Data Services in your Spreadsheet!

Régis Saint-Paul, Boualem Benatallah
School of Computer Science & Engineering
University of New South Wales
Sydney NSW 2052, Australia
{regiss, boualem}@cse.unsw.edu.au

Julien Vayssière
SAP Research, RC Brisbane
Level 12, 133 Mary Street
Brisbane QLD 4000, Australia
julien.vayssiere@sap.com

1. INTRODUCTION

End-user programmers—the 45 million of them, as estimated for 2001 in US alone [7]—routinely use spreadsheet to visualize, manipulate, and analyze data. Thanks to this environment, they can build applications that solve their daily problems. Even building a report can be seen as programming an application that takes corporate data as input and outputs a presentation. To build this application, spreadsheet users have to import data and place them in spreadsheet cells, highlight the important pieces, compute maybe some aggregates, add a chart or two. If well done, this application will be used each time data are updated to effortlessly produce a fresh report.

Service oriented architecture (SOA) emerged as a response to the general problems of enterprise application integration (EAI) and enterprise information integration (EII) [5, 2]. This architecture offers a unified and high-level view of the company resources.

In particular, the advent of data services [4] and standards such as Service Data Objects (SDO)[8], professional developers now benefit from high-level and integrated data access. Integrated because developers can access transparently from a single end-point to information that may be managed by various systems and stored in various locations. High-level because data services rely on a conceptual modeling of the information—namely the Entity-Relationship (ER) model. For example, developers can retrieve from a data service an entity *customer*. This entity presents information retrieved for some parts from a relational database and for the rest from a supply chain management software. From this entity, developers can also access to related entities such as the *purchase orders* or the *invoices* of this particular customer.

Undoubtedly, spreadsheets need to be integrated with SOA. Indeed, there already exist some efforts to this end. For instance, Microsoft Excel Services [1] allows to incorporate Excel computations as part of a larger process. Another example is given by Visual Studio Tool for Office (VSTO) which allows to isolate the presentation elements of a spreadsheet from the data it contains. Data are stored as separate XML documents and can be consumed by other applications.

These initiatives, however, are intended for professional developers. A solid background in object-oriented programming is prerequisite for using VSTO. Manipulating SDO entities can alternatively be done by using the macro language that accompanies spreadsheet environments. But few end-users are ready to invest time in learning a macro language, which resort to learning object-oriented programming.

Other approaches, targeted toward end-users, exist. For instance, StrikeIron [3] is a commercial add-in to MS Excel that allows to perform simple invocation of web services. However, the features it offers do not allow manipulation of complex objects as found in data services. The invocation results in imported data in the spreadsheet that are a collection of atomic values in cells. The relationship that may exist among these data, how they relate to an entity, or how the imported entity is related to other entities is lost.

For the majority of end-users, spreadsheet programming means formulas and cells manipulations only, sometimes assisted by wizards or visual assistants. An integration solution has to preserve this programming model and it has to accommodate existing spreadsheet environment.

In this demonstration, we present SpreadATOR, a middleware for integration of spreadsheets with SOA. We make the following contributions:

- In order to leverage end-users expertise in spreadsheet programming, we base our solution to forming web services invocation messages on a formula language. This language allows more expressiveness than purely graphical approach while remaining simple enough for end-users experienced in spreadsheet programming;

- Data service invocation typically returns composite entities such as customer. Our approach makes these composite entities first class values of cells. Users can then layout the various components of an entity and maintain the original relationships with other entities;

- We propose a method to help users benefit from relationships that exist between entities for accessing related information. For instance, a user may access the list of purchase order entities corresponding to a customer imported by a previous service invocation. We also allow users to program, in the spreadsheet, computations that apply to business entities (e.g. computing the average of the last 5 purchase orders of a customer). These computations can be reused for any entity of the same type (e.g., displaying for all the

EDBT'08, March 25–30, 2008, Nantes, France.

	A	B	C	D
1	**0042**			
2	*Ford*	*Prefect*		
3		150	Beer	
4		2	Towel	
5		1	Babel Fish	
6				
10	...			

