Rapid development of interactive applications based on online social networks

Ángel Mora Segura, Juan de Lara, and Jesús Sánchez Cuadrado

Modelling and Software Engineering Group
http://www.miso.es
Department of Computer Science
Universidad Autónoma de Madrid (Spain)
{Angel.MoraS, Juan.deLara, Jesus.Sanchez.Cuadrado}@uam.es

Abstract. Online social networks, like Twitter or Google+, are widely used for all kind of purposes, and the proliferation of smartphones enables their use anywhere, anytime. The instant messaging capabilities of these services are used in an ad-hoc way for social activities, like organizing meetings or gathering preferences among a group of friends, or as a means to contact community managers of companies or services.
Provided with automation mechanisms, posts (messages in social networks) can be used as a dialogue mechanism between users and computer applications. In this paper we propose the concept of post-based application, an application that uses short messages as a medium to obtain input commands from users and produce outputs, describing several scenarios where these applications are of interest. In addition, we provide an automated, *Model-Driven Engineering* approach (currently targeting Twitter) for their rapid construction, including dedicated Domain-Specific Languages to express the interesting parts to be detected in posts; and query matched posts, aggregate information or synthesize posts.

Keywords: Social Networks, Post-based Application, Model-Driven Engineering, Domain-Specific Languages, Social Applications.

1 Introduction

Online social networks (OSNs) based on microblogging are booming nowadays, thanks in part to the proliferation of smartphones and mobile devices. Hence, services like Twitter or Google+[1] are extremely used nowadays to connect with friends, or to organize social activities. These services are not only used for leisure, but most companies and brands recognise the reach and importance of OSNs today, and use these services to keep in contact with clients [18].

In this setting, we observe a growing need to automate social activities, leveraging on popular OSN platforms, like Twitter. On the one hand, users of OSNs – possibly lacking any programming skills – may wish to define simple applications involving the participation of a community of users. On the other hand,

[1] http://www.twitter.com, https://plus.google.com

companies may like to open their information systems to OSN platforms, but this integration effort needs to be done by hand. Related to this last issue, companies frequently interact with potential customers via the *online community manager*, in charge of managing and moderating the relationships with their clients through posts (messages in OSNs). Even though this role is of crucial importance nowadays, many of his tasks are performed manually, in ad-hoc ways.

Our thesis is that OSNs based on short, instant messages, are suitable as front-ends for computer applications. We call them *post-based* applications, and present many advantages in some scenarios. First, OSNs are designed to support a high load of users and posts, serving as a robust front-end, which could be difficult to achieve for companies or end users. Second, many people are familiar with specific OSNs and have it already installed in their devices. Hence, they do not need to learn a new application, or install a new one. Finally, applications can leverage from the social network structure provided by the OSN platform.

Several scenarios benefit from post-based applications. In the first one, small, simple, self-contained applications can be designed by unexperienced end users. For example, for mobile learning games, or to organize votings. The second one involves the rapid construction of applications to coordinate a large amount of people upon unexpected events, like natural disasters or strikes in airports. Finally, specific OSNs can be used as a front-end for existing information systems. For example, an airport may send notifications with flight information, or with status updates via *tweets* (posts in the Twitter platform) to interested users.

These scenarios present several challenges. First, relevant information needs to be extracted from posts. Posts are unstructured, and users cannot be expect to follow a tight syntax. Hence, we need a simple way to express and detect the interesting information. Second, a mechanism is needed to specify simple actions, like querying the extracted information, or synthesizing posts with collected information. Finally, a quick, easy way for constructing this kind of applications is needed, enabling their use by non-experts, but supporting also their deployment into servers, and their integration with existing information systems.

Contributions. We introduce the concept of *post-based user interface* and *post-based application*, proposing a Model-Driven Engineering (MDE) approach for their automated rapid construction. The solution includes: (a) A Domain Specific Language (DSL) for expressing patterns that is connected with WordNet [16], a lexical database for the English language; (b) a DSL for describing actions; and (c) an Eclipse-based prototype environment to model and deploy post-based applications, which currently targets Twitter. This paper extends [21], a short tool-demo paper which focussed on the tooling aspect of the solution.

Paper organization. Section 2 describes several motivating scenarios for *Post-based applications*, showing their benefits. Section 3 proposes our architecture. Section 4 describes a DSL for expressing patterns on posts. Section 5 presents a DSL for describing simple actions. Section 6 describes tool support. Section 7 analyses related research, and Section 8 ends with conclusions and future work.

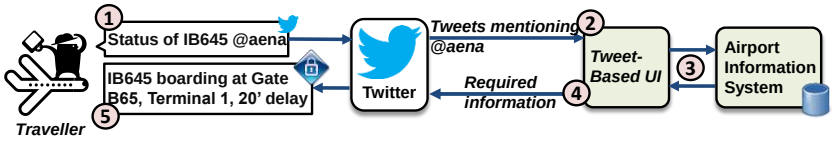

Fig. 1: A tweet-based user interface for an airport information system.

2 Motivating examples

In this section, we provide some motivating scenarios for post-based applications. The underlying concepts are applicable to different OSNs, but for several reasons (popularity, availability of an API, the availability of a social network structure, possibility of both public and private messages), we currently consider Twitter.

