Navigating through Mashed-up Applications with *COMPASS*

Daniel Deutch [1], Ohad Greenshpan [1,2], Tova Milo [1]

[1] *Tel-Aviv University*
[2] *IBM Haifa Research Labs*

***Abstract*— Mashups integrate a set of complementary Web-services and data sources, often referred to as *mashlets*. We consider here a common scenario where the integrated mashlets are part of larger Web-applications, and their integration yields a set of inter-connected applications. We refer to them as *Mashed-up Applications* (abbr. *MashAPP*). The inter-connections between the mashlets enrich the individual Web-applications, but at the same time make the user navigation within them more intricate as actions in one application may affect others. To address this difficulty, we present *COMPASS*, a system that assists users in their navigation through *MashAPPs*. The system employs a novel top-k algorithm to propose users the most effective navigation paths for their specified goals. The suggestions are continually adapted to choices taken by the users while navigating.**

I. Introduction

Mashups integrate a set of complementary Web services and data sources, often referred to as *mashlets*. For instance, consider the following simple components: (1) a patient's personal list of prescribed drugs, (2) a pharmacies directory and (3) a map service. A mashup may glue these mashlets together by feeding the drugs list to the (search facility of the) pharmacies directory, to get a list of pharmacies offering this drug, and feeding the addresses of retrieved pharmacies to the map service, to present their location on a map. Such connections are called *Glue Patterns* (GPs) [1].

Mashlets are typically viewed as isolated "black box" components [1], [2], [3]. In real-life, however, such services are often incorporated within larger applications. For instance, the patient drugs list may be part of an Electronic Health Record application (EHR); the pharmacies directory may be part of a pharmaceutical Web site, etc. The gluing of mashlets, in this case, yields a set of inter-connected *Mashed-up Applications* (abbr. *MashAPP*). Users of *MashAPPs* may navigate, in parallel, in several, *interacting* applications. For instance, consider a patient wishing to find out where her prescribed drugs are sold. She could navigate *separately* within the EHR, pharmacies, and map applications: this may require her to login to her EHR account, retrieve the prescribed drugs, then login to the pharmaceutical application, manually searching for pharmacies that offer these drugs, then turn to the map service and repeatedly seek for the location of relevant pharmacies. Alternatively, she can exploit the *MashAPP* inter-connections to achieve her goal much faster: she may still need to first login to her account at the EHR and at the pharmaceutical application, due to security constraints, but now a single click on each prescribed drug feeds the data to the pharmaceutical application, that retrieves pharmacies offering this drug, and the pharmacies locations will instantly appear on the map.

The above example illustrates two connections within the *MashAPP*, but there may be many others (e.g. information on pharmacies offering deals may be found in medical forums, payment may be done online, etc.). This implies that the number of possible relevant navigation flows in a *MashAPP* may be very large (even in a single application, the number of flows is large [4] and inter-connections between the applications further increase it). Some of these navigation options are significantly better than others (e.g. save work, induce less errors, etc.), but identifying them may be non-trivial.

To address this difficulty, we present here *COMPASS*, a system that assists users in their navigation through *MashAPPs*. Users of *COMPASS* are presented with a flow-map of the *MashAPP*, namely an abstract graphical representation of the *mashlets* within each application, their logical flow and the inter-connections between mashlets. By clicking on this flow-map, users may choose goals (points within the applications) which they want to reach during navigation (e.g. view prescribed drugs in EHR, purchase drugs, etc.). These user goals are compiled into a query, evaluated over the *MashAPP* specification. The system then computes, and presents to the user, the top-k recommended navigation flows within the *MashAPP*. Two ranking metrics are currently employed in *COMPASS*: the first reflects the *popularity* of flows among users - actions done by previous users sharing goals similar to these of the current user, are likely to be of interest to her as well; the second ranking metric measures the incurred navigation effort (number of clicks and the amount of input required from the user) - users often prefer flows that minimize their work. We note however that our underlying top-k algorithm is generic and we do not place any restrictions on the ranking metric except that it satisfies standard notions of monotonicity.

The user then continues her navigation in the *MashAPP*, taking into account the presented recommendations, but may also follow paths different than those proposed by the system. In this case, new recommendations, consistent with the actual choices made by the user, are automatically computed. Namely, *COMPASS* dynamically proposes new top-k continuations that are up to date with the user current navigation.