(a) Source spreadsheet with hierarchical representation of orders

```
QuoteRequest
  Account
    Login ="MyLogin" {MyLogin}
    Password ="MyPass" {MyPass}
  Orders [ ] (1 item)
    Order
      Id =A1 {0042}
      ShipTo
        FirstName =A2 {Ford}
        LastName =B2 {Prefect}
      OrderDetails [ ] (3 items)
        OrderLine
          Quantity =B3:B5 {150, 2, ...}
          ProdName =C3:C5 {Beer, Towel, ...}
```

(b) Target XML schema with mapping specifications for order 0042

Figure 1: A spreadsheet and its exportation specification

customer of a list the average computed on a single customer).

- We propose to enhance the traceability of imported data by offering a generic facility for meta-data display. Meta-data provide information such as the last update of a value, its precision, its source, etc. These information are central concerns in data quality. We also allow users to reference meta-data when writing formula expressions.

2. SERVICE INVOCATION

Web services are invoked by sending an XML query message. For invoking services from a spreadsheet, it is therefore necessary to allow users to build a message that conforms with the interface of a given service operation (typically expressed using XMLSchema). To build the invocation message, a user has to specify a mapping between the source spreadsheet and the message schema, referred to as the target schema.

The role of a mapping specification is to express "correspondences" between spacial location of data in the spreadsheet and elements of the target schema. The obvious way to express spacial location in spreadsheet is through cell or range notations, a fundamental of spreadsheet programming. We propose to reuse it by expressing mappings through statements of the form l = *Formula* where l is a label (i.e., an element or an attribute in the corresponding XML document) of the target schema and *Formula* is an expression based on classic spreadsheet formula language. This design decision guaranties that users' investment in learning spreadsheet programming is leveraged. Any[1] valid classical spreadsheet formula expressions is also a valid mapping expressions in SpreadATOR.

Similar to spreadsheet programming, users can input mapping formula i) independently of each other and ii) in any order. Moreover, each mapping formula input or update immediately triggers an evaluation displayed to the user for feedback.

This approach is illustrated in Figure 1. The user interface shows both the spreadsheet (on the left) and the target schema (on the right) in roughly the same way as depicted in Figure 1. We use this schematic representation rather than a screenshot because, in the same way as classical spreadsheet formulas, mapping formulas are not displayed continuously. Users may edit them upon selecting a given label of the schema and only the evaluation is permanently shown in regard of the label.

Figure 1(b) presents few examples of mapping formulas:

- `Login`="MyLogin" uses a constant expression to specify the value of `login`;
- `Id`=A1 uses a cell reference to specify the value of `Id`, in our example, the value corresponding to A1 in the spreadsheet is the string "0042";
- `Quantity`=B3:B5 uses a range expression to specify that there are 3 values in the spreadsheet representing quantities.

A similar mapping could have been achieved using purely graphical tool such as the XML mapping tool [6] built in MS Excel. However, using formulas to specify mappings offer a greater expressiveness when compared to graphical only solutions. Moreover, the added expressiveness is not obtained at the cost of new learning on the part of users since they can readily reuse their experience in programming with spreadsheet formula language in this new context. We give below few examples (expressed using MS Excel syntax) of the advantages that arise from using formulas in mapping specification:

Data transformation. Suppose that instead of `FirstName` and `LastName` as two separate attribute, the service schema expects a `FullName`, the corresponding mapping can be specified with `FullName`=A2&' '&B2.

Lists of varying size. The mapping shown in Figure 1(b) lakes generality in that the list of purchased item is of exactly 3 items. In order to obtain a mapping valid regardless of the number of rows detailing the purchase, users can use the following expression:

`OrderDetails`=OFFSET(B3, 0, 0, COUNTA(C:C), 2)

This formula returns a range starting at cell B3 and spanning 2 columns. This range is dynamic since the number of rows is computed by COUNTA(C:C) which returns the number of non-empty cells in column C. Note that this mapping replaces the two mappings expressed on `Quantity` and `ProdName` in Figure 1(b). This illustrate another feature of

[1]With the exception of matrix formulas

SpreadATOR which allows mappings to be expressed indifferently at the tuple level (i.e., on label `OrderDetails` or `OrderLine`) or at the element level (i.e., on label `ProdName` and `Quantity`).