Twitter is an OSN based on microblogging, which permits a direct communication among users via posts, short messages limited to 140 characters called *tweets*. Tweets are public by default, and searchable by any person not necessarily a Twitter user. There are also private messages, directed to a specific user. Twitter users may follow a number of other users. A follower receives any tweet of the users they follow. User names start by '@'. Tweets may contain user names, and the mentioned users get notified whenever this happens. Tweets may also contain hashtags (a word preceded by '#'), names agreed by community convention, which facilitate the searching of tweets. Finally, tweets may contain links (in particular to pictures), and be geolocalized (i.e., contain information on the position on earth where the tweet is being sent).

Next, we describe some scenarios where post-based applications are of interest.

1. **Tweet-based user interfaces for Enterprise Systems** In this scenario a company decides to use Twitter as a means to provide access to its information system. The advantage is that users do not need to install a new software for interacting with the information system. They use their Twitter accounts for the interaction, so that they can access the information systems either from smartphones, laptops or desktop computers in a unified way. Moreover, additional information can be obtained from users if tweets are geolocalized. The social network structure of Twitter can also be exploited, e.g., to broadcast messages, received by all followers.
   As an example, Figure 1 shows the integration of a tweet-based user interface with an airport information system. The purpose of such system is to inform travellers of the status of their flights, among other services. In a first step (label 1), a person about to travel sends a tweet requesting information about a certain flight. Such tweet mentions the account of the airport or the management entity, like *@aena* for the case of Spanish airports. In step 2, the tweet-based interface receives a mentioning tweet, and matches such tweet accross a collection of relevant requests. In this case, the tweet request information for a flight, which is directed to the information system (step 3).

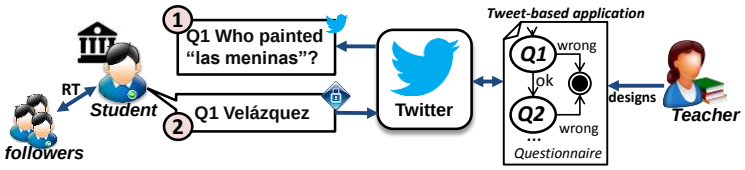

Fig. 2: Tweet-based outdoor quizz.

The flight number is extracted and sought in the airport's database. Then, a private message for the particular user is automatically synthesized (label 4) by the interface, which is received by the user via Twitter (label 5). In this case, the initial message is public, but for privacy issues, a private message could have been used instead.

2. ***Ad-hoc* tweet-based collaborative applications**. In some cases, groups of end-users are interested in using Twitter as a means to collect structured information that could be later automatically processed. However, they may lack the technical abilities to develop such an application from scratch. These situations include all sorts of votings, organization of sport events, educational outdoor games, and many others.
   As an example, Figure 2 shows the working scheme of an outdoor educational game, where the teacher designs a questionnaire, to be answered by groups of students during a visit to a museum. Hence, questions are answered by exploring the museum. Students organize in groups and can use the social network structure of Twitter to collaborate among them (e.g., via retweets and tweets) so that they can search for the answer in parallel. The system provides new questions when a question is answered, and can finally provide some statistics. In this scenario, the teacher may design the questionnaire, not needing to be proficient in programming languages or the Twitter API. Along the paper, we will use a simpler example in this scenario, consisting in a simple yes/no voting system among a community of users.
3. **Massively collaborative application for unexpected situations**. Information technologies play a crucial role for emergency response (so called peace technologies) [9]. In particular in [15] it is reported that Twitter was an important medium for the Japanese government to distribute information to millions of people during the Fukushima nuclear radiation disaster. In this type of applications, the geoposition information offered by geolocalized tweets can be valuable. As an additional example, we may consider a Twitter front-end to help in reassigning flights in case of a strike. This is useful, as the airplane company web site may not support the high peak load if thousands of users try to access the system at a time. Hence, each passenger may provide its flight, which is saved into a database. Once the flight is reassigned (manually by some operator), the user may receive a notification, or he may be asked to contact a service desk. Hence, in this scenario, we profit from the high load supported by Twitter, and its popularity, so that users do not need to install a new application.

3 Architecture

The working scheme of our solution for tweet-based applications is shown in Figure 3, where the numbers illustrate a typical interaction.

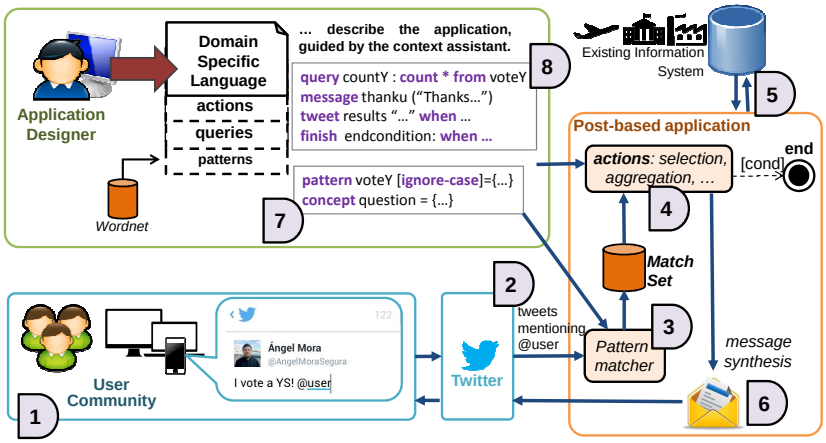

Fig. 3: Architecture provided by our solution.

