DataSheets: A Spreadsheet-Based Data-Flow Language

Angel Lagares Lemos, Moshe Chai Barukh, and Boualem Benatallah

SOC Group, University of New South Wales, Australia
{angell,moshe,boualem}@cse.unsw.edu.au

Abstract. We are surrounded by data, a vast amount of data that has brought about an increasing need for combining and analyzing it in order to extract information and generate knowledge. A need not exclusive of big software companies with expert programmers; from scientists to bloggers, many end-user programmers currently demand data management tools to generate information according to their discretion. However, data is usually distributed among multiple sources, hence, it requires to be integrated, and unfortunately, this process is still available just for professional developers. In this paper we propose DataSheets, a novel approach to make the data-flow specification accessible and its representation comprehensible to end-user programmers. This approach consists of a spreadsheet-based data-flow language that has been tested and evaluated in a service-centric composition framework.

Keywords: Data-Flow, End-User Programming, Spreadsheets.

1 Introduction

During the next minute approximately 640,000Gb of global IP data will be transferred; by 2016, the annual global Internet traffic will surpass the zettabyte threshold, which means a growth rate of 29% yearly [1]. In 2013, the total digital data created (and replicated) will reach 4ZB[1]. Data is getting ever vaster more rapidly, yet data in turn may be combined to generate usable information. At the same time, 8% of all U.S. jobs require programming skills often needed to analyze data and generate reports relying just on simple personal productivity software [2], involving many data transferring tasks (e.g., parse and reformat strings passed between Web services) [3]. However since data is often distributed over a multitude of systems in many different formats, these tasks often remain complex requiring the ability of expert programmers.

For instance, data integration often demands the use of transformation functions, which must be defined in the data-flow. However, (i) existing techniques rely on languages such as XSLT[2] or XQuery[3], which are too complex for end-user programmers [4], and (ii) while there had been multiple attempts to simplify data

[1] http://goo.gl/9Ps63
[2] http://www.w3.org/TR/xslt20/
[3] http://www.w3.org/TR/xquery/

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 616–623, 2013.

transformation languages, they have never been integrated in service composition languages, where creating a data flow to manipulate data sets still requires a user to engage in some form of programming [5]. E.g., specifying data transformation in BPEL is, in general, programmed using native BPEL expressions.

End-users require simple, easy to use and learn visual environments for data-flow management. To address this we introduce the application of spreadsheets which brings unrivalled advantages: (i) they are naturally tidy and uncluttered, as opposed to data-flow visual languages [6], (ii) they offer a simple, but effective formula language using spatial relationships that masks users from the low-level details of traditional programming [7], (iii) a spreadsheet-user only needs to master two concepts, namely cells as variables and functions for expressing relations between cells. The benefits of using spreadsheet-based data-flow languages have been proven and its suitability to EUP is undeniable. In fact, spreadsheet is the most common EUP environment [8].

In light of the aforementioned, we propose DataSheets, a spreadsheet-based data-flow language that aims to remove the complexity of expressing data-flows, allowing users to design data integration and data quality processes without having expertise in the underlying languages. The main contributions of this paper are as follows:

- A spreadsheet-based data-flow language designed for specifying mappings between different web services in a service composition environment. Unlike traditional approaches which calls the need for programmers to be proficient in complex data-flow and transformational languages such as XSLT, XPath, etc., we have been motivated to choose a spreadsheet language since it is already familiar to a vast majority of end-user programmers.
- The implementation and evaluation of a prototype that realizes the proposed data-flow language.

2 Spreadsheet-Based Data-Flow

The specification of data-flow is a two-step process: schema matching and data transformation. Schema matching requires a target and a set of sources and data transformation demands transformation functions, both notions are included in spreadsheets, where the cells can be either sources or targets and spreadsheet functions can be applied to them. We rely on the notion of representing services as forms based on our previous work [9], whereby we adopt the following: a list of services corresponds to a sheet, where each service message (input and output) corresponds to a column, and each field corresponds to a cell. A cell accepts the following data types: integer, float, boolean, string, one-dimensional array (e.g. {1,2,A7}) and two-dimensional array (e.g. {1,2,3;A1:A3}). In contrast to spreadsheets, array values are not expanded along rows or columns; in our approach a single cell can encapsulate these types.