Noncontiguous lists. Suppose that users have an existing spreadsheet where a list is divided into several categories identified by intermediate rows bearing the category title. The user needs to express the list of table parts that are to be exported as a merged list in the invocation message. A mapping expressing such situation would be similar to:

```
Quantity=B3:B5,B9:B13,B17:B30
ProdName=C3:C5,C9:C13,C17:C30
```

In addition to the possibilities natively offered by formula expressions, SpreadATOR extends the formula language with new symbols allowing mapping of various situations. Due to space restriction, it is not possible to present here these extensions.

3. CONSUMING SERVICES RESPONSES

3.1 Formula-based data importation

Data services give access to complex composite entities. However, in traditional spreadsheet applications, cells can only contain atomic values such as integer or string. Formulas are similarly limited to return only values compatible with cell content. While it may seem a natural solution, we argue that modifying spreadsheet formula language so that it accounts for complex data types is not appropriate. Indeed, for most applications and users, this language simplicity is its greatest asset and its limitations are not perceivable.

In order to account for the complexity of external data types while altering as few as possible existing spreadsheet applications, we position our system SpreadATOR as middleware for spreadsheet integration with data services.

A representation is built through a collection of formulas that are attached to spreadsheet cells. Formulas are maintained in a separate context—called the external mapping definition—which leaves untouched the original formula language and the overall behavior of the spreadsheet environment.

Figure 2 shows the mapping definition in one spreadsheet grid and its evaluation in another. In reality, the spreadsheet user only sees one grid only. Traditional formulas and SpreadATOR formulas are merged in a single interface, making the programming very intuitive to spreadsheet developers. They are oblivious of the fact the two types of formulas are maintained by different systems.

SpreadATOR formula language is meant to reflect the way resources are accessed and the model in which entities are represented. Our implementation of SpreadATOR relies on JScript.Net (.Net implementation of javascript) for formula evaluation. Thus, formula syntax (of this implementation) of SpreadATOR corresponds to that of cell `B2`. However, for accessing services built according to the REST model, a same approach could be adopted using a language in the style of XPath (as illustrated in cell `B3` Figure 2.

Both formula examples in Figure 2 return an atomic type (a string), but statements returning object references are also valid. For example, `=Customers['001']` returns a reference to an instance of customer and `=Customers` returns

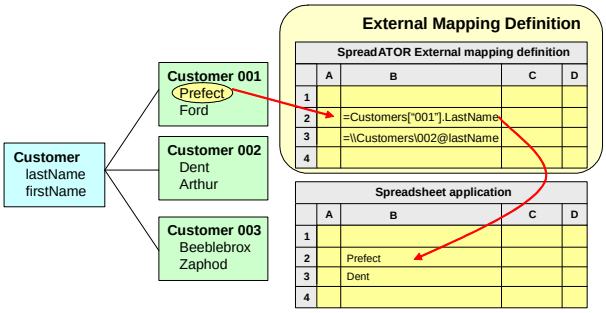


Figure 2: Formula-based representation of composite entities

a reference to the complete list of customers. The reference returned is managed by SpreadATOR; for Excel, the cell simply contains a string representation of these objects (obtained by the default transtyping given by `toString()`). The advantages of storing a reference to a composite entity in a cell are:

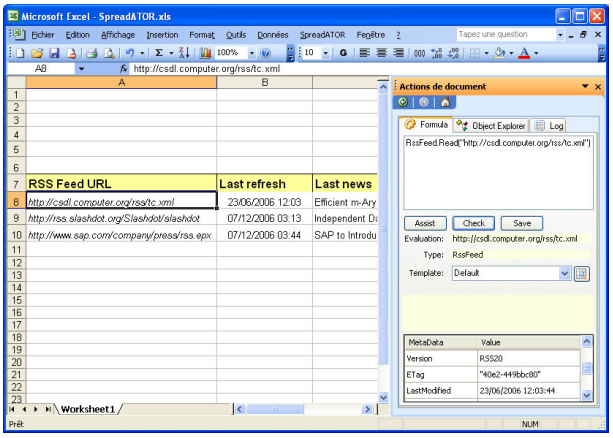
- It is now possible to refer directly to these composite objects to build their representations on the spreadsheet. For example, if `B2=Customers['001']`, we can have a formula `B3=B2.lastName`. Thus, it suffices that the content of `B2` changes (e.g. if it is replaced by a reference to customer 002), for all related formula to change accordingly. This makes formula short, easy to read and efficient to compute;
- The content of cell `B2` now has a type. It is possible to display additional information corresponding to that particular entity type and permit navigation to related entities through the template mechanism, described in the next Section.