In the first place (label 1), users send tweets or private messages via Twitter. Then, the relevant information in tweets needs to be extracted from the application. Our solution relies on the definition of patterns, expected to be found in tweets. Not every tweet is sought, but only those mentioning the user associated to the application, or private messages directed to it (label 2). The patterns (label 3) are defined by the application designer using a dedicated DSL. A typical application may include different queries, selecting the relevant concepts in matching tweets, or calculating different aggregation values from them (label 4). In addition, data can be obtained or sent to existing information systems (label 5). The data extracted from queries, or provided by the information system can be used to synthesize tweets or private messages, directed to the users (label 6). Finally, conditions can be defined to signal the end of the execution.

In order to facilitate the construction of such system, we provide an MDE solution, based on two domain-specific languages (DSLs). The first DSL helps in the definition of relevant patterns, and concepts to be found in them (label 7) in the figure. The latter are sets of relevant words, or fragments, and sets of synonyms can be automatically extracted from WordNet [16].

The second DSL (label 8), is targeted to the description of the processing logic of post-based applications. Both languages are defined in a modular way, permitting extensions for the peculiarities of different OSNs, but we currently target Twitter. The DSL allows defining queries on posts matching some pattern, using an SQL-like syntax. Queries can be used to select relevant information from posts, or to calculate aggregated information from a set of posts. The DSL also

provides commands to synthesize private messages and tweets, and to signal the end of the application execution. Finally, it is also possible to define what we call *data hooks*, as a way to push extracted data into an existing information system, or to gather data from it. Being an MDE framework, we can profit from code generation for different technologies, like REST or SOAP web services.

The next two sections describe the two DSLs in detail.

4 A DSL for describing patterns

We have built a DSL, directed to facilitating the definition of patterns to be identified in tweets. A small excerpt of its meta-model is shown in Figure 4. It is made of a core set of elements (package PatternCore), which then can be specialized for different OSN platforms. In this case, we specialize it with concepts from Twitter.

A pattern is made of concepts (class Concept), and in its simplest form, a concept (called ExplicitConcept) is a set of words (attribute tokens). This set can be either defined explicitly by the designer, or can be automatically taken from a synonym set provided by WordNet. We have defined special concepts for numbers and letters (omitted in the meta-model), with length given by an interval, and permit defining sequences of these.

We have also included specific Twitter concepts (TwitterPatterns package), like patterns to detect user names, URLs (specially pictures), and to define collections of interesting hashtags. As we will discuss in the next section, the meta-data information present in tweets, like the originator, date or geoposition can be retrieved and does not need to be explicitly declared in patterns.

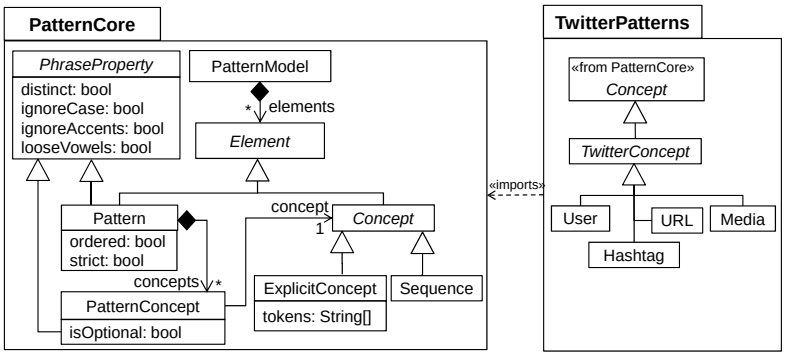

Fig. 4: Simplifed excerpt of the meta-model to describe patterns.

The class Concept actually inherits from ResultSet (not shown in the figure), which allows matched concepts to be sent as external data (see next section), and referenced in queries. As can be observed, we have separated the Concept definition from their usage within Patterns, which permits their reuse in different

contexts. An intermediate class PatternConcept enables the configuration of how the concept is to be used, specifying for example whether it is optional in the pattern. Patterns also indicate if concepts have to appear in some specific order, or allow the interleaving of concepts with other words. It is also possible to specify that some concept cannot occur in a pattern, and whether concepts are to be sought ignoring upper/lower case, ignoring accents, and permitting missing vowels, as this is a usual idiom in posts, due to their restrictive length in some cases (especially in tweets).

