Situational data integration with data services and nested table

Yanbo Han · Guiling Wang · Guang Ji · Peng Zhang

Received: 28 February 2011 / Revised: 3 April 2012 / Accepted: 12 April 2012 / Published online: 27 April 2012
© Springer-Verlag London Limited 2012

Abstract Situational data integration is often ad hoc, involves active participation of business users, and requires just-in-time treatment. Agility and end-user programming are of importance. The paper presents a spreadsheet-like programming environment called Mashroom, which offers required agility and expressive power to support situational data integration by non-professional users. In Mashroom, various data sources are encapsulated as data services with nested tables as their unified data model both for internal processing and for external uses. Users can operate on the nested tables interactively. Mashroom also supports the basic control flow patterns. The expressive power of Mashroom is analyzed and proved to be richer than N1NF relational algebra. All the XQuery expressions can be mapped to Mashroom operations and formulas. Experiments have revealed the potentials of Mashroom in situational data integration.

Keywords Data integration · Data service · Situational application · Spreadsheet · Mashup

1 Introduction

Situational applications refer to the applications that can be constructed on the fly by business users and/or departmental IT staff to deal with transient and ad hoc requirements [1, 2]. They represent the “long-tail” part of enterprise application development and are attracting more and more attention. When data integration requests are changed or new requests are submitted on the fly, a new round of integration should be quickly performed to meet the new requirements in a just-in-time manner. Also, such user-centric applications not only process data from enterprise back-end databases, but also deal with Web data from HTML pages, web services or open APIs, which are not covered by traditional enterprise data integration architectures.

The traditional data integration approaches like data warehousing or virtual view approach are schema-focused and not adequate to deal with situational problems. They suffer from a high upfront development effort to analyze and encapsulate data sources, determine a global schema, and require precise schema mappings. Such schema-focused approaches limit the agility to cater for requirements of dynamic integration of changing data sources on the fly.

Mashup is a new application development method that allows non-professional users to build applications by combining functionalities offered by more than one source to deal with situational and ad hoc problems. Targeted at different problems, there are different kinds of mashup, such as the data mashup, process mashup, and the presentation mashup [3]. Data mashup refers to a particular mashup, which allows users to access, process, and combine data from various data sources and to publish the results as a web service, API or application. Different from the traditional enterprise information integration(EII) systems, data mashup offers better agility by providing the visual user interface to expose the
data integration capabilities to enable non-professional users to build mashups [3,5] just like Damia [2] and Yahoo! Pipes [4]. As such, data mashup becomes an effective approach to dealing with situational data integration problems to certain extent [2,4,6]. However, when serving for not only normal end users but also advanced business users for enterprise integration, the expressive power of the existing data mashup tools is to be improved to offer such capabilities as aggregation, table join as well as other complicated queries and calculations [2]. As analyzed in [5], the existing data mashup tools only support simple operations and are far from reaching the expressive power of database languages or XML languages such as SQL or XQuery.

With the service-oriented computing paradigm, the common approach of building data mashups is to encapsulate data sources as data services with a unified format and then aggregate the data services into a composite one. In order to enable users with little programming expertise to build a situational application, a promising way is to combine the service-oriented computing paradigm with the end-user programming techniques like spreadsheet programming [7]. However, the traditional spreadsheet consists of a two-dimensional array of cells and is unsuitable for displaying and manipulating complex data inside services directly. The operators and functions defined in the formula language of traditional spreadsheet cannot be employed for manipulating services directly either. Although spreadsheet programming is quite mature, it is still mainly used in processing numbers and text. It is challenging to support end-user service composition while inheriting the merits of spreadsheet programming paradigm. There is no productive and easy-to-use way for complex service composition in the existing spreadsheet environments.

The reported work is to develop a programming environment for data service mashup, which inherits the merits of spreadsheet paradigm. It should also offer agility and expressive power to enable business users to solve transient and ad hoc data integration problems. The main contributions of this paper are as follows:

- We present a spreadsheet-like programming environment called Mashroom, which offers agility to facilitate situational data integration. It inherits the merits of spreadsheet-like programming paradigm and allows users without professional programming skills to aggregate multiple data services to create a composite service interactively.
- We propose a novel situational integration pattern and a programming paradigm combining spreadsheet-like programming and programming by example. Concrete contributions include: a component model based on nested table; a composition model with a set of formally defined mashup operations; and a formula language with rich expressive power. The expressive power of Mashroom is comparable to N1NF [8,9] and XQuery [10], which overcomes the weaknesses of the existing data mashup approaches.

The paper is organized as follows: in the next section, an example is illustrated to show how Mashroom works. In Sects. 3, 4, 5, and 6, we discuss the six dimensions of Mashroom: integration pattern, programming paradigm, component model, composition model, development environment and run-time environment. We evaluate the expressive power of Mashroom and discuss its operation consistency in Sect. 7, report and analyze our experimental results in Sect. 8, and give a comparative study in Sect. 9. The paper is concluded in Sect. 10.

2 A scenario: emergency materials allocation

To illustrate how Mashroom works, we present a simplified real-world scenario in this section. In coordinating emergency aid, an important task of material management branch of Red Cross is to quickly make clear whether the inventory can satisfy the demand. The demand may change quickly according to weather status and other factors. Let us take the blanket distribution as an example: if the temperature drops sharply, the priority of blanket distribution may even be higher than that of food. So the priority and quantity of the blankets needed are subject to adjustment according to weather status and the inventory. There are three data sources in this scenario: (1) the inventory of blanket stored in two databases, BlanketRepot1 and BlanketRepot2, from two government departments (for simplicity, let us assume here they have the same data schema); (2) the quantity of the affected people provided via Web API; (3) the information of the weather status provided by official webpages. In this scenario, we assume the repositories from different cities increase their inventories according to different ratios, the office wants to calculate quickly the total amount of the blankets in all repositories. There is no existing system or tool to fulfill the above-stated tasks for the office. To keep the length of this section, we only outline the major steps in synthesizing the blanket inventory in Mashroom:

1. Import the API, SQL database tables and webpages into Mashroom. Mashroom encapsulates them as data services with a unified data model.
2. Access data from the two data services of the blanket inventory. The table instances are shown in Table 1a, b, with names Tab1 and Tab2.
3. Create a new table named Tab3 and fill the content with a union of Tab1 and Tab2, as shown in Table 1c.
4. Aggregate the rows based on the City column, as shown in Table 1d.