On the Systematic Development of Domain-Specific Mashup Tools for End Users

Muhammad Imran, Stefano Soi, Felix Kling, Florian Daniel,
Fabio Casati and Maurizio Marchese

Department of Information Engineering and Computer Science
University of Trento, Via Sommarive 5, 38123, Trento, Italy
lastname@disi.unitn.it

Abstract. The recent emergence of mashup tools has refueled research on *end user development*, i.e., on enabling end-users without programming skills to compose their own applications. Yet, similar to what happened with analogous promises in web service composition and business process management, research has mostly focused on technology and, as a consequence, has failed its objective. In this paper, we propose a *domain-specific* approach to mashups that is aware of the terminology, concepts, rules, and conventions (the domain) the user is comfortable with. We show what developing a domain-specific mashup tool means, which role the mashup meta-model and the domain model play and how these can be merged into a domain-specific mashup meta-model. We exemplify the approach by implementing a mashup tool for a specific domain (research evaluation) and describe the respective user study. The results of the user study confirm that domain-specific mashup tools indeed lower the entry barrier to mashup development.

1 Introduction

Mashups are typically simple web applications that, rather than being coded from scratch, are developed by integrating and reusing available data, functionalities, or pieces of user interfaces accessible over the Web. ***Mashup tools***, i.e., online development and runtime environments for mashups, ambitiously aim at enabling non-programmers to develop their own applications. The mashup platforms developed so far either expose too much functionality and too many technicalities, so that they are powerful and flexible but suitable only for programmers, or only allow compositions that are so simple to be of little use for most practical applications. Yet, being amenable to non-programmers is increasingly important, as the opportunity given by the wide range of applications available online and the increased flexibility that is required in both businesses and personal life management raise the need for situational applications.

We believe that ***the heart of the problem*** is that it is impractical to design tools that are *generic enough* to cover a wide range of application domains, *powerful enough* to enable the specification of non-trivial logic, and *simple enough* to be actually accessible to non-programmers. At some point, we need to give

up something. In our view, this something is generality. Giving up generality in practice means narrowing the focus of a design tool to a well-defined *domain* and tailoring the tool's development paradigm, models, language, and components to the specific needs of that domain only.

As an example, in this paper we report on a mashup platform we specifically developed for the domain of research evaluation, that is, for the assessment of the performance of researchers, groups of researchers, departments, universities, and similar. There are no commonly accepted criteria for performing such analysis in general, and evaluation is highly subjective. Computing evaluation metrics that go beyond the commonly adopted h-index is still a complex, manual task that is not adequately supported by software instruments. In fact, computing an own metric may require extracting, combining, and processing data from multiple sources, implementing new algorithms, visually representing the results, and similar. In addition, the people involved in research evaluation are not necessarily IT experts and, hence, they may not be able to perform such IT-intensive tasks without help. In fact, we may need to extract, combine, and process data from multiple sources and render the information via visual components, a task that has all the characteristics of a data mashup.

In this paper, we champion the notion of ***domain-specific mashup tools*** and describe what they are composed of, how they can be developed, how they can be extended for the specificity of any particular application context, and how they can be used by non-programmers to develop complex mashup logics within the boundaries of one domain. Specifically, (1) we provide a *methodology* for the development of domain-specific mashup tools, defining the necessary concepts and design artifacts; (2) we detail and exemplify all *design artifacts* that are necessary to implement a domain-specific mashup tool; (3) we apply the methodology in the context of an *example mashup platform* that aims to support research evaluation, (4) we perform a *user study* in order to assess the viability of the developed platform.

Next we outline the methodology we follow to implement the domain-specific mashup tool. In Section 3 we briefly describe the actual implementation of our prototype tool, and in Section 4 we report on our preliminary user study. In Section 5, we review related works. We conclude the paper in Section 6.