The DSL has been provided with a simple textual syntax. Figure 5 shows some example patterns and concepts for the voting application mentioned in Section 2. The goal of the application is to detect positive or negative votings to simple questions, and the patterns detecting it are defined in lines 1–2. They are tagged as ordered meaning that concepts should appear in order, but are not strict, so that other elements may occur in the tweet, possibly interleaved with the concepts of the pattern. Only the yes and the no concepts are mandatory, while the question concept is optative as indicated by the ? symbol. Moreover, for all concepts in the pattern we ignore whether they are in upper/lower case (flag ignore-case), and admit variations with missing vowels (flag loose-vowels). The patterns make use of the concepts of lines 4–6. The words making concept *question* were actually taken from WordNet synonyms, as we will see in Section 6.

```
1  pattern voteY [ordered loose−vowels ignore−case] = {question?, yes}
2  pattern voteN [ordered loose−vowels ignore−case] = {question?, no}
3
4  concept question = {question, inquiry, enquiry, query, interrogation}
5  concept yes = {yes, affirmative, 1, true, y}
6  concept no = {no, negative, 0, false, n}
```

Fig. 5: Patterns for the voting example.

Altogether, *voteY* matches tweets like *"My vote 4 the qstn is YES!"*. In this case, the actual value of concept question is qstn while the value of yes is YES.

5 A DSL for queries and actions

Our approach considers the description of actions by means of another DSL. Its meta-model is split in a core package which includes those elements typically applicable to OSNs, and a package specific to Twitter. An excerpt of both packages is shown in Figure 6 (many concrete classes have been omitted, leaving only the base abstract classes of the main hierarchies). The action DSL uses the patterns DSL to perform queries on the concepts found in posts matching a certain pattern, and in addition supports other actions, namelly:

– **Querying**. Queries can be issued using an SQL-like syntax. They may refer to a set of matches of a pattern, as if they formed an SQL table, and

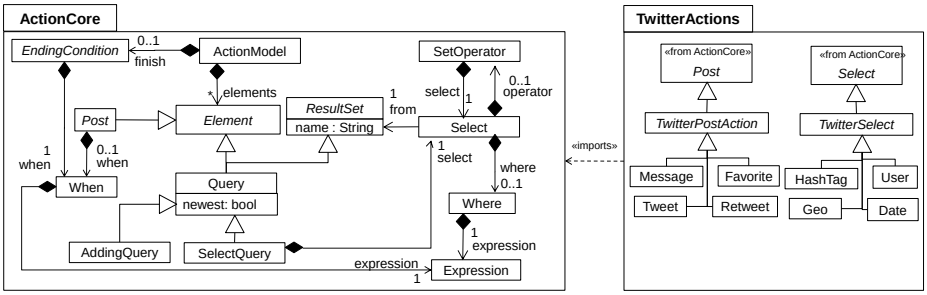

Fig. 6: An excerpt of the actions meta-model with the Twitter extension.

```
1  query countY : count yes from voteY;
2  query countN : count no from voteN;
3  @newest query users_voting : users from voteY union users from voteN;
4  message thanku ("Thanks for your vote.") to users_voting;
5
6  tweet presults ("Partial results: (%s) yes, (%s) no", countY, countN) when (countY + countN) = 15;
7  tweet results ("Results: (%s) yes, (%s) no", countY, countN) when (countY + countN) = 30;
8  finish end: when (countY + countN) = 30;
```

Fig. 7: Actions for the voting example.

the concepts in the pattern, as if they were SQL columns (as explained before, Concept inherits from ResultSet to this end). Three kinds of specialized queries can be issued. *Pattern queries* to select some concepts from a set of posts matching a pattern (represented as SelectQuery, which takes data from a result set obtained in the pattern matching phase), *aggregation queries* (class AddingQuery, to perform some arithmetical operation on result sets, like counting, and *metadata queries* (class TwitterSelect in the case of the Twitter package), to obtain a result set made of some tweet metadata, like its users, geopositions, images, or hashtags. Aggregation queries can calculate the maximum, minimum, average, count or add elements in a result set. Every query inherits from ResultSet and has a name, so that its results can be obtained from other actions. We currently support three kinds of operations (subclasses of SetOperator) with result sets from queries: union, intersection and substraction.

For example, Figure 7 shows two *adding* queries (lines 1–2) and one metadata query (line 3). The former counts the tweets matched by the voteY or voteN patterns. The later gathers the users that issued a tweet matching pattern, performed by a union operator.

While queries are similar to SQL queries, the data gathered from Twitter is dynamic. Hence, similar to data stream management systems [3] we may query using temporal windows [11]. Currently, we support two kinds of temporal windows, one considering all data, and another one with the last tweet. The two queries in lines 1 and 2 simply consider all data from the beginning of the execution. The users_voting, being labelled with @newest, discards an

incoming tweet as soon as considers it has been processed by a dependent action. Other temporal windows could be interesting as well, and we will consider them in future work.

- **Composing and Sending messages**. Once data becomes available from queries, messages can be composed and sent to a collection of users. This is reflected by class Post and its specialization TwitterPostAction. In the case of Twitter, messages can be public (class Tweet), or private, directed to a certain user (class Message). In addition, received tweets can also be retweeted, and be categorized as favorite. Public messages can be sent to a number of users (obtained through a query), in which case the messages contain a mention to those users. Messages may have a trigger (class When), so that the message is sent when the trigger becomes true.

