From a Simple Flow to Social Applications

Juan Jara, Florian Daniel, Fabio Casati, and Maurizio Marchese

University of Trento, Via Sommarive 5, 38123 Povo (TN), Italy
{juan.jara, daniel, casati, marchese}@disi.unitn.it

Abstract. Currently, there are a lot of people trying to leverage on the success of social networks by implementing social applications. However, implementing social applications is complex, due to the requirements and constraints put by the social networks to protect their data. In this work we present Simple Flow, a tool that simplifies the creation of social applications. Simple Flow proposes a processes-based approach to the design and execution of social applications. Simple Flow targets end-users and programmers with no experience in programming for social networks, giving them the possibility to design processes by concatenating social network actions (like post a message or comment a photo). For the execution of the designed processes Simple Flow interconnects, at runtime, template web pages (one page per action) according to the process design defined previously. These templates abstract the complexities of the interactions with social networks.

Keywords: Social Applications, Design Tools and Techniques, Component-based development

1 Introduction and Motivation

Social applications are applications that use a social network infrastructure to reach more users and disseminate themselves. These social applications have different goals like meeting new people, sharing experiences, making professional contacts, getting recommendations, advertising products among other activities.

There are a lot of people trying to leverage on the success of social networks by implementing social applications. However, programming social applications has some particular aspects that are different from generic web applications, for example:

- The access to social network resources is usually done through restful APIs (application programming interface), and already learning how to get the desired data from a resource becomes a time consuming task;
- The application authentication, which is the process in which social applications have to authenticate themselves with the social network with which they want to interact; and
- The user permission system. In social networks, usually, only a small portion of the data is public. In order to get the rest of the data, the social applications need to get specific user-permissions, which depend on the data that the applications want to access.

These new aspects increase the complexity required to program social applications and usually discourage the people that approach it.

With our work we aim at reducing the complexity of programming social applications. We propose Simple Flow, a tool that facilitates the way social applications are designed and executed. Simple Flow has a design phase, where users use a process-based approach to design social applications, and an execution phase, in which users can execute the processes from the design phase.

In the design phase, we provide a method that makes it easier to interact with the desired resources in a social network. We abstract the social network API model into **action patterns** (like select user, create a post or upload a photo) that users can combine into processes that model a social application and express its logic.

In the execution phase, we provide a set of **template web pages** (one template per action pattern) that can be linked among each other based on the designed process from the design phase. Each template comes with a set of user interface elements related to the action pattern it represents and also internally handles the authentication process and the user permissions, abstracting from the user the complexity of these procedures.

With Simple Flow we do not only target programmers with no experience in social applications, we want to enable the widest possible set of people to create social applications. For this reason, an additional goal of our work is to grant the ability to create social applications to a set of people that goes beyond professional programmers. It will be part of our future work therefore to identify how far we can go in terms of simplifying this "programming" and in terms of lowering the skill levels required to design social applications.

The remainder of this paper is organized as follows. In the next section we present the context analysis. In Section 3, we explain our social application model. In Section 4, we describe our proposal. In Section 5, we introduce some related work. In Section 6 we present the discussion and the future work.

2 Context Analysis

In this section we explain who needs to create social applications and what types of social applications are the most needed or requested.

2.1 Who needs to implement social applications?

To learn about the needs of social applications, we searched and analyzed the social applications requests in three application-development outsourcing websites: Freelancer (`http://www.freelancer.com/`), oDesk (`https://www.odesk.com/`) and vWorker (`http://www.vworker.com/`). Most of the requests are related to the creation and management of contests which aim at motivating user participation and brand advertisement. We also found out that there is a constant demand by marketing people for analytic tools that measure the impact of actions in social networks. These tools should be able to compute a wide range of

metrics, which are calculated using metrics like the number of posts, comments and likes (or a combination of thereof).

From our own work as researchers we know, for example, that there is a constant need of recruiting people for participating in experiments and user studies. It would be useful to have an application that automatically disseminates information about these experiments and helps recruiting people. There is also the need to create applications for crowdsourcing content creation and content description, all related to a specific topic.