The specification of the data-flow is performed per service, meaning just one service message at a time, from where the data-flow for every target field is specified (for short we call this service message "target message" and the service

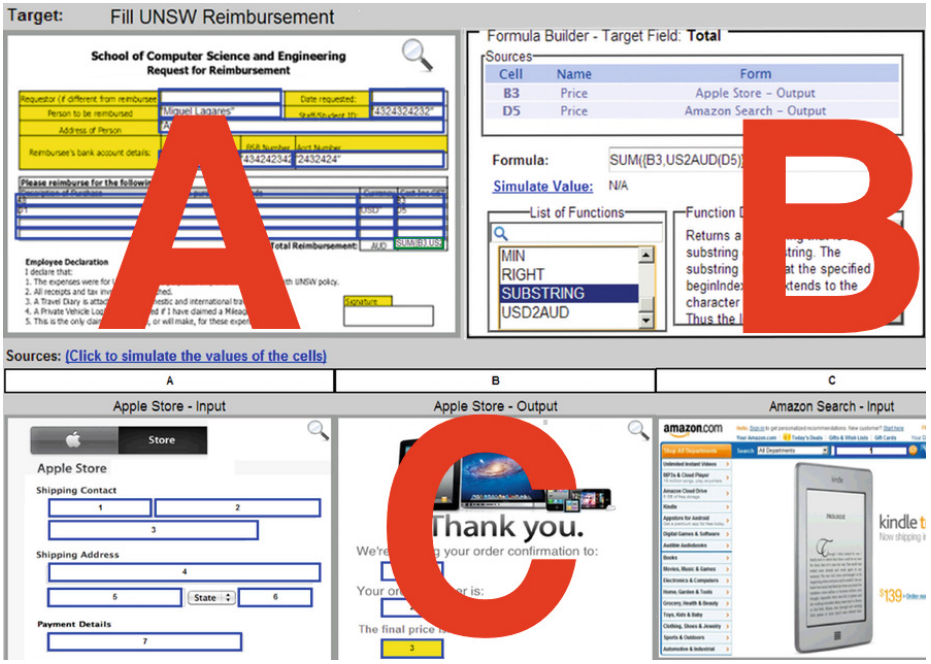

Fig. 1. Data-Flow Specification - Main Areas

that contains it "target service"). The target message has to be one of the input messages (by definition an output message cannot play this role) of the services in the composition. The sources consist of all the service message fields belonging to the services that precede the target service within the control flow. The target fields are editable cells, where the user can enter data-flow formulas. The source fields are cells grouped by messages; each message is located in a column; and the messages are grouped by services.

An example is depicted in Fig. 1, where the input message of the service "Fill UNSW Reimbursement" is the target service (section A) and three messages are shown as sources (section C). "Apple Store" service is one of the sources represented in the spreadsheet, where column "A" contains the fields of the input message and column "B" the fields of the output message. The fields are the cells of the spreadsheet and can be identified in the same manner as cells are in regular spreadsheet editors, for instance "B3" corresponds to the field "Price" of the output message of "Apple Store" service.

2.1 Formulas

Formulas describe the operations that place the result in the target field. The data-flow for target fields is to be specified by entering a formula the same way as one would do it in a spreadsheet. Formulas can contain:

- Constant Values, such as "'some text'", "5", "7.3";
- References to single fields, such as "C7", or to a range of fields, e.g., "B2:D5"

– 1-D and 2-D arrays, such as "{1,2;3,4}"
– Functions, such as "SUM", "INDEX" (e.g. INDEX({1,2;3,4},2,1)).