  As an example, in Figure 7, line 4 sends a private message to the user that has voted. The user is obtained from query users_voting, which obtains the user of the last positive or negative vote. Lines 6 and 7 show the construction of two tweets. Similar to C's printf function, the tweet is composed by inserting data (countY, countN) into a string, in the places indicated using '%s' (independently of the variable type). Both tweets have trigger conditions, so that they are sent when the number of votes reach 15 and 30 respectively. In this case, they will not mention any user, but we could synthesize tweets mentioning users, in a similar way to private messages.
- **Execution end**. We also provide means to signal the end of the application execution (class EndingCondition and omitted subclasses). The condition may depend on several factors, like the number of tweets received that match a certain pattern, or time conditions. In Figure 7, line 8 shows a finish condition, when the number of votes reaches 30.

In this DSL, each action has a name, so that actions can refer to the data they produce simply by that name. The type of data does not need to be declared, but it is inferred by simple rules: data produced in a tweet match is considered String, and adding queries produce Integer data.

The execution model of the DSL is based on data flow, relying on data dependencies. This is the recommended execution model for reactive, event-driven, scalable applications [19]. In this way, an action is performed as soon as its data becomes available, unless it contains an explicit trigger, in which case it is executed when the data is available *and* the trigger becomes true. Figure 8 shows the data dependencies obtained from the example of Figure 7. The squares refer to the different actions (patterns, queries, messages, tweets and finish). Every action may produce some data, as indicated by solid arrows, and may depend on some data for their execution (shown by dashed arrows). The data produced by queries is assigned the same name as the query. Actions may have triggers, depicted as black circles (for simplicity we omit the actual trigger condition). For example, the thanku message can be sent when the users_voting data becomes available, which is produced by the users_voting query. Such query depends on the user metadata produced by tweets matching patterns voteY or voteN. Hence,

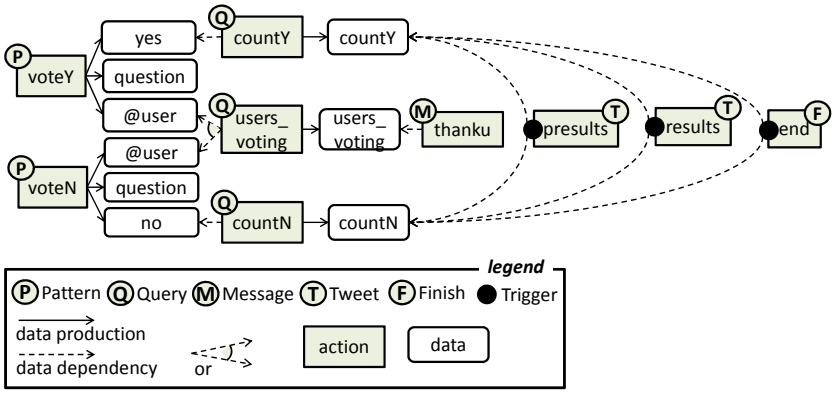

Fig. 8: Data flow execution model for the example of Figure 7.

users_voting needs to be reexecuted whenever a tweet matching voteY or voteN arrives. We forbid cycles of dependencies to avoid deadlock situations.

Our prototype does not make use of any data stream management system or stream processing library, but as future work we plan to improve its efficiency and scalability by compiling our DSL programs to a selected system.

Finally, in order to facilitate the connection with external information systems, the DSL permits declaring external data dependencies (expected data from an existing information system), and also data that is pushed into the information system. The former are asynchronous events, triggered by the external source, which provide some data to the model. Currently, we generate service interfaces from those two descriptions, one for the existing information system (to obtain the data from), and another for the tweet processing system (so that the information system can notify our system). The former service interface needs to be manually integrated with the existing information system.

6 Tool support

In this section, we present a prototype modelling environment, built as an Eclipse plugin. The environment permits describing patterns (Section 4) and actions (Section 5) using both DSLs, test the application in the environment, and then deploy it on a server. More information about the tool, including a video of the tool in action can be found at: `http://www.miso.es/tools/twiagle.html`.

The tool supports an agile development method, as during development, the user may test the patterns against tweets received in real-time, and send testing tweets using the tool itself. Figure 9 shows the tool being used to define some patterns corresponding to the running example. The question concept is being defined through the use of WordNet, and a contextual window offers the different synonym sets (i.e., the different *senses* for a given concept) for the given concept. Selecting one synonym set makes the tool copy all words into the concept.

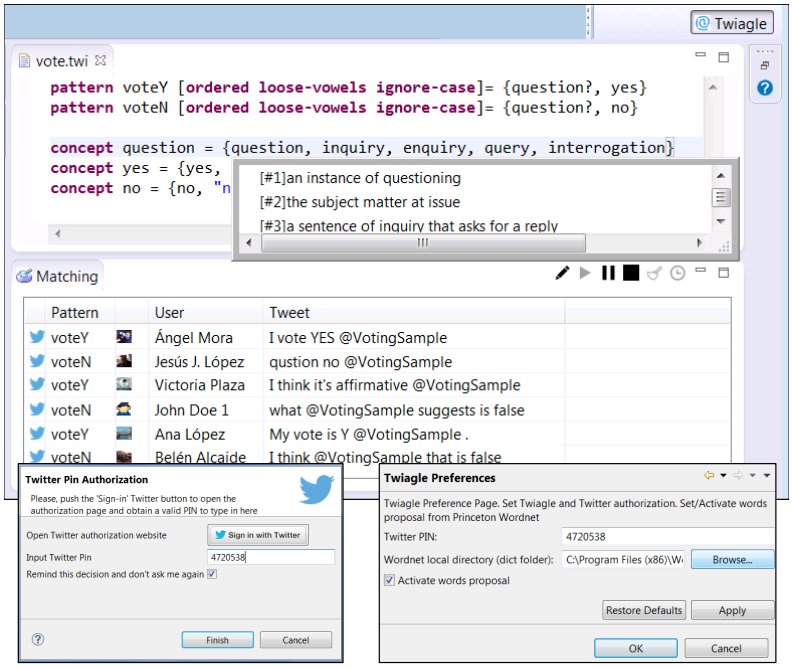

Fig. 9: Defining and testing patterns against live tweets.