We want to emphasize that, despite the object-like syntax of the formula language used in SpreadATOR, we do not assume any familiarity of end-users with object-oriented programming. First, users access only pre-built objects available from data services, they do not actually "create" these objects. Second, the use of formula does not preclude the complementary usage of wizards or visual assistants to generate the formulas. Traditional spreadsheet formulas are themselves often built by using wizard dialogs. The visual assistant we propose in SpreadATOR allows to build entity representation by drag&drop manipulations.

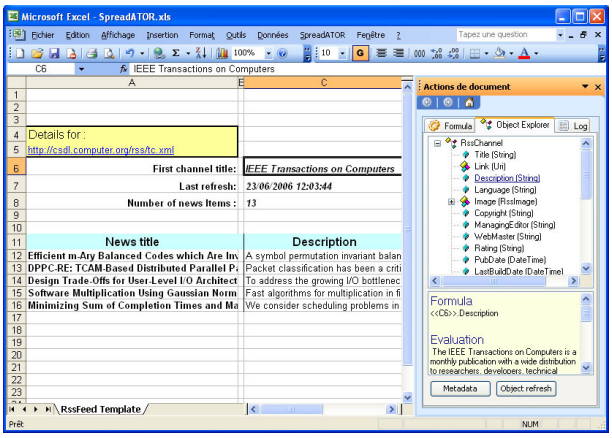
3.2 Relationship based manipulations

Our aim is to allow spreadsheet users to benefit from the rich relationships that may exist between entities. To this end, we propose a *template* mechanism. The idea of template is not new to spreadsheet and end-users are already familiar with it. The innovation of SpreadATOR is to associate templates with the type of composite objects—each type may have several templates—and allow to define a generic representation used for all instances of that type. Templates are given names and are proposed in a drop-down menu (see Figure 3(a)).

Suppose a worksheet with a formula `A1=Customers[001]`; that is, cell `A1` contains, from SpreadATOR point of view, a reference to the instance of type *Customer* that represents



(a) A worksheet accessing 3 RSS feeds



(b) Template view of one RSS feed and object explorer

Figure 3: Representing Composite Entities in Spreadsheet using SpreadATOR

the customer 001. When `A1` is selected, users can open a template associated to the type *Customer* (or create a new template for that type). An internal object named `obj` is associated to the instance referenced in `A1`. Users can use this reference to build a representation of this object.

While formulas used in a worksheet refer to external entities (e.g. formula `=Customers[001].lastName` represents an access to a particular customer), formulas used in templates references an internal object denoted `obj`. For example, in a template suited for objects of type *Customer*, we would have a formula such as `=obj.lastName`. The approach used to build the template is exactly the same as for a worksheet and relies on the same visual assistant, only the formula generated by the assistant are different.

The template defines how to represent an object `obj`. Accessing a template is equivalent to a relationship navigation since the template can display any information related to the selected instance—for example, the list of purchase orders of the selected customer.

Furthermore, SpreadATOR allows to access the customized grid representation of an object type from a worksheet that contains instances of that type. For example, suppose that a customer template called "PO details" is used to compute in cell G4 the average of POs worth more than 100$. From our worksheet example above, where cell `A1` contains a reference to customer 001, we can access the custom aggregate of the template using the formula `=template(A1,'PO details',G4)`.