2 Methodology

Our development of a specific mashup platform for research evaluation has allowed us to conceptualize the necessary tasks and to structure them into the following ***methodology*** steps:

1. Definition of a *domain concept model* (CM) to express domain data and relationships. The domain concepts tell the mashup platform what kind of *data objects* it must support. This is different from generic mashup platforms, which provide support for generic data formats, not specific data objects.
2. Identification of a generic *mashup meta-model* (MM) that suits the composition needs of the domain. A variety of different mashup approaches, i.e.,

meta-models, have emerged over the last years and before focusing about domain-specific features, it is important to identify a meta-model that accommodates the domain processes to be mashed up.

3. Definition of a *domain-specific mashup meta-model*. Given a generic MM, the next step is understanding how to inject the domain into it. We approach this by specifying and developing:
   (a) A *domain process model* (PM) that expresses classes of domain activities and, possibly, ready processes. Domain activities and processes represent the dynamic aspect of the domain.
   (b) A *domain syntax* that provides each concept in the domain-specific mashup meta-model (the union of MM and PM) with its own symbol. Domain concepts and activities must be represented by visual metaphores conveying their meaning to domain experts.
   (c) A set of *instances of domain-specific components*. This is the step in which the reusable domain-knowledge is encoded, in order to enable domain experts to mash it up into new applications.
4. *Implementation* of the domain-specific mashup tool (DMT) as a tool whose expressive power is that of the domain-specific mashup meta-model and that is able to host and integrate the domain-specific activities and processes.

In the next subsections, we expand each of these steps.

2.1 The Domain Concept Model

The ***domain concept model (CM)*** is obtained via interactions between an IT expert and a domain expert. We represent it as ER diagram or XSD schema. It describes the *conceptual entities* and the *relationships* among them, which, together, constitute the domain knowledge. For example in the chosen domain we have *researchers, publications, conferences, metrics, etc.* The core element in the evaluation of scientific production and quality is the *publication*, which is typically published in the context of a specific *venue*, e.g., a conference or journal, and printed by a *publisher*. It is written by one or more *researchers* belonging to an *institution*.

2.2 The Generic Mashup Meta-Model

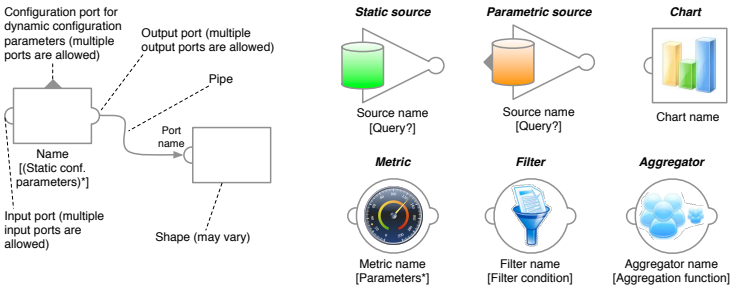
We first define a generic mashup meta-model, which may fit a variety of different domains, then we show how to define the domain-specific mashup meta-model, which will allow us to draw domain-specific mashup models. Specifically, the generic ***mashup meta-model (MM)*** specifies a *class* of mashups and, thereby, the *expressive power*, i.e., the concepts and composition paradigms, a mashup platform must know in order to support the development of that class of mashups. Thus the MM implicitly specifies the expressive power of the mashup platform class. Identifying the right features of the mashups that fit a given domain is therefore crucial. For our domain, we start from a very simple MM, both in terms of notation and execution semantics, which enables end-users to model their own mashups. Indeed, it can be fully specified in one page:

- A ***mashup*** $m = \langle C, P, VP, L \rangle$, consists of a set of *components* C, a set of data *pipes* P, a set of view ports VP that can host and render components with own UI, and a *layout* L that specifies the graphical arrangement of components.
- A ***component*** $c = \langle IPT, OPT, CPT, type, desc \rangle$, where $c \in C$, is like a task that performs some data, application, or UI action. Components have *ports* through which pipes are connected. Ports can be divided in *input* (IPT) and *output* ports (OPT), where input ports carry data into the component, while output ports carry data generated by the component. Each component must have at least either an input or an output port. Components with no input ports are called *information sources*. Components with no output ports are called *information sinks*. Components with both input and output ports are called *information processors*.*Configuration* ports (CPT) are used to configure the components. They are typically used to configure filters or to define the nature of a query on a data source. The configuration data can be a constant (e.g., a parameter defined by the end user) or can arrive in a pipe from another component. Conceptually, constant configurations are as if they come from a component feeding a constant value. The type (*type*) of the components denotes whether they are *UI components*, which display data and can be rendered in the mashup, or *application components*, which either fetch or process information. Components can also have a description *desc* at an arbitrary level of formalization, whose purpose is to inform the user about the data the components handle and produce.
- A ***pipe*** $p \in P$ carries data (e.g., XML documents) between the ports of two components, implementing a data flow logic. So, $p \in IPT \times (OPT \cup CPT)$.
- A ***view port*** $vp \in VP$ identifies a place holder, e.g., a DIV element or an IFRAME, inside the HTML template that gives the mashup its graphical identity. Typically, a template has multiple place holders.
- Finally, the ***layout*** L defines which component with own UI is to be rendered in which view port of the template. Therefore $l \in C \times VP$.

In the model above there are *no variables* and *no data mappings*. This is at the heart of enabling end-user development as this is where much of the complexity resides. It is unrealistic to ask end-users to perform data mapping operations. Because there is a CM, each component is required to be able to process any document that conforms to the model.

The ***operational semantics*** of the MM is as follows: execution of the mashup is *initiated* by the user. All the components that are *ready* for execution are identified. A component is ready when all the input and configuration ports are filled with data, that is, they have all necessary data to start processing. All ready components are *executed*. They process the data in input ports, consuming the respective data items form the input feed, and generate output on their output ports. The execution proceeds by identifying ready components and executing them, until there are no components to be executed left.

Developing mashups based on this meta-model, i.e., graphically composing a mashup in a mashup tool, requires defining a ***syntax*** for the concepts in the



(a) Basic syntax for the concepts in the mashup meta-model that are to be exposed to the user. Data mappings are configured in a dedicated pop-up window.

(b) Domain-specific syntax for the concepts in the domain-specific meta-model extension

Fig. 1. Generic and domain-specific syntax for research evaluation

MM. In Figure 1(a) we map the above MM to a basic set of generic graphical symbols and composition rules. In the next section, we show how to configure domain-specific symbols.

2.3 The Domain-Specific Mashup Meta-Model

The mashup meta-model (MM) described in the previous section allows the definition of a class of mashups that can fit into different domains. Thus, it is not yet tailored to a specific domain. Now we want to push the domain into the mashup meta-model. The next step is therefore understanding the dynamics of the concepts in the model, that is, the typical classes of processes and activities that are performed by domain experts. What we obtain from this is a *domain-specific mashup meta-model*. Each domain-specific meta-model is a specialization of the mashup meta-model along three dimensions: (i) domain-specific activities and processes, (ii) domain-specific syntax, and (iii) domain instances.

The ***domain process model (PM)*** describes the classes of *processes* or *activities* that the domain expert may want to mash up to implement composite, domain-specific processes. Operatively, the PM is again derived by specializing the generic meta-model based on interactions with domain experts. This time the topic of the interaction is aimed at defining classes of components, their interactions and notations. In the case of research evaluation, this led to the identification of the following classes of activities, i.e., classes of components: *source extraction*, *metric computation*, *filtering*, and *aggregation* activities.

A possible ***domain-specific syntax*** for the classes in the PM is shown in Figure 1(b). Its semantic is the one described by the MM in Section 2.2.