While references in spreadsheets are absolute, here the references to source fields are always relative. If the position of a service is modified in the control flow, the fields of that service that were acting as sources and all the formulas where they are used are updated automatically. As an example, let us consider the formula for the field "Total" shown in Fig. 1 (Section B) that contains the reference "B3" (field "Price" of "Apple Store" Output). If the service "Apple Store", previously preceding the service "Amazon Search", swaps the position with "Amazon Search", the "Apple Store" service will change the position into the list of sources and its output message will be in the column "D"; in consequence, the reference "B3" would be automatically converted into "D3".

To help the user in the task of building the formulas and maintain the consistency with the spreadsheet analogy, this approach includes the concept of a formula builder, where the user: (i) is presented a list of the sources used in the formula with all the information to clearly identify them, (ii) is entitled to edit the formula using a list of available functions and obtain feedback through tooltips, and (iii) is able to simulate the result of the formula.

3 DataSheets Implementation

DataSheets has been implemented and integrated in an EUP process composition platform, namely FormSys [9], as a proof-of-concept. The architecture presented here shows the modules related to DataSheets (Fig. 2).

Their interaction occurs as follows: A user selects a set of Web services and defines the control-flow. Then, the user specifies the data-flow via the *Spreadsheet Data-Flow* interface, which is generated by the *Spreadsheet Generator*. The *Formula Parser* evaluates the formulas by generating a tree of tokens from the

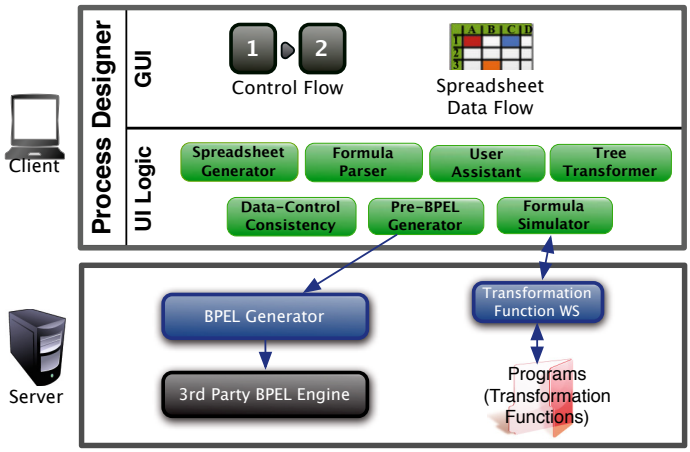

Fig. 2. DataSheets Architecture

functions, references and constants extracted; if an error is found, it will be indicated by the *User Assistant*. *Data-Control Consistency* maintains the coherence between the control and data flow by means of a combined control and data-flow directed graph that is evaluated using the Tarjan's strongly connected components algorithm [10]: in case a strongly connected component is detected, there is an inconsistency and the user is warned. The *Formula Simulator* permits the user to simulate the data-flows at the design time by means of consuming the *Transformation Function WS* with testing values. Once the data-flow is defined, the system translates it into a proprietary XML document using the *Pre-BPEL Generator* and then into BPEL using the *BPEL Generator*. Finally the BPEL code is deployed using a *3rd Party BPEL Engine*.

Regarding the technologies used for the implementation, the system can be split in two different parts: (i) the *Process Designer* and *BPEL Generator*, which include all the modules for process modeling and BPEL code generation, has been implemented in Symfony, a MVC framework for PHP, and jQuery; and (ii) the *3rd Party BPEL Engine*, which deploys and executes BPEL code, the engine used is Intalio|Server.

4 Evaluation

We have conducted a user study, where we collected the participants' experience using DataSheets techniques, which were integrated in FormSys, and we compared it with a state-of-the-art third party tool, Altova MapForce[4] (MapForce for short). We chose MapForce since it offers an interface, based on connecting elements through arrows, which is considered an EUP technique. A total of 14 participants took part in the study, of which none of them knew how to program in BPEL or any other process composition language.