This formula can easily be duplicated for all the customers present on a worksheet, simply changing the reference `A1`. In object-oriented terminology, it is as if the type *Customer* had been extended with a new method that computes the custom aggregate. When this formula is evaluated, `obj` is associated with the reference contained in `A1`. It can be seen as the SpreadATOR equivalent to keyword `this` in object oriented programming. However, `obj` stands for *"current cell composite content"*, rather than *"current instance of that class"*.

By comparison, computing this custom aggregate on a list of customers in traditional spreadsheet programming approach—that is, without resorting to another programming paradigm such as a macro language—is more complex. For example, one could import the list of customers in a worksheet and either (i) import as a large table all the purchase orders of all customers in a second worksheet or (ii) import the list of POs in a separate worksheet for each customer. In either case, one would have to rebuild the join between the list of customers and that of POs, i.e., to look for the starting and ending row corresponding to a customer (case (i)) or to retrieve the worksheet corresponding to a customer (case (ii)). The formula language of spreadsheet includes these lookup functions. But why should users have to build this join when it is readily available in the ER data model?

3.3 Meta-data management

Meta-data are not very different in nature from other information related to a given entity. They mainly differ by their semantics and usages. Meta-data are typically not values that users want to lay out on the spreadsheet because they are not essential attributes of the information accessed. They represent a complement of information, kind of a documentation; they speak about data. For example, a Business Intelligence (BI) software often provides access to Key Performance Indicator (KPI) such as "Order processing delay" which expresses in days the average time needed to process a customer purchase order. Such KPI are high level aggregates. Users need to know what they exactly mean, how they are evaluated, when they were last refreshed, how accurate they are, etc. We need meta-data (i) to always be accessible whenever we examine a KPI and (ii) not to occupy cells of their own on the worksheet—unless, of course, the specific application we build calls for it.

Thus, we propose in SpreadATOR to display meta-data separately from the worksheet in some reserved space of the user interface (see the bottom-right area in Figure 3(a)). We define meta-data as a collection of $\langle name, value \rangle$ pairs obtained from a collection of $\langle name, formula \rangle$ tuples that depend—in the same way as the template mechanism—on the type of the composite object contained in the cell. The former is used for display in a list when a cell containing a composite object is selected, while the later corresponds to a collection defined by the user where each formula refers to the selected object through the keyword `obj` introduced in the previous section. For example, end-users can de-

	C	D	E	F	G	H	I	J	K
1		ASX Gains					ASX Declines		
2	Company	Code	Price	Change		Company	Code	Price	Change
3	CSL	CSL	$88.44	$2.44		RIO TINTO	RIO	$88.72	-$2.48
4	LEIGHTON	LEI	$40.00	$1.64		JB HI-FI	JBH	$10.50	-$0.70
5	FORTESCUE	FMG	$31.50	$1.46		JUBILEE	JBM	$14.20	-$0.53
6	QBE INSUR.	QBE	$29.77	$1.23		INCITEC PV	IPL	$72.50	-$0.51
7	ST.GEORGE	SGB	$33.85	$1.05		PUBL&BROAD	PBL	$17.60	-$0.42
8									
9				Google News		Input Company Code Here:		LEI	
10		1 Weekend Best Bets: Get Lei'd Luau Party - goTriad.com							
11		2 Moldovan government to provide disadvantaged people with ... - Moldpress							
12		3 LEI Stalks MAH - Asia Corporate News Network (press release)							
13		4 There is not food crisis in Moldova, despite drought, Moldovan ... - Moldpress							
14		5 Japan urged to face WWII legacy - Taipei Times							
15		6 Outcry over choice of King sculptor - Minneapolis Star Tribune (subscription)							
16									
17				Company's Detail					
18	PROSPECT	ID	NAME			ACC_OWNER	ACC_MNGR	PHONE	
19	1138498	322590	Leighton			Peter	Peter	6 624 322300	
20	1169633	394652	Leisure house			Not assigned	Not assigned	-	
21	1214652	483712	Town group			Deavon	Deavon	(03) 4652 666	
22									

Figure 4: Mashup application for sales opportunities

fine a meta-data for type *Customer* with ⟨Last contacted, `obj.lastContactDate`⟩. When a cell containing a customer is selected, the evaluation of this formula is displayed in the bottom-right section of the screen, e.g. ⟨Last contacted, 21/06/2006⟩.

