; the user simply writes "M" code (or just uses the UI) and "Montego" constructs queries as possible to federate the computation.

One of the key obstacles when attempting query federation is semantic differences among query processors. Such differences range from varying floating-point, to differences in date and time handling, to differences in the interpretation of 'null' or sparse data, to varying capabilities to express complex and simple types. "Montego" addresses these issues by relying on "soft" semantics, accepting reality. That is, no promise is made to "fix" an external query engine that diverges in niche semantics from the baseline "M" semantics. It is too early to evaluate the impact of this nontraditional choice on actual users of "Montego". However, there is precedence in the soft semantics of the Web.

Figure 5 shows an overview of the "Montego" architecture. In essence, "Montego" builds a bridge from UI or "M" authored expressions to a diverse world of data sources and back to uniform data publication mechanisms. The target audience is information workers, such as staff analysts working on periodic but constantly changing tasks over a multitude of data sources.

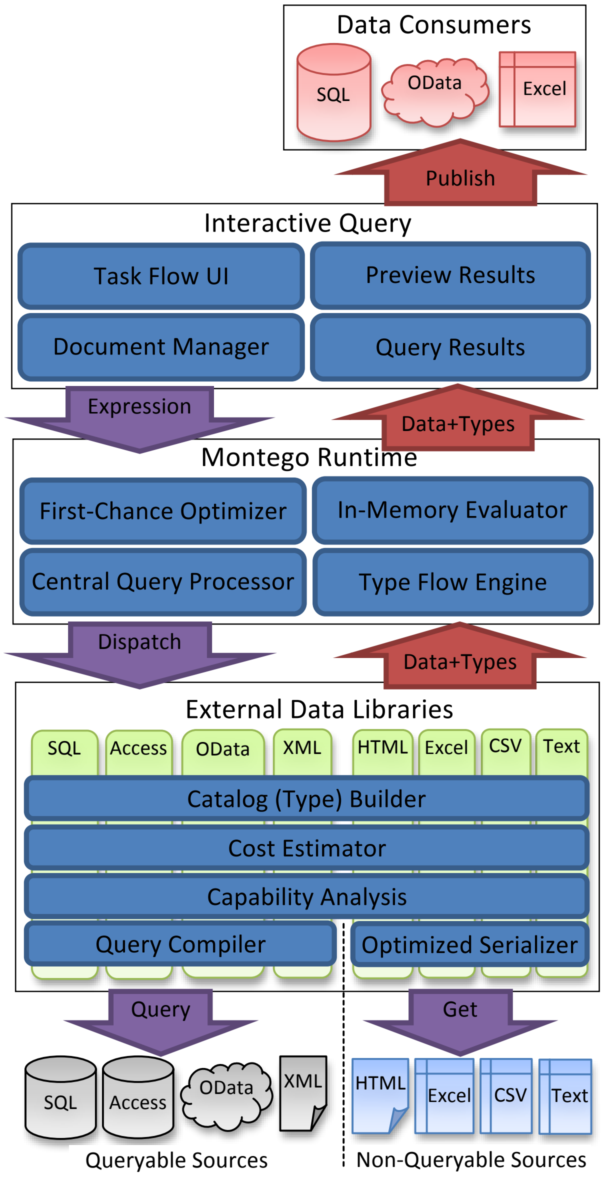


Figure 5. The Montego Architecture.

4. RELATED WORK

JackBe [5] is a leader in the mashup space. The company's Presto suite uses flowcharts to visualize the flow operations in a mashup. Like Montego, JackBe places significant value on presenting the sometimes complex mashup in an understandable way. At the engine level JackBe promotes its own language for authoring data transformations, the 'Enterprise Mashup Markup Language' (EMML). Yahoo Pipes [12], another tool in this space, distinguishes itself by its large catalog of adapters. Like JackBe, it uses the flowchart analogy to visualize a mashup.

Kapow [6] takes a similar approach to SSIS [9]. Having created an impressive offering for programmers, the company seeks to simplify the concepts involved to make the suite appealing to information workers.

Microsoft Excel [7], coming from spreadsheet space, is expanding its data processing capabilities. As the world becomes more integrated, Excel users want to process increasingly complex datasets. The addition of PowerPivot [8] to the Excel ecosystem illustrates this trend.

5. REFERENCES

[1] J. A. Blakeley, C. Cunningham, N. Ellis, B. Rathakrishnan, M.-C. Wu. Distributed/Heterogeneous Query Processing in Microsoft SQL Server. In: *ICDE,* pp. 1001–1012, 2005.

[2] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, J. Goldberg-Kidon. Google Fusion Tables: Web-Centered Data Management and Collaboration. In: *SIGMOD,* pp. 175–180, 2010.

[3] L. M. Haas, R. J. Miller, B. Niswonger, M. T. Roth, P. M. Schwarz, E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.

[4] IBM Corporation. IBM Mashup Center. http://www-01.ibm.com/software/info/mashup-center/.

[5] JackBe Corporation. Presto Suit. Online http://www.jackbe.com/.

[6] Kapow Software. Kapow Katalyst. http://kapowsoftware.com/products/kapow-katalyst-platform/index.php.

[7] Microsoft Corporation. Excel. http://office.microsoft.com/en-us/excel/.

[8] Microsoft Corporation. PowerPivot. http://www.powerpivot.com/.

[9] Microsoft Corporation. SQL Server Integration Services. http://msdn.microsoft.com/en-us/sqlserver/cc511477.

[10] B.C. Pierce and D.N. Turner. Local type inference. In: *POPL*, pp. 252–265, 1998.

[11] V. Raman, J. M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In: *VLDB,* pp. 381–390, 2001.

[12] Yahoo. Yahoo Pipes. http://pipes.yahoo.com/pipes/.