Participants were required to use the tools by performing two tasks, both required the specification of complex data-flows, including the application of nested transformation functions. The first one was an introductory task to help them in familiarizing with the tools. The second one involved a more complex scenario and the user was asked to accomplish it without any other help than the one the tools provide.

After competing the tasks, they were tested in a questionnaire. Also task completion times were measured during the experiment, and notes gathered from users thinking aloud.

With the aim of evaluating the benefits of the language proposed, through this section we evaluate the veracity of the following hypothesis:

$\mathbf{H}_1$: DataSheets enables end-user programmers to specify composite services that require complex data transformation.

$\mathbf{H}_2$: DataSheets is more familiar and expressive to users.

Completion and H_1 Task 2 was completed in both tools by 85.7% of the participants, just one participant did not complete it in both tools, and another one did not complete it in MapForce.

[4] `http://www.altova.com/mapforce.html`

Table 1. Completion

Completed	Second Att.	FormSys			MapForce		
		f	%	%*	*f*	%	%*
Yes	No	4	28.6	30.8	2	14.3	16.7
	Yes	9	64.3	69.2	10	71.4	83.3
	Total	13	92.9	100.0	12	85.7	100.0
No		1	7.1		2	14.3	
Total		14	100.0		14	100.0	

f=frequency, %=percent, %*=percent over completed

To evaluate H_1 we carried out a binomial test using the one-tailed p-value. We consider that if the tool succeeds by an average greater than 50%, H_1 should be considered true (hence, $H_1 : p > 0.5$). The null hypothesis is $H_0 : p \leq 0.5$.

The value obtained from the binomial test was $p = 0.00085$. Considering a standard criterion of $\alpha = 0.05$, p is less than α, in consequence, H_0 can be rejected, which confirms that the results support H_1.

Usability and H_2 We consider that the combination of the results obtained by the user interface and functionality questions gives a reasonable measure of the tool usability. The comparison of the opinions from participants about the tools is to be used to evaluate H_2. The Wilcoxon signed rank sum test for paired samples has been applied to test H_0. The results are shown in table 2.

Table 2. Usability Comparison - Wilcoxon Signed Ranks Test

Ranks					Test Statistics	
		N	Mean Rank	Sum of Ranks		FormSys - MapForce
FormSys - MapForce	Negative Ranks	22[a]	51.89	1141.50	Z	-8.512[d]
	Positive Ranks	124[b]	77.33	9589.00	p-value (2-t)	.000
	Ties	88[c]				
	Total	234				

a: FormSys(FS)<MapForce(MF). b: FS>MF. c: FS=MF. d: Based on negative ranks

The resulting p-value (0.000) is less than the significance level, it indicates a difference statistically significant between the two groups and implies the rejection of H_0. The evidences extracted from the test results support H_2.

Discussion

The results reveal a number of positive trends in the current research and pointed out enhancements to consider for future directions:

- DataSheets facilitates defining and understanding data-flows. Quite significantly, we note that, in fact, 92.9% of the participants solved task 2 without any additional help provided; more so, even the two participants who had never programmed before were able to solve the task.
- The spreadsheet-like representation has been proved to be more familiar to end users. At the same time, the users found that the proposed language allowed them to express formulas seamlessly.
- As to matters of improvement: the participants pointed out the excessive use of vertical scrolling, which at times caused them to loose focus on the task; as well as some participants felt the types of errors and their description we not meaningful enough.

5 Related Work

Process-centric composition languages, amongst which BPEL4WS (collectively known as BPEL) is considered the prevailing standard, are intended for professional programmers (e.g., business process developers) and beyond the scope of non-programmers. Support tools for process management (e.g., Intalio BPM[5], Oracle BPEL Process Manager[6]) often provide a graphical interface from which BPEL code is generated. While these graphical aids improve the productivity of programmers, these tools remain complex for non-programmers.