Demonstration Scenario: We demonstrate the operation of *COMPASS* on a real-life patient portal developed at IBM [5]. The portal integrates many real-life applications, including a personalized Electronic Health Record, Pharmaceutical Web sites, map applications, etc, forming a *MashAPP*. In our

demonstration, we will first present the original portal, ask the audience to navigate within it to achieve some particular goals, and show the difficulties encountered during navigation. Then we shall turn on *COMPASS* to assist the users in navigation: we will present the *MashAPP* flow-map and ask users to choose navigation goals by clicking on a set of points within it. *COMPASS* will then provide a top-k list of relevant navigation flows (displayed along-side the application) and we shall exemplify navigation within the *MashAPP* following the recommendations. We will also show the adaptive nature of *COMPASS* when different navigation paths are chosen. Finally, we will compare the navigation courses, with and without *COMPASS* (in terms of number of clicks, required user input, time spent), illustrating the improved user experience.

As a last part of the demonstration, we allow users to look under the hood of our system. Specifically, we will show queries constructed out of the user requests and illustrate main features of our adaptive top-k query evaluation algorithm.

II. Technical Background

We provide here some technical background on our model for *MashAPPs* and for queries over their navigation flows. These serve as the basis for *COMPASS*.

Previous works have suggested models for mashups connecting atomic services [1], [3], [6] and for (Web) applications [7], [8], [9] and their execution. Our model for *MashAPPs* integrates 4 main ingredients from these models: Mashlets and Glue Patterns adapted from [1], and application graphs and their flows adapted from [7]. We informally explain these ingredients and the way they are put together.

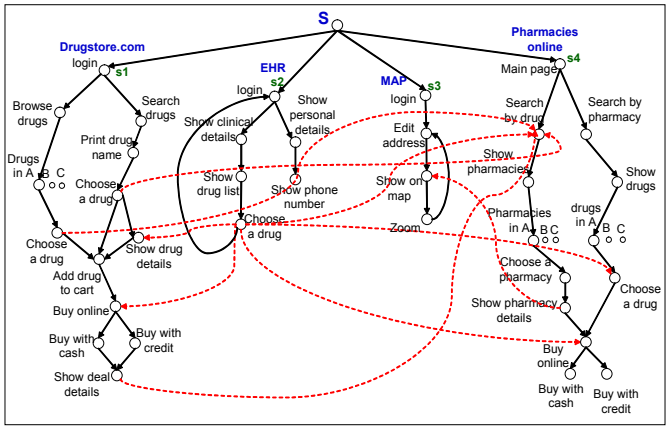

Fig. 1. Real-Life *MashAPP*

A. Mashlets, Glue Patterns, and Applications

A *mashlet* is a module implementing some functionality and supporting an interface of input and output variables. Its input variables must be fed for it to operate properly. Following [1], we model the interface of mashlets as a set of relations. *Glue Patterns* (GP for short) describe which output data is fed into which input relation and when (using a set of active rules). Then, following [7], we model a *Web-Application* as a node-labeled, directed graph describing a set of mashlets and the flow thereof. Each application has a distinguished start point, with no in-going edges, and may have some nodes marked as *accepting*. Last, a *MashAPP* is a set of Web applications, accompanied by a set of GPs gluing some mashlets and possibly crossing applications. Fig. 1 depicts a graphical abstract view of a part of real-life *MashAPP*, namely a patient portal [5]. This partial *MashAPP* consists of four applications (Drugstore.com, Electronic Health Record, a Map application and Pharmacies Online). Regular edges detail the logical flow within each application (e.g. in the Map application, the "Show on map" mashlet may receive its input from its ancestor "Edit address"). Dashed edges represent cross-applications GPs (e.g. the same mashlet may also receive input from "Show pharmacy details" in Pharmacies online).

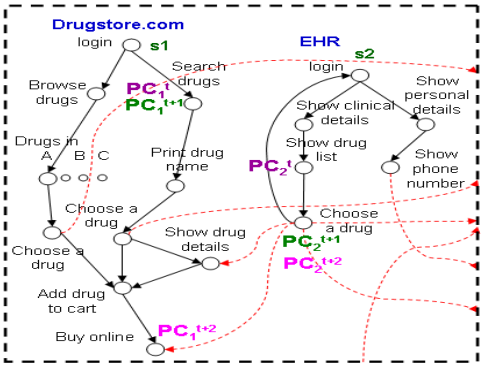

Fig. 2. Navigation Flow

B. Navigation Flows

A *Navigation Flow* is an actual instance of navigation within a *MashAPP*, consisting of a sequence of *navigation steps*. Intuitively, a valid flow at a single stand-alone application starts at its starting point and follows the application edge relation, at each step activating a following mashlet, with its input being fed either by the output of some previously activated mashlets, or by the user. We model this using a *Program Counter* (PC) signifying the current location (node) of the flow. A navigation flow in a *MashAPP* induces a parallel navigation in all participating applications and is captured by a vector of *Program Counters*, one for each application.