2.2 Classification of most required applications

We can classify the identified needs into the following categories of social applications:

- **Event advertisement:** These applications allow users to define an invitation message and schedule it for periodical dissemination through social networks.
- **Guided content creation:** These applications guide users through the content creation process, e.g., uploading a photo to a specific album to participate in a photo contest, adding information related to the context of a photo, etc.
- **Voting:** These applications present the user with a list of objects and ask him/her to select one or more of them, e.g., asking a user to select from a list of videos the one that he/she thinks is the funniest.
- **Contest drawing generator:** These applications select one or more objects from a list following a specific criterion, e.g., from a list of message posts select the one with most comments.
- **Targeted dissemination:** These applications send, to a selected number of friends, a user defined request, e.g., selecting three friends and asking them to comment a photo.
- **Simple analytics:** These applications get periodically information about specified objects (e.g. albums, photos, posts). This information can be used later for impact analysis, e.g., getting daily the number of likes or number of comments for a photo.

Some of these simple applications can be combined to create more complex ones, e.g., the guided content creation and targeted dissemination can be combined to create an application for crowdsourcing content creation or content annotation among a user's friends.

3 The Social Application Model

The first step for facilitating the programming of social applications is to transform the restful API model to a more familiar model. For this we abstract the typical social networking APIs to action patterns (like select user, create a post or upload a photo). Then we provide a method to combine the action patterns into processes that represent the models of social applications.

3.1 The Action Conceptual Model

We select from social networks the objects that we consider most relevant, e.g., users, posts, photos, comments, etc. Then, for each object, we add their available actions, e.g., upload, comment, like. The result is a set of patterns that represent the actions available to social network users, e.g., select a friend, comment a photo, create a post, etc. From now on, we will refer to action patterns just as actions. The actions can require an input, may produce an output and have some options that slightly modify their standard behavior. All of the previous elements (actions, inputs, outputs, options, etc.) define our action conceptual model. Figure 1 shows the model and the how its elements are related.

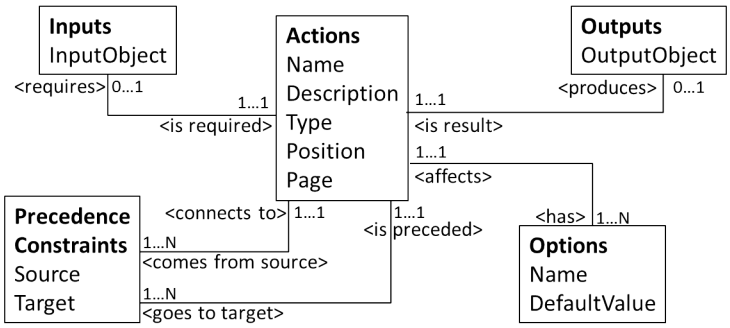

Fig. 1. Simple Flow action conceptual model

We describe the entities in Figure 1 as following:

- **Actions:** are the action patterns that represent a single step in our process model (which is explained in the next section), e.g., *upload photo to wall*, *like a post*, *comment an album*. We defined this list by combining the available actions exposed by social networks through their APIs and, by analyzing the interactions in the applications described in the previous section.
- **Precedence Constraints:** represent the relation between actions. The precedence defines what actions can be used in each step of a process during the design of the process.
- **Options:** represent action attributes that are used to modify the standard behavior of its related action. Taking as an example the action *comment an album*, one of the options for this action defines who creates the comment; if it will be the process designer, the user executing the process, or both.
- **Outputs:** represent the data produced by an action after its execution. For example, the *comment an album* action produces a comment object, and the *upload photo to wall* action has a photo object as result.
- **Inputs:** represent the data requirements of an action. For example, the *comment a photo* action requires a photo object that is the target of the

comment. The input object for an action can be chosen from one of the produced outputs by other actions or from objects that already exist in social networks.

The action conceptual model helps us to formally represent and design processes for social networks.

3.2 The Process Model

We combine the actions and the precedence constraints from the conceptual model to produce a directed graph that we call the *action graph*. The actions are connected according to the data navigation structures of social networks (with a major influence from Facebook). The *action graph* is the core of our proposal and is used to guide the user during the design of processes. The *action graph* is our approach to relieve end-users from most of the complexities of designing processes for social networks.