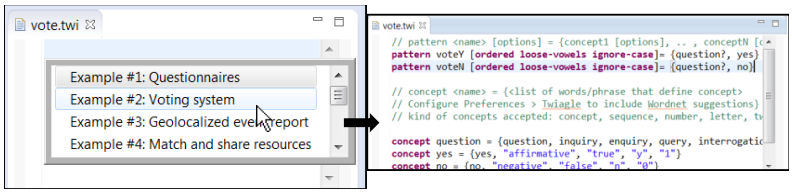

Fig. 10: Twiagle assistant suggesting different application kinds.

The tool is designed for its use by both software engineers and non-experts. For this purpose, it incorporates an assistant that suggests typical examples for a range of applications, as Figure 10 shows. Upon selection of one application kind, a skeleton example is presented, with explanations in the form of comments. The range of examples have been extracted from an analysis of current web applications, details of which are available from the tool web site.

The tool can be used for testing purposes, and the left-bottom of Figure 9 shows a connection window, where the user needs to authorize the tool to access the Twitter account associated to the application (@VotingSample in the figure). The tool enables storing such authorization pin in the tool preferences registry, so that it does not need to be entered for every execution. Once Twiagle is authorized, tweets can be obtained and matched against the defined patterns, as

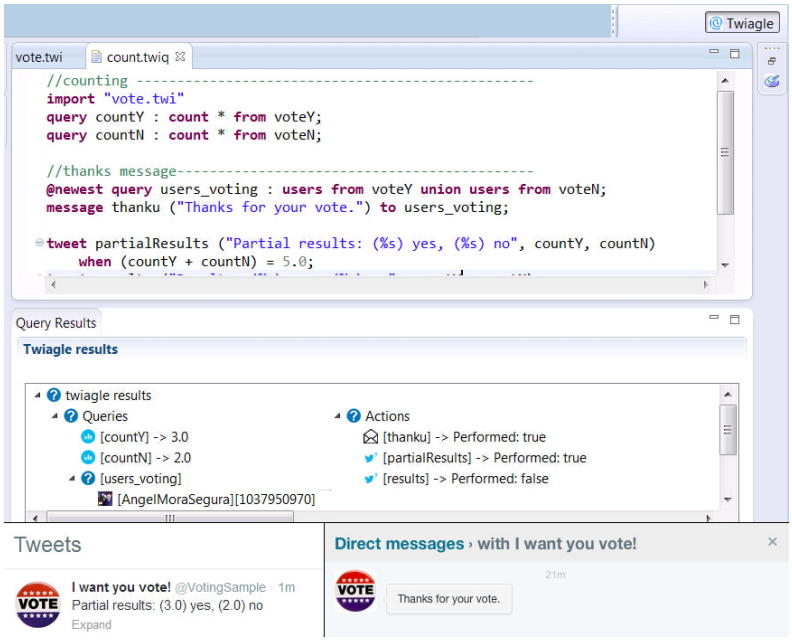

Fig. 11: Executing and tracing actions (up). Tweet-based application results shown in the Twitter console (bottom).

shown in the *"Matching"* view of the main window. The reception of live tweets can be paused, resumed, and testing tweets can be issued from the tool itself.

Figure 11 shows the tool being used to define the actions for the voting example, and test them over live tweets, so that the results of the different queries are visualized. In particular, the tool offers two trees, the one on the left showing the results of the different queries, while the one to the right indicates the actions being executed. The latter capability is very useful for debugging purposes, and as an execution log for analysis.The bottom of the figure shows the Twitter web console, showing the emitted tweet with the partial results (left), and a private message confirming the vote for a user (right).

While applications can be executed in the development environment, for integration purposes, they can also be deployed on a server. For this purpose, the tool generates skeleton service interfaces from the external data dependencies, which need to be completed manually. We consider both REST and SOAP web services, but this feature is currently under development.

7 Related work

This section compares related work along several axes: (i) approaches to build social applications over OSNs, (ii) analysis of social interactions in OSNs, (iii)

applications for OSNs, built manually, (iv) processing posts, and (v) approaches to create specific kinds of social applications, e.g. for crowsourcing.

(i) Tools for creating social applications. As we have seen, a social application uses the infrastructure of OSNs to enhance their reach and dissemination [13]. We find several approaches to build such kind of applications: OSN-specific, multi-channel, or based on web-engineering systems.