In the absence of GPs, a navigation step in a *MashAPP* is simply a single step in one application, updating the corresponding PC and keeping all others intact. However, GPs allow mashlets to receive some of their input values from different applications. Thus, a mashlet m in an application A may be activated even if the PC of A has not reached it, but rather the *combination* of the output of its predecessors in A (or of input provided by the user) along with that of some mashlet (in application B) that is glued to it, feeds its required input. In this case, we allow the PC of A to "jump" to m, continuing the flow from that point.

For instance, a partial navigation flow is depicted in Fig. 2, where PC_m^t denotes the PC of application m at time t of navigation. Focusing on the "Buy online" mashlet in Drugstore.com, we assume that its required input consists of user login information (as only registered users can buy drugs using this service), as well as a requested drug name. We note that at time $t+1$, the user navigation had advanced in the EHR application to "Choose a drug" node, supplying to

"Buy online" the name of a chosen drug. Since the PC at Drugstore.com already passed the "login" mashlet, user login information was also provided, thus the flow of Drugstore.com "jumps", at time $t+2$, to "Buy online".

We say that a navigation flow in a *MashAPP* is *accepting* if it reaches an accepting state in all applications for which such accepting state is defined.

C. Top-k qualifying Navigation flows

We have briefly described our model for *MashAPPs*, and we now turn to the analysis of their flows. As mentioned in the Introduction, our system allows users to specify points of interest within the *MashAPP*. This specification is compiled into a *navigation pattern*, an adaptation of tree patterns, common for querying XML, to *MashAPP* graphs. These navigation patterns are used to filter, out of all possible navigation flows, those of interest. As the set of results may still be large, we introduce *weights* over all flow edges and GPs (e.g. capturing popularity, or number of incurred user clicks). We define a flow weight as the aggregated weight over all choices and GPs within it, and identify the *top-k* weighted navigation flows out of those corresponding to the pattern.

We can show that such top-k query evaluation is $\sharp P$-hard (data complexity) by reduction from 3-SAT. Nevertheless, we can handle this intractability in practical cases: a finer analysis shows that the evaluation complexity must only be exponential in c_1, the number of *GPs* in-between applications, and c_2, the number of applications that contain accepting states. Indeed, we constructed a Dijkstra-style, Dynamic Programming algorithm of time complexity $O(min(|s| * c_1! * 2^{c_1}, |s|^{c_2}))$, with $|s|$ being the total size of the *MashAPP*.

To gain intuition on practical sizes of the exponent in the above formula, 99% of the mashups in ProgrammableWeb.com [10], the largest online Mashup repository, contain at most 6 applications (i.e. $c_2 \leq 6$), thus the complexity is in effect polynomial (bounded by $|s|^{c_2}$) for practical needs (we found no published statistics for c_1, but as the complexity is the minimum of $|s|^{c_2}$ and $|s| * c_1! * 2^{c_1}$, $|s|^{c_2}$ is an upper bound).

D. Related Work

Previous work on mashups focused mostly on tools that assist *developers* in mashups *design and construction* [1], [3], [11]. The present work takes a different angle, assisting *users* in *using* the constructed mashups. Works on Web-service composition also typically aim at easing the composition task [12], or at *statically* verifying its correctness [9]. Our use of Program Counters resembles the model of Petri Nets [13]. However, works analyzing this model typically focus on verification rather than recommendations, and additionally suffer from very high complexity. In contrast, our model is tailor-made for mashups, and while having weaker expressive power, it allows efficient query evaluation. The work in [4] studied recommendations for navigation in a *single Web-site*, but did not consider interactions between actions taking place in parallel, even in a single-application context. The inter-connections between different applications in a *MashAPP* introduce further computational difficulty, addressed by *COMPASS*.

III. System Overview

We next present a brief overview of the system architecture. *COMPASS* is developed on top of Mashup Server developed in IBM Haifa Research Labs. The server allows to compose mashups (and thereby *MashAPPs*) and use them. The architecture of *COMPASS* is depicted in Fig. 3. We describe below the main system components and explain how they work together.