A process is designed by concatenating the nodes from the *action graph*, following the possible paths from one node to another. The first step of a process has to be always the start node of the *action graph* and the last step the end node. Only connected nodes can be executed consecutively in a process.

Figure 2 shows the conceptual representation of the *action graph* during the design of a process. The highlighted node (with a big arrow on top) represents the current selected action. The nodes pointed by the current node represent the actions that can be executed after the selected action. The start node has the "Login" label and the end node has the "End" label.

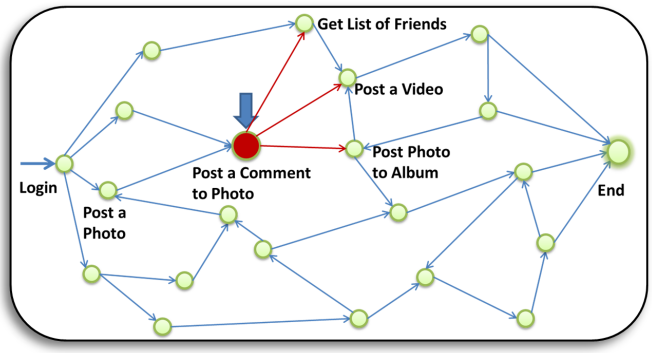

Fig. 2. Conceptual representation of the *action graph*

In the next section we explain how end-users can design and execute processes using the concepts presented in this section.

4 Design and Execution of Social Applications

To test our process model, we propose Simple Flow, a tool for the design and the execution of social applications. Simple Flow uses a process approach to define social applications, that is, social applications are represented as simple processes that run over social networks. Simple Flow has two phases:

- **The design phase:** where users create and define their processes, their logic and what data from social networks it uses and affects.
- **The execution phase:** where users select and run a process; here, Simple Flow runs a process by concatenating predefined template web pages following the process design that was created in the design phase.

We implemented a first version of the design phase and we are working on the implementation of the execution phase. In the following sections we explain in details both phases, for the design phase we show an example of the working UI (user interface) and for the execution phase we sketch an example of how it will work.

4.1 The Design Phase

The design phase is where users define and create the processes that represent the social applications they want to create. The core of the design phase is the action graph, which is our approach to lift from end-users the complexity related to learning how to interact with the social network resources through the social networks APIs.

Design Phase Components. The components of the design phase are classified as data, logic or presentation components as shown in Figure 3. The arrows indicate the dependency relation between the components, e.g., the process designer depends on the action graph.

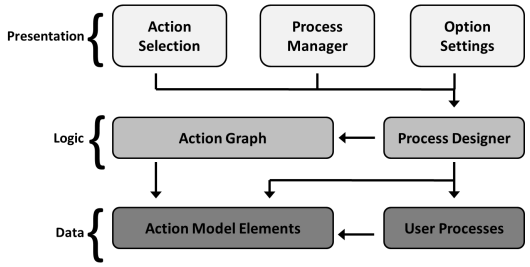

Fig. 3. Components of the design phase

The data layer stores the elements of the action conceptual model and also the user defined processes. The components of this layer are explained as following:

- The *Action Model Elements* contain the instances of the entities of the action conceptual model that was explained in Section 3.1.
- The *User Processes* represent the set of processes already defined by the users by concatenating sequentially actions and that are ready to be executed.

The components of the logic layer are responsible for associating to each action its corresponding elements (inputs, outputs and options) and to also logically relate the actions using the action graph. The components of this layer are explained as following:

- The *Action Graph* represents the navigation structure between actions, it is explained in Section 3.2.
- The *Process Designer* is the component that connects all the elements of an action. When an action is added to a process, it associates to the added action all its corresponding elements like input, output and options. Furthermore, using the action graph, it prepares the actions that could be executed in the next step.

The components of the presentation layer manage how the information is presented to the users. The components of this layer are explained as following:

- The *Process Manager* shows all the steps of the designed process, the input and the output of each action.
- The *Action Selection* shows the list of actions that can be executed after the last selected action.
- The *Option Settings* presents what options the last selected action has available.

Designing a Social Application. Figure 4 shows a view of the UI during the design of a process. Each component of the presentation layer assists the user in the design of a process.