Recently, authoritative research have argued for the need to integrate EUP techniques and tools in BPM/SOA platforms [11] and slowly they are emerging, specifically, business process composition platforms [12]. The most successful EUP tools for Web service composition can be found in the area of mashups. In this area, Marmite [5], introduced a linked data-flow/spreadsheet view, where formulas are column references and transformation functions are represented as new services, hence, the spreadsheet abstraction, opposite to DataSheets, is applied rather loosely. Furthermore, (i) mashups are oriented to data aggregation and they lack in techniques to provide a complete process-centric composition environment, and (ii) they still require the understanding of data-flow related programming concepts such as data and message passing [13].

In spite of the fact that spreadsheets are the most common EUP environment and they have been applied in an ample number of EUP systems [14], to the best of our knowledge, DataSheets is the first spreadsheet-based data-flow language integrated in a BPM/SOA composition environment.

6 Conclusion

In this paper we focused on the problem of data-flow modeling for EUP. Based on an approach using spreadsheets, we implemented and evaluated the approach. The key contributions are: (i) an integrated data-flow tool, which, during the

[5] `http://www.intalio.com/bpm`
[6] `http://www.oracle.com/technetwork/middleware/bpel/overview/index.html`

design time, gives the user the faculty to specify expressions to be used at runtime, that determine the value of a target field from values of source fields using a language similar to the one used to specify a cell formula in spreadsheets, and (ii) a user study conducted to evaluate the effectiveness of the proposed language, which results demonstrate that the research prototype where we applied the proposed techniques can compete and even outperform state-of-the-art tools on data-flow specification and representation.

References

1. Cisco Visual Networking Index: Forecast and methodology, 2009-2014. White paper, CISCO, vol. 2 (June 2010)
2. Ko, A.J., Myers, B.A., Aung, H.H.: Six learning barriers in end-user programming systems. In: IEEE Symposium on Visual Languages and Human Centric Computing, pp. 199–206 (2004)
3. Asavametha, A., Ayyavu, P., Scaffidi, C.: No application is an island: Using topes to transform strings during data transfer. In: International Conference on Information Science and Applications (ICISA), pp. 1–10. IEEE (2011)
4. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. The VLDB Journal 10(4), 334–350 (2001)
5. Wong, J., Hong, J.I.: Making mashups with marmite: towards end-user programming for the web. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1435–1444. ACM (2007)
6. Nunez, F., Blake, E.: Vissh: A data visualisation spreadsheet. In: Joint Eurographics-IEEE TCVG Symposium on Visualization (VisSym), pp. 209–218. The Eurographics Association (2000)
7. Jones, S., Blackwell, A., Burnett, M.: A user-centered approach to functions in excel. In: Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming, pp. 165–176. ACM Press (2003)
8. Panko, R.: Spreadsheet errors: What we know. What we think we can do. Arxiv preprint arXiv:0802.3457 (2008)
9. Weber, I., Paik, H., Benatallah, B., Gong, Z., Zheng, L., Vorwerk, C.: Formsys: form-processing web services. In: Proceedings of the 19th International Conference on World Wide Web, pp. 1313–1316. ACM (2010)
10. Tarjan, R.: Depth-first search and linear graph algorithms. In: 12th Annual Symposium on Switching and Automata Theory, pp. 114–121. IEEE (1971)
11. Casati, F.: How end-user development will save composition technologies from their continuing failures. In: Piccinno, A. (ed.) IS-EUD 2011. LNCS, vol. 6654, pp. 4–6. Springer, Heidelberg (2011)
12. Mehandjiev, N., Namoune, A., Wajid, U., et al.: End user service composition: Perceptions and requirements. In: Proceedings of the 8th IEEE European Conference on Web Services, pp. 139–146. IEEE Computer Society (2010)
13. Namoun, A., Nestler, T., De Angeli, A.: Service composition for non-programmers: Prospects, problems, and design recommendations. In: 8th European Conference on Web Services (ECOWS), pp. 123–130. IEEE (2010)
14. Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., et al.: The state of the art in end-user software engineering. ACM Computing Surveys (CSUR) 43(3), 21 (2011)