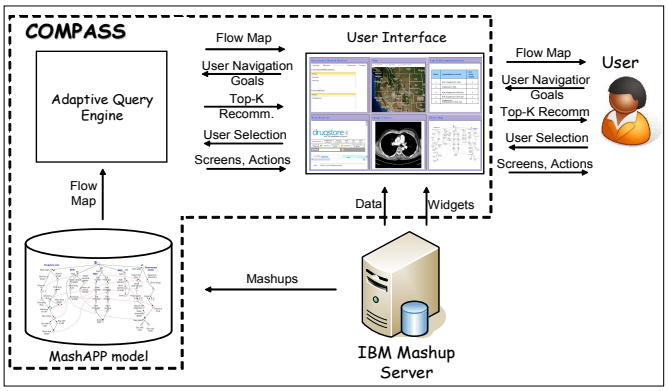

Fig. 3. System Architecture

COMPASS UI: This component is the user interface for navigating through *MashAPPs*. To construct the UI of *COMPASS*, we used as a basis the user interface of the Mashup Server, and enhanced it with plug-ins enabling the user to (1) view the flow-map of a given *MashAPP*, (2) set navigation goals; (3) obtain from *COMPASS top-k* navigation recommendations and (4) examine various recommendation by clicking on them and viewing their (highlighted) image on the *MashAPP* flow-map (see Fig. 5).

MashAPP abstract Model: The *MashAPP* model, stored in the *COMPASS* database, includes the flow-map of each participating application, as well as the GPs connecting them. The first component, namely the flow-map for each application, was manually configured for all applications participating in the demonstration, following their logical structure. We note however that many Web applications are specified in declarative languages such as BPEL [14], allowing automated extraction of their flow-map. The GPs connecting the applications mashlets were retrieved from Mashup Server, along with input and output specifications of each mashlet. Last, we consider the construction of a weight function over flows. We demonstrate here two weight functions: the first weigh flows by the total number of navigation steps (user clicks and input) taken, and the second captures the total popularity of user choices. Weights accounting for navigation steps are easily obtained from the applications flow-map; the popularity weights were obtained as follows: for flow edges connecting services of a Web application, we obtain information about transition popularity by an experimental study, collecting and statistically analyzing user logs. For GPs, we employ the *MatchUp* system [1] that computes popularity ranks for connections in-between two mashlets.

MashAPP Adaptive Query Engine: This last component computes and provides users with recommendations. This

engine, querying the abstract model Database, is incorporated as a plug-in to the *Mashup Server*. Queries are received through the *COMPASS* UI, directed to the Query Engine that processes them and returns the result back to the *COMPASS* UI for presentation. Furthermore, the Query Engine remains active throughout the navigation process, to receive from the UI a report on each user navigation step. Then, new recommendations consistent with the user choices are computed and sent to the UI, and so forth.

IV. Demonstration Scenario

We demonstrate a scenario that exemplifies how *COMPASS* significantly eases the navigation in *MashAPPs*. The *MashAPP* depicted in Fig. 4 is a real-life patient portal [5] developed in IBM, enabling users to manage data, services and applications relevant to their health. The mashlets composing the portal include, among others, the patient Electronic Health Record (EHR) storing the history of the patient diagnoses, treatments and status; applications that suggest medical-related online services, such as purchasing drugs and finding details about pharmacies; personal applications such as a calendar, a map and an image viewer; and applications that enable communication and collaboration, such as SMS and online messaging. We use real-life applications, but the underlying personal and medical data is synthetic due to confidentiality constraints. The applications in the demonstrated *MashAPP* are strongly interconnected, inducing many possible ways to address a given goal and making the choice of best navigation difficult.