The *Process Manager* shows the process that is being designed and is represented by the table in the "Flow Actions" column. The first column of the table indicates the step of the process associated to the action and the last column indicates the output that will be produced after the execution of the action, this data can later be used as an input for another action. If an action requires an input, its description will contain the following string: "[???]". The step that is currently being designed in Figure 4 is the third one: Comment a Photo.

The *Action Selection* presents to the user a list of actions that can be executed after the current action (the list is contextual to the last action added to the flow). In Figure 4 is represented by the "Available Actions" column. If the user selects an action from the list, that action becomes the current action, which would be the fourth step of the designed process.

The *Option Settings* presents to the user a set of options associated to the current action. In Figure 4 is represented by the "Action Options" column. When the user selects an action, the options for that action are loaded with default

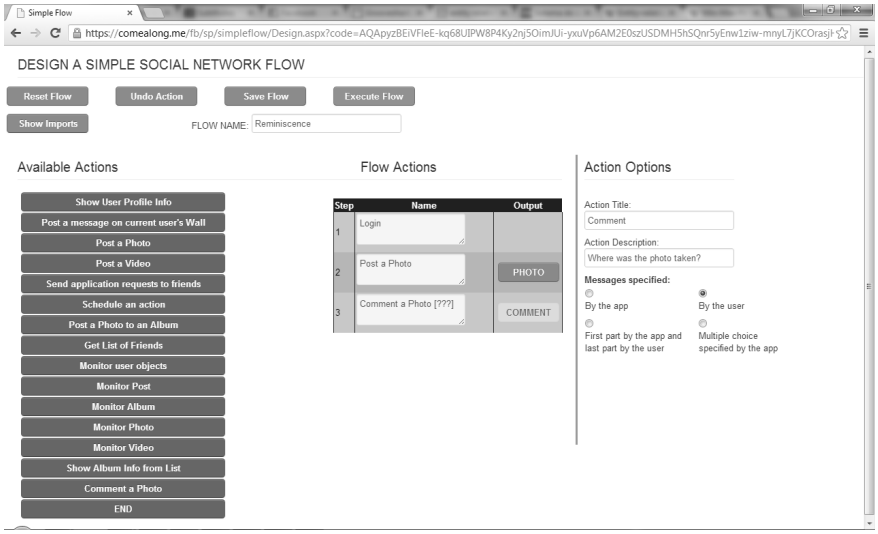

Fig. 4. UI of the design phase during the design of a process

values that express to some degree the intended behavior of the action. The user can change these options to modify the action behavior to meet specific needs.

Figure 5 shows a finished process model. This process represents a social application for crowdsourcing photos that have annotated context information in the form of comments. The goal of this application is to collect photos from several users and to attach to each photo specific data related to the context of the photo. A sketch example of the application during its execution can be seen in Figure 6.

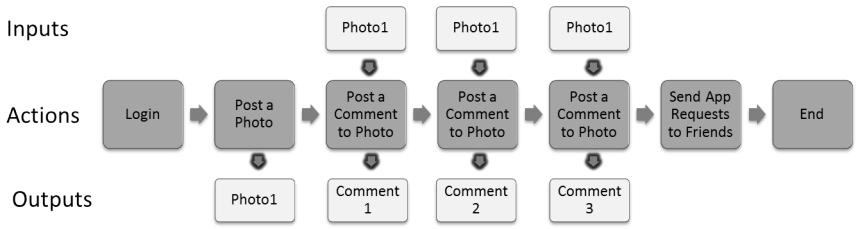

Fig. 5. A process model that represents a social application

The designed processes can be deployed for execution by their owners as a part of Simple Flow. Each process can be disseminated by sharing its link and can be executed by one or more people.

4.2 The Execution Phase

The execution phase is where Simple Flow runs the processes designed in the previous phase. The main element of the execution phase is the action page.

The action page is a template web page that represents an action from the action conceptual model. Each page is prepared to manage all the elements that are associated to the action it refers to (input, output and options). The action page facilitates the creation of social applications by:

- **Implementing a UI for interacting with a social network API:** an action page is related to a specific action, which again is based on one of the social network APIs; therefore, the action page implements a UI for a social network API.
- **Automatically authenticating the application with the social network:** each action page manages internally the authentication process of the Simple Flow application with the social networks and uses these credentials to interact with social networks when users run processes.
- **Managing user permissions:** Simple Flow includes all the required user permissions by the set of actions that it provides; therefore, when users register the Simple Flow application, they will be asked to give the necessary permissions to run any designed process.