4. DEMONSTRATION

In order to identify new opportunities for selling enterprise software, one method used by sales persons is to monitor the stock price of publicly-listed companies for any discontinuity. A strong rise or fall of a company's stock is often a sign that a significant event just happened in the life of the company, and any such event may potentially be an opportunity to sell software.

For example, a sharp increase in the stock price may be the consequence of new plans to expand the business, or of the company becoming the target of an acquisition. In the first case, the company will need new software to manage its expanded operations. In the second case, sales person need to react quickly since mergers and acquisitions often translate into IT integration projects. This kind of projects is an opportunity as well as a threat since existing software could be ripped and replaced in the process. Therefore, a clear advantage can be gained from talking to the customer before competitors do.

The implementation of this scenario leads to building a form of mashup application in the spreadsheet by combining data obtained from i) a web site delivering XML documents with financial informations ii) RSS feeds of news and iii) SAP web services for accessing a CRM system. The application comprises four different parts, which can be seen in the screenshot presented in Figure 4.

The top part of the application displays the top 5 biggest increases and the top 5 biggest decreases in the stock price of companies on the Australian Securities Exchange (ASX). We obtain this information by performing an HTTP GET call to the `asx.com.au` Web site, which returns an XML document. Because the information returned is properly structured, we can display it in columns. One important column is the one displaying the stock symbol, which is a three-letter code that uniquely references a given company's stock. Since SpreadATOR comes with a built-in library for performing HTTP calls, all we have to do is drag&drop labels from the structure of the returned XML document onto the spreadsheet in order to display the different fields of the message in different columns. The resulting set of formulas is then copied in order to show only the top 5 best and worst performers of the moment.

Using this information, the sales person can see that something happened. Now, the next thing to do is to understand why. At this stage, the user may input into the cell with a yellow background the symbol of a stock he is interested in investigating. This then triggers the download of recent financial news where the stock symbol appears. These news are obtained from Google News using an RSS feed. Here again, since SpreadATOR natively supports newsfeeds as a complex entity, all we have to do is use the drag-and-drop features in order to properly arrange the display of the six most recent news items that contain the required keyword.

The bottom part of the user interface is a lookup of the company's internal CRM system that gives us information about LEI as a customer: which salesperson is responsible for that account for example, and how much software has been sold to them so far. These information are important when making phone calls: a sales person would not approach a customer in the same way if they sold them a major software upgrade last months or haven't heard from them in five years. The enterprise application used here was the SAP CRM product which exposes a number of web services.

Through this scenario, we demonstrate:

- How service invocation is achieved by building messages using formulas;
- How service responses bearing complex documents can be seen as a first-class cell value and used to built a spreadsheet representation also using formula;
- The graphical environment that allows to perform most of these manipulation by simple point and click operations;
- Access to related information in the form of meta-data such as the last refresh time/date of a data obtained by service invocation;
- Access to related entities by navigation through entity relationships using the template mechanism.

5. ACKNOWLEDGMENTS

Authors wish to thank Trang Nguyen and Xiaoping Yang for their work on the implementation of SpreadATOR.

6. REFERENCES

[1] Excel Services Overview. Microsoft Corp., 2006.
[2] G. Alonso et al. *Web Services - Concepts, Architectures and Application*. Springer-Verlag, 2004.
[3] B. Brauer. Next evolution of data integration into microsoft excel. Technical report, StrikeIron Inc., 2006.
[4] M. Carey. Data delivery in a service-oriented world: the BEA AquaLogic data services platform. In *SIGMOD'06*, pages 695–705, New York, USA, 2006.
[5] A. Halevy et al. Enterprise information integration: successes, challenges and controversies. In *SIGMOD'05*, pages 778–787, New York, USA, 2005.
[6] F. Rice. Creating XML mappings in excel 2003. Technical report, Microsoft Corp., 2005.
[7] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *VL/HCC'05*, pages 207–214, 2005.
[8] Next-generation data programming: Service data objects. Technical report, IBM, BEA, 2003.