A set of ***instances*** of domain activities must be implemented, providing concrete mashup components. For example, the *Microsoft Academic Publications* component is an instance of *source extraction* activity with a configuration port (*SetResearchers*) that allows the setup of the researchers for which publications are to be loaded from Microsoft Academic.

3 The ResEval Mash Tool

The ResEval Mash platform is composed of two parts, i.e., client side and server side. The heart of the platform is the *mashup execution engine* on the client side, which support client-side processing, that is, it controls data processing on the server from the client. The engine is responsible for running a mashup composition, triggering the component's actions and managing the communication between client and server. The client side *composition editor* (shown in Figure 2) provides the mashup canvas and a list of components from which users can drag and drop components onto the canvas and connect them. The composition editor implements the *domain-specific mashup meta-model* and exposes it through the *domain syntax*. The platform also comes with a *component registration interface* for developers to set up and configure new components for the platform. On the server side, we have a set of RESTful web services, i.e., the *components services*, *authentication services*, *components and composition repository services*, and *shared memory services*. Components services allow the invocation of those components whose business logic is implemented as a server-side web service. These web services, together with the client-side components, implement the *domain process model*. Authentication services are used for user authentication and authorization. Components and composition repository services enable CRUD operations for components and compositions. Shared memory services provide an interface for external web services (i.e., services which are not deployed on our platform) to use the shared memory. The *shared memory manager* provides and manages a space for each mashup execution instance on the server side. The *common data model (CDM) module* implements the *domain concept model* (CM) and supports the checking of data types in the system. CDM configures itself using an XSD (i.e., an XML schema representing domain concept model). All services are managed by a *server side engine*, which fulfills all requests coming from the client side. A demo of ResEval Mash is described in [3] and a prototype is available online at `http://open.reseval.org/`.

4 User Study and Evaluation

In order to evaluate our domain-specific mashup approach, we conducted a user study with 10 users. Participants covering a broad range of domain and technical expertise were invited to use ResEval Mash. At the beginning participants were asked to fill in a questionnaire reporting their computing skills and to watch a video tutorial followed by a set of tasks to complete.

Overall, the tool was deemed to be usable and the participants were comfortable using it. Independently of their level of computing knowledge, all participants were able to accomplish the tasks with minimal or no help at all. The only visible difference was a different level of confidence in task execution. IT experts appeared to be more confident during the test. The results of our study indicate real potential for the domain-specific mashup approach to allow people with no computing skills to create their own applications. The definition of the

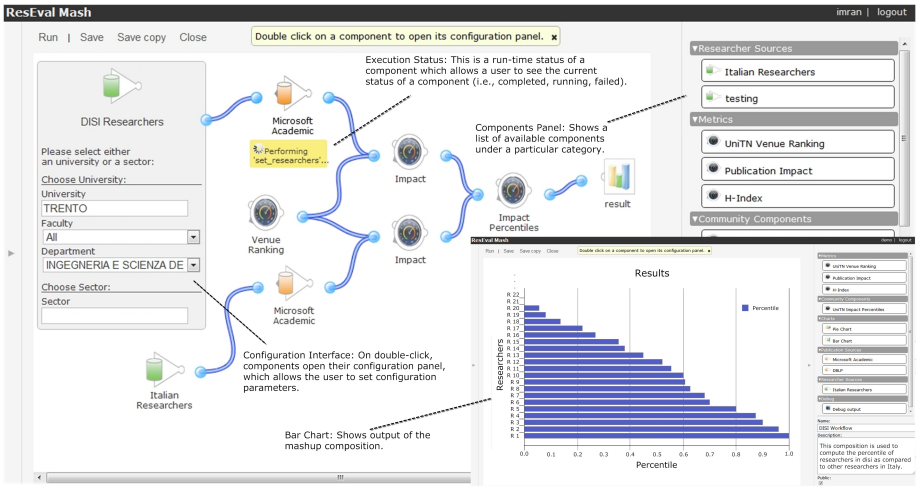


Fig. 2. Composition editor and example mashup output.

mappings among the components, which is a well-acknowledged problem known form several user studies of EUD tools [6], did not occur at all in the our study. This preliminary study suggests that ResEval Mash is a successful tool appealing to both expert programmers and end-users with no computing skills.