The execution phase runs a process by concatenating action pages following the process model. The execution phase starts when a user clicks the link associated to a process. At the beginning of the phase, Simple Flow reads the process information and redirects the user to the first action page and makes this page point to the next action page defined in the process model.

During the rest of the execution phase, for each step of the process, Simple Flow:

- gets the input required by the action,
- prepares the page according to the action options, and
- makes the page point to the next action page.

When the user finishes carrying out the requested action, he/she is redirected to the next page where the cycle starts again. Figure 6 shows a mockup of the action pages for the process model from Figure 5 during the execution phase.

Simple Flow also facilitates the creation of social applications by allowing its users to skip the steps related to registering the social application in the social network and then installing it in a web server (this will be helpful especially for users that only want to implement a few processes).

5 Related Work

In this section we present the works related to end-user development, we group them according to their proposed approach to the creation of applications.

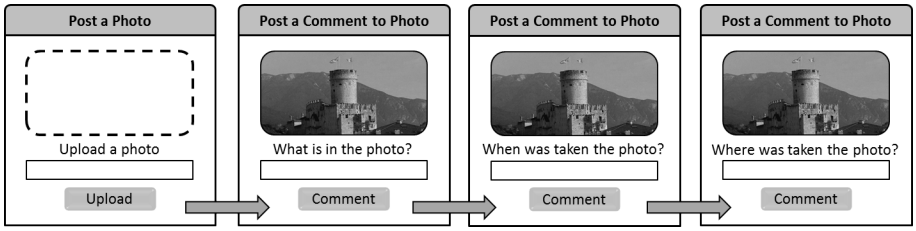

Fig. 6. Execution of the process from Figure 5, steps 2 to 5

5.1 Using Processes

First, we present the IFTTT (If this then that) project [1], which allows users to define simple ECA (Event - Condition - Action) rules over social networks. Users create rules by combining triggers (e.g., an upload of a photo, a post of a message) and actions (e.g., tweeting a message, commenting a photo) with their associated channels (e.g., Twitter, Facebook). The simplicity of IFTTT is what inspired us to build our tool and we think that this is how applications should be designed, specially by end users. IFTTT allows the design of 1-step processes that are executed when their associated trigger is fired. The processes designed with our tool can have more than one step and, generally, are executed directly by users. There is also, Atooma [2], which is an IFTTT like approach for smartphones. The difference with IFTTT is that users can add several triggers for the If this condition and also do several actions for the then that part.

Then we have the BPM4People project [3], which extends BPMN (business process modeling notation) with new task types that represent the unstructured social interactions between the process stakeholders (e.g., social posting, commenting, voting, invitation to activity). Both, the BPM4People project and our tool, propose the design of applications using a process representation. The difference is that the authors in [3] define processes using BPMN, which requires time to learn and make this approach not suitable to end-users.

Then we also have [4], which allows users to define crowdsourcing programs for the crowd computer using a process-based approach. The crowd computer considers humans as part of the hardware of the system and available for the realization of computational tasks. The difference with Simple Flow is that the crowd computer is focused exclusively in the creation of crowdsourcing tasks and the steps of a task can involve or not the interactions with social networks, while Simple flow can define processes that aim or not to define a crowdsourced task but all the steps of a task involve an interaction with a social network.

5.2 Using Mashups

In [5], the authors point to the fact that most of the proposals for general purpose web-service composition (mashups) aimed at end-users failed. On the basis of these failures, the authors propose to use a domain specific approach to

mashups. The proposed approach indeed improved the results of the general purpose mashups by lowering the entry barrier for end-users to mashup composition. We can consider Simple Flow as a domain specific mashup, the difference between [5] and Simple Flow is that we also guide end-users during the web-service composition with the action graph, preventing end-users from selecting actions (web-services) that cannot be combined.