In the first kind, in [13] the authors propose a flow metaphor to concatenate actions to be performed on resources of an OSN, like posting a comment, or uploading a photo. Similar to ours, that framework is amenable for non-experts, as it abstracts from low level programming tasks based on APIs. However, we are directed to tweet-based applications, having a dedicated DSL to detect patterns, and we allow the connection with external sources.

Concerning multi-channel approaches, IFTTT[2] is a mashup approach to construct simple "recipes" to automate tasks from different *channels* (like Facebook or Twitter), like "tweet my facebook status updates". In a similar vein, in Zapier[3] users write graphical transformation rules between different web services, in which the input data may come from a general-purpose web service or OSN, and the output data can be another web service. Compared to our work, we enable the definition of more complex tasks, involving e.g., queries, and we make available a dedicated DSL for pattern-match and consider mechanisms for connection with other sources, hence enabling the integration of existing information systems with social applications. A more advanced way to integrate web services is proposed by the MuleSoft commercial tool[4], however it is a heavyweight solution, requiring from expert knowledge in software architecture and design.

Finally, BPM4People [5] extends the BPM language for enabling to model complex data flow, coming from a social domain. The approach is integrated with WebRatio[5]. This is useful because, ever more frequently, web applications need to be extended with features enabling their interaction with OSNs. For this purpose, in [6] an extension of WebML is proposed, to incorporate social primitives, permitting either cross-platform operations (login and search) applicable to several social networks, and specific actions, e.g., to send a tweet.

All these tools differ from our approach in the level of expertise required from the user, since being generic web-engineering tools, they are generally expert-user, or IT staff, oriented. Our proposal enables non-expert - or with a really short training - users to collect data from OSNs immediately for using in it in a domain-specific field that they can define by themselves. This is facilitated by our use of DSLs specific for OSN applications (for pattern definition, actions).

(ii) Analysis of social interactions in OSNs. Twitter has been used in numerous recent academic works, studying its network structure [12, 24], or its use as social medium for communication. As an example of the latter, in [23] a classifier was developed to detect messages contributing to situational aware-

[2] `https://ifttt.com/`
[3] `http://zapier.com/`
[4] `http://mulesoft.com/`
[5] `http://www.webratio.com/`

ness, using a combination of manually annotated and automatically-extracted linguistic features. In [15] an analysis of tweet content and retweet behaviour during the Fukushima nuclear disaster is performed. In our case, our goal is to detect simple patterns in text, so for now we did not use sophisticated natural language processing (NLP) techniques. We believe frameworks similar to ours could be very valuable as a basis to build this kind of applications.

(iii) Applications for OSNs. Some works aim at using Twitter to create new applications, but they are normally built manually with no automated support from frameworks like ours. This has the drawback that developers need to deal with complicated programming concepts, directly manipulating the OSN API. For example, in [20], tweets are used to detect earthquakes in Japan, by classifying tweets according to whether they convey the occurrence or not of an earthquake. While this application was built ad-hoc, it could have benefited from an automated framework, like ours, for their construction.

(iv) Processing posts. Tweets contain unstructured information, and in order to make them computer-processable, some works [22, 8] have proposed the incorporation of structured information in tweets, while others use NLP techniques. In the first group of works, in [22], a simple workflow language is proposed, which combines SOA principles within Twitter. Therefore, SOA primitives, e.g., for service discovery or service binding are embedded in tweets, and Twitter is used as a means to reuse existing infrastructure. In this way, tweets may be used to invoke services and coordinate crowd-sourced activities that are required to fulfill a certain task. However, tweets need to follow a strict syntax, and are not suitable for a natural human-machine or machine-human communication. HyperTwitter [8] proposes the use of Twitter for collaborative knowledge engineering. For this purpose, it defines RDF-like syntaxes to be detected on tweets, which are converted into RDF statements. As our goal is not knowledge engineering, but building tweet-based applications, we support the definition of a richer syntax for tweets, and also provide the machinery for the rapid creation of tweet-based applications, as well as an advanced MDE-based developing environment.

Regarding the use of NLP, EquatorNLP [7] uses deep NLP and machine learning to extract relevant facts of posts in social networks during an emergency situation; and [14] proposes an hybrid technique to recognise named entities. Other works, like [2] are concerned with efficient processing of streams of tweets. We will consider in future work the possibility to integrate some of these techniques in our framework. Tweets are limited to 140 characters, and Twitter users tend to use slang, or abbreviations. Hence, in order to use NLP techniques in unrestricted domains, normalization techniques, like those in [17], are targeted to ammend words with missing vowels. In our case, the recognision task is much simpler, because we define patterns a priori, and enable variations of their concepts (e.g., missing vowels, or distinct letter case).

(v) Crowdsourcing applications. There are approaches to facilitate the construction of crowdsourcing (or human-computation) applications [10, 4, 1] a kind of social application, based on the distribution of tasks among a high number of human participants. For example, *we flow* [10] is an approach to facilitate

the creation of human computation applications, based on the availability of a coordination language, and a generator that synthesizes a collaborative web application from such specification. Similar to our work, *We Flow* is directed to empower users with the ability to create their own applications. However, we use the OSN capabilities of Twitter, make available DSLs, and enable the connection with existing information systems. In [4] a model-based approach to systematize the definition of crowdsourcing applications is proposed. It is based on modelling task types and the interaction with performers, so that the system guarantees certain properties. For some applications, their models could be compiled to our DSLs to obtain a Twitter-based crowdsourcing applications.