5 Related Work

The idea of focusing on a particular domain and exploiting its specificities to create more effective and simpler development environments is supported by a large number of research works [5, 1]. Mainly these areas are related to Domain Specific Modeling (DSM) and Domain Specific Language (DSL). In DSM, domain concepts, rules, and semantics are represented by one or more models, which are then translated into executable code. Managing these models can be a complex task that is typically suited only to programmers but that, however, increases his/her productivity. In the DSL context, although we can find solutions targeting end users (e.g., Excel macros) and medium skilled users (e.g., MatLab), most of the current DSLs target expert developers (e.g., Swashup [4]). Also here the introduction of the "domain" raises the abstraction level, but the typical textual nature of these languages makes them less intuitive and harder to manage and less suitable for end users compared to visual approaches. Benefits and limits of the DSM and DSL approaches are summarized in [1] and [5].

Web mashups [8] have emerged as an approach to provide easier ways to connect together services and data sources available on the Web [2], together with the claim to target non-programmers. Yahoo! Pipes (`http://pipes.yahoo.com`), for instance, provides an intuitive visual editor that allows the design of data processing logics. Support for UI integration is missing, and support for

service integration is still poor while it provides only generic programming features (e.g., feed manipulation, looping) and typically require basic programming knowledge. The CRUISe project [7] specifically focuses on composability and context-aware presentation of UIs, but does not support the seamless integration of UI components with web services. The ServFace project (`http://www.servface.eu`), instead, aims to support normal web users in composing semantically annotated web services. The result is a simple, user-driven web service orchestration tool, but UI integration and process logic definitions are rather limited and again basic programming knowledge is still required.

6 Status and Lessons Learned

The work described in this paper resulted from actual needs within our university and within the context of an EU project, which were not yet met by current technology. It also resulted from the observation that in general composition technologies failed to a large extent to strike the right balance between ease of use and expressive power. They define seemingly useful abstractions and tools, but in the end developers still prefer to use (textual) programming languages, and, at the same time, domain experts are not able to understand and use them. What we have pursued in our work is, in essence, to constrain the language to the domain (but not in general in terms of expressive power) and to provide a domain-specific notation so that it becomes easier to use and in particular does not require users to deal with one of the most complex aspect of process modeling (at least for end-users), that of data mappings.

References

1. R. France and B. Rumpe. Domain specific modeling. *Software and Systems Modeling*, 4:1–3, 2005.
2. B. Hartmann, S. Doorley, and S. Klemmer. Hacking, Mashing, Gluing: A Study of Opportunistic Design and Development. *Pervasive Computing*, 7(3):46–54, 2006.
3. M. Imran, F. Kling, S. Soi, F. Daniel, F. Casati, and M. Marchese. ResEval Mash: A Mashup Tool for Advanced Research Evaluation. In *Proceedings of WWW 2012*, pages 361–364.
4. E. M. Maximilien, H. Wilkinson, N. Desai, and S. Tai. A domain-specific language for web apis and services mashups. In *ICSOC*, pages 13–26, 2007.
5. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.
6. A. Namoun, T. Nestler, and A. De Angeli. Service Composition for Non Programmers: Prospects, Problems, and Design Recommendations. In *Proceedings of ECOWS*, pages 123–130. IEEE, 2010.
7. S. Pietschmann, M. Voigt, A. Rümpel, and K. Meißner. Cruise: Composition of rich user interface services. In *Proceedings of ICWE'09*, pages 473–476. 2009.
8. J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding Mashup Development. *IEEE Internet Computing*, 12:44–52, 2008.