Then we have [6], where the authors propose to support the development of mashups by integrating in existing mashup tools two techniques: automatic composition, which automatically creates mashups according to the goals of the end-user and; interactive pattern recommendation, which uses mashup composition patterns to recommend components to the end-user based on the current mashup design. The difference with Simple Flow is that the proposal of [6] depends on the existence and correctness of user-created patterns while in our proposal the recommendation is based on the action graph. However, the proposal of [6] could be integrated in general purpose mashup tools while Simple Flow is constrained to a specific domain.

5.3 Other approaches

WeFlow [7], a tool for creating simple collaborative applications. The authors propose the use of a specification language (similar to natural language) for defining web applications. WeFlow has a generator engine that takes as input a script written in this language and generates a web application. Although WeFlow has similar goals than Simple Flow, the applications produced by WeFlow do not interact with social networks.

The Jabberwocky programming environment [8], which is composed of three components: a resource management system for human and computer workers, a framework for programming sequential and parallel tasks, and a high-level programming language for abstracting the low-level concepts of their framework. The main difference with our tool is that they define applications using a programming language similar to a structured query language and we use a process model approach.

In [9], the authors propose to use a spreadsheet environment for the construction of mashups. The authors define custom functions that call web services and pass the cell value as input of the service. Then, end-users can compose mashups by linking functions using the cell reference property of spreadsheets. The authors in [9] offer a developing environment that is well-known to end-users, however, they have the same drawbacks as the general purpose mashups.

6 Discussion and Future Work

When designing Simple Flow, we decided to make it as simple as possible to lower the learning curve required to use it. To achieve this, we minimized the number of inputs, outputs and options that each action had, which in turn increased the total number of actions. To explain this better, we take as an example the action

of uploading a photo. With this action we have the option to upload a photo to the user stream or to a user album. In the implementation of Simple Flow we could have made this an action with two input parameters, one for specifying the destination of the photo (stream or album) and the other for specifying the identifier of the album in case that the selected destination was an album. Instead, we decided to minimize the number of input parameters, which resulted into two actions to upload a photo: one that uploads a photo to the user's stream (without parameters) and one that uploads a photo to a user album (with one parameter for the album identifier). We think that the overhead of having many actions instead of one can be overcome by giving meaningful descriptions to each action, while still maximizing the simplicity of the interface.

For the users that want more independence from Simple Flow we plan to add the option to export the implemented applications as an installable package. Then, to run the exported applications, they require a new web server to install it and to register the application to the desired social network.

Finally, we plan to carry out user studies to evaluate the design time and the correctness of the applications created using Simple Flow, and the usability of the UI of the page templates during the execution of the designed applications.

References

1. IFTTT: Put the internet to work for you, `https://ifttt.com/`
2. Atooma, `http://www.atooma.com/welcome`
3. Brambilla, M., Fraternali, P., Vaca, C.: BPMN and design patterns for engineering social BPM solutions. In: Business Process Management Workshops, Springer (2012) 219–230
4. Kucherbaev, P., Tranquillini, S., Daniel, F., Casati, F., Marchese, M., Brambilla, M., Fraternali, P.: Business Processes for the Crowd Computer. In La Rosa, M., Soffer, P., eds.: Business Process Management Workshops BPM 2012 International Workshops Tallinn Estonia September 3 2012 Revised Papers. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2013) 256–267
5. Casati, F., Daniel, F., Angeli, A.D., Imran, M., Soi, S., Wilkinson, C.R., Marchese, M.: Developing Mashup Tools for End-Users: On the Importance of the Application Domain. International Journal of Next-Generation Computing **3**(2) (2012)
6. Chowdhury, S.R., Chudnovskyy, O., Niederhausen, M., Pietschmann, S., Sharples, P., Daniel, F., Gaedke, M.: Complementary Assistance Mechanisms for End User Mashup Composition. In: WWW13. (2013) 269–272
7. Kokciyan, N., Uskudarli, S., Dinesh, T.B.: User Generated Human Computation Applications. In: Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom), IEEE (2012) 593–598
8. Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. Proceedings of the 24th annual ACM symposium on User interface software and technology (2011) 53–68
9. Hoang, D.D., Paik, H.y., Benatallah, B.: An analysis of spreadsheet-based services mashup. In: Proceedings of the Twenty-First Australasian Conference on Database Technologies - Volume 104. ADC '10, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2010) 141–150