Automating the TANDEM Design Method in End-User Programming Environments

Steve Goschnick, University of Melbourne, Melbourne, VIC, Australia

ABSTRACT

Mashups are newly envisaged applications, made up from local information sources and processes, Web services and other distributed resources, bound together technically in some way. Interactive Development Environments (IDE) used to build mashups are becoming more accessible to end-user programmers. Design methods that end-users may apply to a given problem addressed by a mashup, are much less prevalent. This paper describes an end-user-friendly design method called TANDEM and demonstrates the use of it in detail, by way of an example: the design of a mashup of services that solves the so-called movie-cinema problem. An implementation of the newly designed movie-cinema app is then built within the DigitalFriend, an end-user programmer IDE. Furthermore, a significant part of the TANDEM design method, is then automated within the development tool itself. This automation removes the most skilled task required by TANDEM of the end-user: the automation of the process of Data Normalization. The automation applies data normalization to the initial model of components and data sources that feed into the mashup. The presentation here relies on some understanding of Data Normalization, so a simple example is presented. After this demonstrated example of the method and the implementation, the paper discusses the applicability of a model achievable by end-users using TANDEM coupled with the automated normalization process built into the IDE, versus, using a top-down model by an experienced information analyst. In conclusion, the TANDEM method combined with the automation as demonstrated, does empower an end-user to a significant degree in achieving a workable mashup or distributed application. And furthermore, the TANDEM method does have broader applicability to designing a broad class of logic programs, complementing the use of collected patterns in logic programs.

Keywords: Computer Programming, Design Method, End-User Programming, Logic Programming, Logic Programming Methodology, Mashups, Movie-Cinema Problem, People-Oriented Programming, Prolog, Web Services Orchestration

1. INTRODUCTION

End-users are more and more able to build mashups of Web services, local resources and processes, and other distributed information sources, into newly envisaged, and often personalized applications. While the tools that end-users may use have increased in number and accessibility over the last decade, the methods and design techniques that are targeted at end-user programmers, have not followed suit. This paper describes an end-user-friendly method
called TANDEM (Goschnick et al., 2006) and demonstrates the use of it in detail, by way of the design of a mashup of services that solves the so-called movie-cinema problem. An implementation of the newly designed movie-cinema app is then built within an end-user-friendly development environment called the DigitalFriend (Goschnick, 2006). While many publications targeted at end-user programmers making mashups, have promoted imperative programming languages for the task, such as JavaScript, PHP and Python (e.g., Orchard, 2005; Feiler, 2008), the DigitalFriend uses CoLoG, a built-in logic programming language. CoLoG features overlap a substantial subset of the Prolog language (Sterling & Shapiro, 1994; Colmerauer & Roussel, 1993), together with added extra-logical predicates concerned with character-based I/O and the GUI interface in order to interact with an end-user, together with some features of a Constraint Logic Language (Marriott & Stuckey, 1998). The use of logic languages is more often associated with AI (Artificial Intelligence) and agent-oriented software development environments, then it is with IDEs (Integrated Development Environments) targeting end-user programmers; nonetheless, logic languages could have a big role to play in end-user programming, hopefully foreshadowed by the approach taken here. And although the DigitalFriend is usable as a multi-agent system (MAS), it was envisaged from the outset of its development, as an IDE targeted at end-user programming (Goschnick, 2006), via a methodology grounded upon people-oriented programming (Goschnick, 2009).

Even from the early days of Prolog it was recognized as a language that could be used to bring together code, additively over time, that included both descriptive logic (data structures) and procedural logic (algorithms), as Ceri & Gottlob (1986) noted: “Prolog makes possible an integrated description of data structures (‘facts’) and algorithms (‘rules’), where the algorithms are produced and presented additively, as small ‘granules’ of the overall system.” Although the authors went on to describe incremental development on one computer, the quote remains descriptive of how we use CoLoG today, to bring together data records (facts) from multiple local and distributed Web-based sources, including Relational DBMS, often in real-time, and combine them with rules that have been devised for the purpose of a mashup, in the DigitalFriend IDE.

Given that logic programming is not widely used as compared to imperative programming (e.g., JavaScript, Python, PHP), particularly with respect to making mashups, and that some appreciation of the highly compact expressive-power of it, is useful to the reception of this paper, two small sample logic programs written in CoLoG are given here: one procedural logic, and the second one mostly descriptive logic. This first example given in Figure 1 is all procedural in its logic and quite cryptic, as procedural logic often is. Whereas the second example given in Listing 1 is mostly descriptive logic, made up by a single procedural rule at the top that is supplemented by the many facts (hundreds of data records), coming from various data sources, that follow it. Note: Those lines in Listing 1 consisting of an ellipsis only (...), indicates the omission of many similar facts, for example the lines starting with countryCurrency each represent one country, and there are 248 such lines in the full program, needed to cover all countries.

The example in Figure 1 implements the often used mortgage calculator, useful when dealing with compound interest rates on loans from banks. The algorithm comes from Marriott & Stuckey (1998: p.178). The query or goal that it is currently answering, put into English is: “I initially borrowed $148,000 at a compound interest rate of 7.5%, and so far I have made 180 monthly payments of $1250 each, so what is the current Balance I still owe?” And the program answers: $40,388.51. The figure is a screenshot of the user-interface to the CoLoG interpreter built into the DigitalFriend. It has three multi-line textfields titled: Program, Query and Results. The first 6 lines in the Program textfield are just comments describing the 5 variables, represented by capital letters P,T,I,R and B, that are the parameters in the
Capturing and Comprehending the Behavioral/Dynamical Interactions within an ERP Implementation
James R. Burns, Don G. Jung and James J. Hoffman (2011). Organizational and End-User Interactions: New Explorations (pp. 189-212).
www.igi-global.com/chapter/capturing-comprehending-behavioral-dynamical-interactions/53091?camid=4v1a

Case Study of a Patient Data Management System: A Complex Implementation in an Intensive Care Unit
www.igi-global.com/chapter/case-study-patient-data-management/4422?camid=4v1a