Altogether, we are witnessing an increasing interest in both, the construction of all sorts of social applications, and in the analysis of the interactions produced in ODNs. We believe that an MDE framework, like ours would greatly help in these two aspects, as otherwise expert programmers are needed to deal with the intricacies of the OSN API and pattern detection.

8 Conclusions and future work

In this paper, we have introduced the concept of post-based applications: applications whose inputs and outputs are extracted and produced from OSNs, like Twitter. We have shown some scenarios where those applications are useful, and demonstrated the feasibility of their construction through an MDE approach. We have presented a prototype realization, targeting Twitter.

Even though we target short messages, we would like to increase the expressiveness of our pattern DSL, considering more advanced NLP techniques for pattern match. We are currently improving our action language with new primitives (e.g., for presenting outputs in maps or in charts) and taking inspiration from data-stream systems for tweet querying. We are currenlty working on improving our tool, in particular the deployment mode, and considering a web-based version of the tool. We are integrating other OSNs, in addition to Twitter, which would allow from inter-platform applications. Finally, we are designing more specific languages for certain applications on top of our DSLs, like for mobile learning.

Acknowledgements. Work supported by the Spanish Ministry of Economy and Competitivity with project Go-Lite (TIN2011-24139).

References

1. S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. The jabberwocky programming environment for structured social computing. In *UIST*, pages 53–64. ACM, 2011.
2. N. Asadi and J. Lin. Fast candidate generation for real-time tweet search with bloom filter chains. *ACM Trans. Inf. Syst.*, 31(3):13, 2013.
3. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.
4. A. Bozzon, M. Brambilla, S. Ceri, and A. Mauri. Reactive crowdsourcing. In *Proc. 22nd Int. Conf. on World Wide Web*, pages 153–164, 2013.

5. BPMN4People. `http://www.bpm4people.org`.
6. M. Brambilla and A. Mauri. Model-driven development of social network enabled applications with webml and social primitives. In *ICWE Worksh.*, volume 7703 of *LNCS*, pages 41–55. Springer, 2012.
7. L. Döhling and U. Leser. EquatorNLP: Pattern-based information extraction for disaster response. In *Foundations, Technologies and Applications of the Geospatial Web*, pages 127–138, 2011.
8. M. Hepp. Hypertwitter: Collaborative knowledge engineering via twitter messages. In *EKAW*, volume 6317 of *LNCS*, pages 451–461. Springer, 2010.
9. P. Hyman. 'Peace technologies' enable eyewitness reporting when disasters strike. *Commun. ACM*, 57(1):27–29, 2014.
10. N. Kokciyan, S. M. Üsküdarli, and T. B. Dinesh. User generated human computation applications. In *SocialCom/PASSAT*, pages 593–598. IEEE, 2012.
11. J. Krämer and B. Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM Trans. Database Syst.*, 34(1), 2009.
12. S. Kumar, F. Morstatter, and H. Liu. *Twitter Data Analytics*. SpringerBriefs in Computer Science. Springer, 2014.
13. J. Laconich, F. Daniel, F. Casati, and M. Marchese. From a simple flow to social applications. In *ICWE Wsh.*, volume 8295 of *LNCS*, pages 39–50. Springer, 2013.
14. C. Li, A. Sun, J. Weng, and Q. He. Exploiting hybrid contexts for tweet segmentation. In *SIGIR*, pages 523–532. ACM, 2013.
15. J. Li, A. Vishwanath, and H. R. Rao. Retweeting the Fukushima nuclear radiation disaster. *Commun. ACM*, 57(1):78–85, Jan. 2014.
16. G. A. Miller. Wordnet: A lexical database for english. *CACM*, 38(11):39–41, 1995.
17. J. Porta and J.-L. Sancho. Word normalization in twitter using finite-state transducers. In *Tweet-Norm@SEPLN*, volume 1086 of *CEUR*, pages 49–53, 2013.
18. E. Qualman. *Socialnomics: How Social Media Transforms the Way We Live and Do Business, 2nd edition*. Wiley, 2012.
19. Reactive manifesto. `http://www.reactivemanifesto.org`.
20. T. Sakaki, M. Okazaki, and Y. Matsuo. Tweet analysis for real-time event detection and earthquake reporting system development. *IEEE Trans. Knowl. Data Eng.*, 25(4):919–931, 2013.
21. A. M. Segura, J. de Lara, and J. S. Cuadrado. Twiagle: a tool for engineering applications based on instant messaging over Twitter. In *ICWE'14*, volume In Press of *LNCS*, pages 1–4. Springer, 2014.
22. M. Treiber, D. Schall, S. Dustdar, and C. Scherling. Tweetflows: Flexible workflows with twitter. In *PESOS '11*, pages 1–7. ACM, 2011.
23. S. Verma, S. Vieweg, W. . Corvey, L. Palen, J. Martin, M. Palmer, A. Schram, and K. Anderson. Natural language processing to the rescue? extracting "situational awareness" tweets during mass emergency. In *ICWSM*. The AAAI Press, 2011.
24. F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.