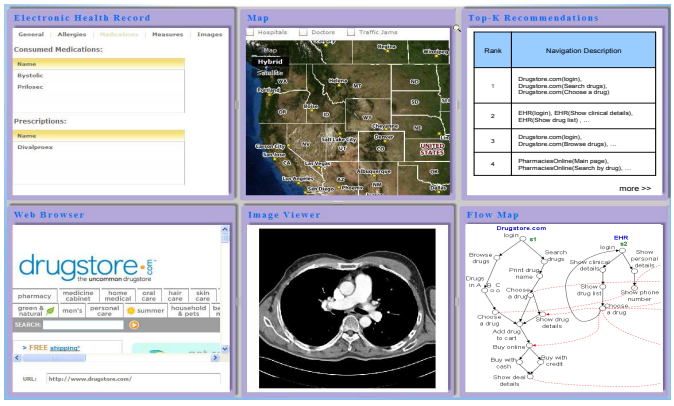

Fig. 4. *COMPASS* Top-K Recommendations

We start the demonstration by showing the flow-map of the patient portal applications along with the inter-connections between them, and present the audience with the following goal: order, from some nearby pharmacy, your prescribed drug, and show the chosen pharmacy location on the map to allow pick-up of the drug. We then allow users to navigate within the patient portal, in two modes. In the first, the user navigates by herself, with no guidance from *COMPASS*. While doing so, the system will collect information on the navigation quality, such as the number of user clicks, the popularity of her choices according to previous users choices, and the proximity of her chosen pharmacy to the conference venue. Then, we turn on *COMPASS*. The user will now be able to view the portal flow-map alongside the applications (see Fig. 4). By clicking on nodes in the flow-map, she will define navigation goals and have *COMPASS* assisting her in obtaining them. The system will provide recommendations on navigation flows, in a textual form; the user may click on a recommendation to have the relevant navigation flow lit up on the flow-map (see Fig. 5, where F denotes flow-paths and GP denotes Glue Patterns). The user then continues her navigation in the *MashAPP*, taking into account the presented recommendations, but may also follow paths different than those proposed by the system, in the latter case the system will adapt its recommendations to the actual choices made. We will collect here the same statistical data on the navigation quality as before. At the end of the process, we will present to the user a comparison of her navigation with and without the assistance of *COMPASS*, in terms of the collected statistics.

To conclude the demonstration and explain the top-k algorithm underlying *COMPASS*, we will pick one of the suggested navigation flows and explain its computation.

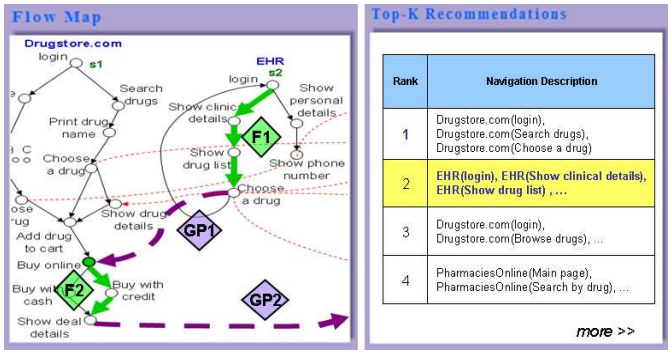

Fig. 5. Recommending Navigation Flows

References

[1] O. Greenshpan, T. Milo, and N. Polyzotis, "Autocompletion for Mashups," 2009, to appear in VLDB '09.

[2] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding Mashup Development," *IEEE Internet Computing*, vol. 12, no. 5, 2008.

[3] B. Lu, Z. Wu, Y. Ni, G. Xie, C. Zhou, and H. Chen, "sMash: semantic-based mashup navigation for data API network," in *WWW '09*, 2009.

[4] D. Deutch, T. Milo, and T. Yam, "Goal Oriented Website Navigation for Online Shoppers," 2009, to appear in VLDB '09.

[5] O. Greenshpan, K. Kveler, B. Carmeli, H. Nelken, and P. Vortman, "Towards Health 2.0: Mashups To The Rescue," in *NGITS '09*, 2009.

[6] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, "A framework for rapid integration of presentation components," in *WWW '07*, 2007.

[7] D. Deutch and T. Milo, "Type inference and type checking for queries on execution traces," in *VLDB '08*, 2008.

[8] D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi, "Automatic Service Composition and Synthesis: the Roman Model," *IEEE Tech. DE*, 2008.

[9] A. Deutsch, L. Sui, V. Vianu, and D. Zhou, "Verification of communicating data-driven web services," in *PODS '06*, 2006.

[10] "Programmbleweb," http://www.programmableweb.com/.

[11] R. J. Ennals and M. N. Garofalakis, "MashMaker: mashups for the masses," in *SIGMOD '07*, 2007.

[12] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella, "Automatic composition of transition-based semantic web services with messaging," in *VLDB '05*, 2005.

[13] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. of IEEE*, vol. 77, no. 4, 1989.

[14] "Business Process Execution Language for Web Services," http://www.ibm.com/developerworks/library/ws-bpel/.